جيرارد سوينو
هشام رزق الله

تعلم البرمجة بالبايثون 3

مجتمع لينكس العربي

النسخة الألكترونية من هذه النصوص يمكنك الحصول عليه مجانا و بحرية من :
http://inforef.be/swi/python.htm

بعض الفقرات من هذا الكتاب قد تم تكييفها لكتاب :

How to think like a computer scientist

بواسطة : *Allen B. Downey* و *Jeffrey Elkner* و *Chris Meyers*

متاحة على : http://thinkpython.com

أو : http://www.openbookproject.net/thinkCSpy

جميع الحقوق محفوظة لجيرالد سوينو (2000 - 2012)

يتم توزيع هذا الكتاب وفق لرخصة Creative Commons "الإسناد, الإستخدام الغير التجاري, المشاركة بالمثل 2.0 , فرنسا" .

هذا يعني أنك تستطيع نسخ و تعديل و إعادة توزيع هذه الصفحات بحرية تامة, شرط أن تتبع عدد من القواعد هذا الترخيص .

في الأساس, إعلم أنك لا يمكنك الحصول على ملكية هذا النص و إعادة توزيعه (معدل او غير معدل) بتعريف بنفسك حقوق تأليف و نشر أخرى . المستند الذي قمت بإعادة توزيعه, معدل أو غير معدل, يجب أن يتضمن النص الكامل للرخصة أعلاه و هذا الإشعار و التمهيد الذي ياتي بعده . الوصول إلى هذه الملاحظات يجب أن يبقى حر للجميع . يمكنك طلب مساهمة مالية لتوزيع هذه الملاحظات, لكن المبلغ المطلوب يجب أن يرتبط بتكاليف الإستنساخ . لا يمكنك إعادة توزيع هذه الملاحظات و جعلها بحقوق تأليفك و نشرك, و لا يمكنك ان تحد من حقوق الإستنساخ للنسخ التي قمت بتوزيعها . التوزيع التجاري لهذا النص في شكل مطبوع, في شكله الكلاسيكي لدليل مطبوع, هو محجوز لدار النشر Eyrolles (باريس) .

غراس هوبر, مخترع المترجم :

"بالنسبة لي, البرمجة هي أكثر من فن تطبيقي مهم . و هي أيضا مسعى طموح قادة إلى أعماق المعرفة ."

إلى أليكسان و أوغيستن و لوسيل و أليس و ماكسيلين

تمهيد

مثل المعلم الذي يدرس البرمجة بالتوازي مع التخصصات الأخرى, أعتقد أنني يمكن أن أقول أن هذه هي الوسيلة مفيدة للغاية للتعلم من أجل تدريب شباب القيمة الفكرية المتساوية, إن لم يكن متفوقا, في بعض التخصصات الكلاسيكية مثل اللاتينية .

فكرة عظيمةإذا, لذا أقترح أن يتم تعلم هذا في بعض قطاعات بما في ذلك التعليم الثانوي . دعونا نكون واضحين : ليس من مبكرا تدريب المبرمجين لمهن في المستقبل . نعتقد ببساطة أن تعلم البرمجة لديها مكانتها في تعليم العام عند الشباب (أو على الأقل البعض منهم), لأنها مدرسة غير عادية من الصرامة و المنطق, و حتى الشجاعة .

في الأصل, كتب هذا الكتاب للطلاب الذين أخذوا دورة " البرمجة و اللغات" لخيار "العلوم و المعلوماتية" الدرجة الثالثة للتعليم الثانوي في بلجيكا . و يبدو أن هذه الدورة قد تكون مناسبة لأي شخص لم يسبق له أن برمج من قبل, و لكن يريد تعلم هذا التخصص بنفسه .

نقترح عملية تعلم غير خطية, و التي هي بالتأكيد موضع تساؤل . و نحن ندرك أنه سوف يظهر بعض الفوضى في نظر بعض الأصوليين, و لكننا نريد هذا لأننا مقتنعون بأن هنالك العديد من الطرق للتعلم (و ليس فقط البرمجة), و يجب علينا أن نتقبل على الفور أن مختلف الأفراد لا تستوعب نفس المفاهيم في نفس الترتيب . و لذلك سعينا قبل كل شيئ إثارة الإهتمام و فتح أكبر عدد من الأبواب, و نحن نسعى إلى مراعاة الإرشادات التالية :

- التعليم الذي نسعى إليه عام : نحن نريد تسليط الضوء على ثوابت البرمجة و الحاسوبية, دون أن تكون ذا إختصاص, أو أن القارئ لديه قدرة فكرية إستثنائية .
- يجب أن تكون الأدوات المستخدمة عند التعلم حديثة و فعالة, و لو كان يجب شراؤها لكن يجب أن تكون قانونية و بأسعار منخفضة للإستخدام الشخصي . و عنواننا في الواقع هو الأولوية للطلاب, و كل

خطواتنا للتعلم تهدف إلى إعطائهم فرصة لبدء العمل في أقرب وقت ممكن لإنجاز إنجازاتهم الشخصية التي يمكن أن يطوروها و يستخدموها في أوقات فراغهم .

- و سوف نناقش برمجة واجهة المستخدم الرسومية في وقت مبكر, حتى قبل أن نتعرف على جميع هياكل البيانات المتاحة, لأن هذا النوع من البرمجة يشكل تحديا في نظر المبرمج المبتدئ . نلاحظ أيضا أن الشباب الأن الذين في صفوفنا الدراسية "يسبحون" بالفعل في ثقافة الحاسوبية بأساس النوافذ و غيرها من الكائنات التفاعلية الرسومية . فإذا إختاروا تعلم البرمجة, فهم سيكونون حريصين بالضرورة لإنشاء تطبيقات بأنفسهم (ربما بسيطة جدا) حيث نظرة الرسومية موجودة . لقد إخترنا هذا النهج الغير عادي قليلا لنسمح للقارئ أن يبدأ مبكرا جدا في صناعة مشاريعه الشخصية الصغيرة الجذابة, و مع ذلك, نترك بيئات البرمجة المتطورة التي تكتب تلقائيا أسطر عديدة من الكود على الجانب عمدا, لأننا لا نريد أن نخفي التعقيد الكامن .

بعض الأشخاص ينتقدون أن نهجنا لا يركز بما فيه الكفاية على خوارزميات البساطة و الوضوح . نحن نعتقد أنه أصبح أقل أهمية مما كان عليه في الماضي . تعلم البرمجة الحديثة يتطلب كائنات تتواصل في أقرب وقت ممكن مع الكائنات و المكتبات للصنف الحالي . و هكذا يجب أن يعرف في وقت مبكر على التفكير من حيث التفاعلات بين الكائنات, مثل إجراءات بناء, و التي تأذن بها بالسرعة الكافية للإستفادة من المفاهيم المقدمة, مثل الميراث, و التجسيد متعدد الأشكال .

و لقد قمنا أيضا بتوفير مكان كبير لما فيه الكفاية للتعامل مع أنواع آخرى من هياكل البيانات, لأننا نعتقد أن هذا إنعاكس على البيانات يجب عليه أن يظل العمود الفقري لأي تطوير للبرمجيات .

إختيار لغة البرمجة الأولى

هنالك عدد كبير من لغات البرمجة, و لكل منها مزاياه و عيوبه . يجب علينا أن نختار لغة واحدة . عندما بدأنا بالتفكير في هذه المسألة خلال إعدادنا لمنهاج جديد لخيار العلوم و المعلوماتية, لقد تراكمت خبراتنا الشخصية الطويلة في البرمجة ب Visual Basic (ميكروسوفت) و في كلاريون (Topspeed) . و لقد جربنا أيضا قليلا من دلفي (Borland) . و لذلك كان من الطبيعي أننا إستخدمنا في البداية لغة واحدة أو أكثر من هذه اللغات . و كان لهذه اللغات إذا أردنا إستخدامها كأدوات أساسية لتعلم البرمجة العامة إثنين من العوائق الرئيسية :

- ترتبط ببيئات برمجية (معناها برامج) خاصة . و هذا يعني أنه ليس فقط على المدرسة يجب أن تكون على إستعداد لشراء ترخيص لإستخدام مثل هذه البرامج لكل محطة عمل (و التي يمكن أن تكون مكلفة), و لكن حتى بالنسبة للطلاب الذين يرغبون في إستخدام مهارات البرمجية في أماكن أخرى خارج المدرسة, و هذا الأمر لا يمكن أن نتقبله . و ثمة عيب آخر خطير و هو أن هذه المنتجات تحتوي على "صناديق سوداء" أي أننا لا نستطيع أن نعرف محتواها, و وثائقها ستكون ناقصة و غير مؤكدة .
- هذه اللغات مرتبطة بنظام تشغيل واحد و هو ويندوز . فهي ليست "محمولة" على أنظمة التشغيل أخرى (يونكس, ماك, إلخ) . و هذا لا يتناسب مع مشروعنا التعليمي الذي يهدف إلى تعليم عام (و بالتالي متنوع) الذي سيتم تسليط الضوء على ثوابت الحاسوبية إلى أقصى حد ممكن .

قررنا بعد ذلك دراسة العرض البديل, و هذا معناه اللغات المقترحة مجانا من قبل حركة البرمجيات الحرة . وجدنا أننا كنا متحمسين : ليس لأنه يوجد في عالم المصادر المفتوحة مفسرات و مترجمات مجانية لمجموعة كبيرة من اللغات, و لكن لأن هذه اللغات حديثة و ذات كفاءة عالية و محمولة (و هذا معناه أنها تستخدم على أنظمة تشغيل مختلفة مثل ويندوز و لينكس و ماك), و هي موثقة توثيقا جيدا .

اللغات السائدة هي بلا شك السي و السي بلس بلس . هذه اللغة تفرض نفسها المرجع المطلق, و كل خبير حاسوب سوف يتعلمها عاجلا أم آجلا . و لكن للأسف هذه اللغة شاقة و معقدة جدا, وقريبة من الحاسوب . و تركيب جملها قليل القابل للقراءة و قوي الربط . و إن البرامج الكبيرة المكتوبة بلغة السي أو السي بلس بلس طويلة و مؤلمة . (و ينطبق نفس الشيئ على لغة جافا .)

من ناحية أخرى, فإن الممارسة الحديثة لهذه اللغة تستخدم على نطاق واسع مولدات التطبيقات و الأدوات الدعم المتطورة الأخرى مثل C++Builder و Kdevelop إلخ . يمكن لهذه البيئات أن تكون فعالة جدا في أيدي المبرمجين ذوي خبرة, لكنها تقدم العديد من الأدوات المعقدة كثيرا جدا, و هي صعبة على المستخدم المبتدء و الذي من الواضح لا يتقنها . و لذلك سيكون في نظره أنه قد يخفي الآليات الأساسية للغة نفسها . سوف نترك السي و سي بلس بلس لوقت لاحق .

في بداية تعلمنا البرمجة, يبدو من الأفضل أن نستخدم لغة عالية المستوى, و أقل تقييد, و تكوين الجمل أكثر قابلية للقراءة . بعد أن فحصنا و واجهنا عدة لغات مثل Perl و Tcl/tk, قررنا أخيرا أن نعتمد على البايثون, لغة حديثة و شعبيتها متزايدة .

تقديم لغة البايثون

> هذا النص لـ "ستيفان فرميجيا" المؤرخ قليلا, و لكن لا يزال ذا صلة للنص الأساسي . و تم نقل مقال من مجلة « !*Programmez*" عدد شهر ديسمبر/كانون الأول من سنة 1998 . كما أنه متاح على *http://www.linux-center.org/articles/9812/python.html* . و "ستيفان فرميجيا" هو مؤسس مشارك لـ *AFUL* (الرابطة الفرنسية لمستخدمي لينكس و البرمجيات الحرة) .

لغة البايثون هي لغة محمولة و حيوية(ديناميكية) و مجانية و موسعة, التي تسمح (و لكنها لا تتطلب ذلك) بإتباع نهج الوحدات و البرمجة الشيئية(OOP) . تم تطوير لغة البايثون سنة 1989 من قبل غيدو فان روسم و عدد كبير من المتطوعين و المساهمين .

مميزات اللغة

سوف نقوم بوضع المميزات الرئيسية للبايثون مع بعض التفصيلها :

- لغة البايثون لغة محمولة, و ليس فقط على مختلف أنظمة يونكس, و لكن حتى أنظمة تشغيل : ماك و BeOS و NexTStep و MS-DOS و مختلف إصدارات ويندوز . و هنالك مترجم جديد, يدعى JPython, تم كتابته بالجافا و يولد كودبايت جافا .
- البايثون مجانية, و لكن يمكنك إستخدامها في المشاريع التجارية دون قيود .
- البايثون مناسبة لسكريبتات من 10 سطر إلى المشاريع المعقدة التي تحتوي على عشرات الآلاف من الأسطر .
- تكوين جمل البايثون بسيط جدا, و يعمل جنبا إلى جنب مع أنواع البيانات المتقدمة (القوائم و القواميس ..), و التي تصنع برامج مدمجة جدا و قابلة للقراءة . و للمقارنة, برنامج البايثون غالبا ما يكون أقصر من 3 إلى 5 مرات من برنامج السي أو سي بلس بلس (أو حتى الجافا) أو ما يعادلها, و وقت تطوير من 5 إلى 10 مرات أقصر و سهل جدا في الصيانة .

- البايثون تدير الموارد بنفسها (الذاكرة, واصفات الملفات) دون تدخل من قبل مبرمج عن طريق آلية عد المراجع (مماثلة, لكن مختلفة, هواة جمع القمامة) .
- لا توجد مؤشرات واضحة في البايثون .
- البايثون هي متعدد الخيوط (إختياري) .
- البايثون تدعم البرمجة الشيئية, و هي تدعم الوراثة المتعددة و مشغلات الحمولة الزائدة . في نماذج الكائنات, و عن طريق إتخاذ مصطلحات السي بلس بلس (جميع الأساليب إفتراضية) .
- البايثون تدعم (مثل الجافا أو الإصدارات الأخيرة من السي بلس بلس) نظام الإستثناءات, الذي يسمح بتبسيط معالجة الأخطاء بشكل كبير .
- البايثون حيوية(ديناميكية) (المفسر يمكنه تقييم السلاسل النصية التي تمثل عبارات أو تعليمات البايثون)و متعامد (عدد قليل من المفاهيم كافية لتوليد بنى غنية)و الإنعكاسية (و هي تدعم الميتابروغراميك, على سبيل المثال, يستطيع الكائن إضافة أو إزالة سمات أو أساليب أو حتى تغيير صنف قيد التنفيذ), و الإستقراء (عدد كبير من أدوات التطوير, مثل المصحح أو المحلل, موجودة في البايثون نفسها) .
- مثل Scheme أو SmallTalk, يتم كتابة البايثون حيويا . جميع الكائنات التي تم معالجها من قبل المبرمج يتم تعريف نوع واضح عند التشغيل, و الذي لا يحتاج إلى أن تعلن نوعه مسبقا .
- البايثون حاليا هي تطبيقان . الأول, و هو المفسر, حيث سيتم تجميع برامج البايثون في تعليمات محمولة, ثم يتم تشغيلها بواسطة آلة إفتراضية (مثل الجافا, مع فارق مهم : يتم كتابة الجافا بشكل ثابت, و يصبح من السهولة تسريع تشغيل برنامج جافا أسرع من البايثون) . و الثاني يولد مباشرة كودبايت جافا .
- البايثون لغة موسعة : مثل Tcl و Guile, أي أننا يمكننا بسهولة التعامل مع مكتبات السي الموجودة . و يمكننا أيضا أن نستخدمها كلغة موسعة لأنظمة برامج تمديد معقدة .
- إن المكتبة البايثون القياسية, و حزم المساهمة, توفر لك الوصول إلى مجموعة واسعة من الخدمات : سلاسل نصية و تعابير عادية, و معايير خدمات اليونكس(ملفات, sockets, الخيوط ... إلخ), البروتوكولات الإنترنات (ويب, الأخبار, FTP و CGI و HTML), و قواعد البيانات و واجهات المستخدم الرسومية .

- البايثون هي لغة لاتزال تتطور, بدعم من مجتمع المستخدمين و المديرين المتحمسين, و معظمهم من أنصار البرمجيات الحرة . بالتوازي مع المفسر الرئيسي, المكتوب بلغة السي و هي اللغة التي تم صنع بها البايثون, و مفسر ثاني, مكتوب بالجافا, و هو قيد التطوير .
- و أخير, البايثون هي لغة الإختيار لمعالجة الـ XML .

للأستاذ الذي يريد إستخدام هذا الكتاب كدعم لدروسه

نحن نأمل مع هذه الملاحظات فتح أكبر عدد ممكن من الأبواب . على مستوى التعليم لدينا, يبدو من المهم إظهار أن برمجة الحاسوب هو عالم واسع من المفاهيم و الأساليب, التي يمكن لكل شخص أن يجد مجاله المفضل . نحن لا نعتقد أن جميع الطلاب يجب أن يتعلموا بالضبط نفس الأشياء . نحن نريد أن يكونوا قادرين على تطوير مهاراتهم في مشاريع فردية تختلف إلى حد ما, و التي تسمح لهم بتطوير برامجهم الخاصة و برامج أقرانهم, و كذلك المساهمة عندما يقترح أحدهم التعاون لعمل كبير .

و على أي حال, يجب أن يكون عملنا الرئيسي إثارة الإهتمام, للذي لا يزال بعيدا عن تحصيل حاصل لمادة صعبة مثل برمجة الحاسوب . نحن لا ندعي الإعتقاد بأننا سوف نحمس الشباب على الفور لبناء خوارزميات جميلة . نحن مقتنعون تماما أنه سيتم تثبيت مصلحة عامة بمجرد أنهم يشعرون بأنهم أصبحوا قادرين على تطوير مشاريعهم الشخصية, بقدر معين من الإستقلال الذاتي .

و من هذه الإعتبارات التي أدت بنا إلى تطوير هيكل دراسي و الذي يعتقد البعض أن به القليل من الفوضى . سوف نبدأ مع سلسلة من الفصول القصيرة جدا لفترة وجيزة التي تفسر ما نشاط البرمجة و تشكل الأساسات القليلة التي لا غنى عنها لتحقيق برامج صغيرة . قد يعتقدون أنه من المبكر البدء بالمكتبات الكائنات الرسومية, على سبيل المثال, واجهات المستخدم الرسومية tkinter, بحيث يصبح مفهوم الكائن مألوفا لديهم . ينبغي علينا أن نكون جذابين بما فيه الكفاية للذين يشعرون أنهم إكتسبوا بالفعل إتقان معين . و نود حقا أن يتمكن الطلاب من برمجة تطبيق GUI (واجهة المستخدم الرسومية) صغير في نهاية السنة الأولى من الدراسة .

بشكل ملموس جدا, هذا يعني أننا نتوقع إستكشاف أول ثمانية فصول من هذه الملاحظات خلال السنة الأولى من الدورة . و هذا يعني أنا سنتناول أولا مجموعة من المفاهيم الهامة (انواع البيانات و المتغيرات و التعليمات التحكم في تدفق و الدالات و الحلقات) بصورة سريعة و دون الحاجة إلى القلق كثيرا على ما يفهم

تماما من كل مفهوم قبل الإنتقال إلى مفهوم آخر, بدل من محاولة غرس الذوق الشخصي في البحوث و التجارب . و غالبا ما سيكون أكثر كفاءة لإعادة شرح مفاهيم و الآليات المطلوبة في وقت لاحق في حالات و سياقات متنوعة .

في السنة الثانية سوف نسعى إلى تنظيم المعرفة و تعميقها . و سوف يتم تشريح و مناقشة الخوارزميات . و سوف نناقش المشاريع و المواصفات و أساليب التحليل . و نحن نطلب منك دفتر ملاحظات لكتابة تقارير تقنية على وظائف معينة .

و الهدف النهائي لكل طالب هو إكمال مشروع برمجي له بعض الأهمية . و سنعمل جاهدين لإنهاء مفاهيم الأساسية النظرية الكافية في وقت مبكر من السنة الدراسية. بحيث يستطيع أي شخص لديه وقت فراغ صنع مشروع .

يجب أن يفهم أن المعلومات المتوفرة في هذه الملاحظات تحتوي على مجموعة واسعة من المجالات (إدارة واجهات المستخدم الرسومية و الإتصالات و قواعد البيانات و إلخ .) الإختيارية . و هذه سوى سلسلة من الإقتراحات و المعايير التي أدرجناها لمساعدة الطلاب على إختيار و بدأ مشاريع الشخصية للتخرج . نحن لا نسعى بأي شكل من الأشكال تدريب متخصصين في لغة معينة أو في مجال تقنية معين : نحن نريد ببساطة إعطاء لمحة عن الفرص الهائلة لولئك الذين يأخذون معنات لتعلم البرمجة بمهارة .

إصدارات اللغة

لغة البايثون لاتزال تتطور, لكن الهدف من هذا التطور هو تحسين أو ترقية المنتج . و يجب تعديل برامج للتكيف مع النسخ الجديدة التي من شأنها أن تصبح غير متوافقة مع تلك السابقة . و الأمثلة في هذ الكتاب تطورت على مدى فترة طويلة نسبيا من الزمن : بعض تم تطويره بايثون 1.5.2 ,ثم بايثون 1.6 و 2.0 و 2.1 و 2.2 و 2.3 و 2.4 إلخ . و إنهم بحاجة إلى تغيير قبل أن يتكيفوا لبايثون 3 .

هذا الإصدار الجديد من اللغة, يحمل بعض التغييرات الفنية التي تعطي المزيد من التماسك و سهولة أكبر للإستخدام, و لكن هنالك حاجة إلى تحديث صغير لكافة السكريبتات المكتوبة للإصدارات السابقة . و قد تم إعادة تصميم النسخة الحالية من هذا الكتاب و ليس فقط للتكيف مع أمثلة الإصدار الجديد, و لكن للإستفادة أيضا من هذه التحسينات, و التي هي على الأرجح أفضل وسيلة لتعلم البرمجة اليوم .

إذا قم بتثبيت أحدث إصدار بايثون متاح على نظام التشغيل الخاص بك(بعض الأمثلة لدينا تتطلب الإصدار 3.1 أو أحدث), و إستمتع ! و لكن, إذا كنت بحاجة إلى تحليل سكريبتات المقدمة للإصدار السابق, لاحظ وجود أدوات تحويل (أنظر خاصة للسكريبت 2to3.py). و التي موجودة على الإنترنات في موقعنا http://inforef.be/swi/python.htm للإصدار السابق من هذا النص, و التي تم تكييفها للإصدارات السابقة من البايثون, و دائما تستطيع التحميل بحرية .

توزيع البايثون و المراجع

الإصدارات المختلفة من البايثون (لويندوز و يونكس إلخ .), و الدرس التعليمي الأصلي و الدليل المرجعي و وثائق مكتبات الدالات و إلخ . متوفرة للتحميل مجانا من الإنترنات, من الموقع الرسمي :
http://www.python.org .

أمثلة الكتاب

يمكنك تحميل الكود المصدري من أمثلة هذا الكتاب من خلال موقع الكاتب :

http://inforef.be/swi/python.htm

أو على العنوان التالي :

http://infos.pythomium.net/download/cours_python.zip

و كذلك على شكل كتاب ورقي مطبوع :

http://www.editions-eyrolles.com

شكر

هذا الكتاب هو نتيجة لبعض الأعمال الشخصية, و لكنها "أكبر من ذلك بكثير",فقد تم تجميع المعلومات و الأفكار المتوفرة لجميع المتطوعين من أساتذة و باحثين .

مصدر المسودات التي إستلهمت منها هو كتاب ل A.Downey, J.Elkner & C.Meyers : How to think like a computer scientist (http://greenteapress.com/thinkpython/thinkCSpy). أشكر مجددا هؤلاء الأساتذة المتحمسين . أعترف أيضا أنني إتخذت إلهام من برنامج تعليمي كتب بواسطة غيدو فان روسم نفسه (مؤلف لغة البايثون). و كذلك الأمثلة و الوثائق المختلفة من مجتمع مستخدمي البايثون . و لكن للأسف لا يمكن حصر مراجع كل هذه النصوص, و لكن أريد أن أعبر عن إمتناني لهم .

و شكرا لكل من شارك في تطوير البايثون و ملحقاته و وثائقه بدءا من غيدو فان روسم بالطبع, و لا ننسى الآخرين, للأسف لا أستطيع وضع أسمائهم كلها هنا .

أشكر مرة أخرى زملائي Freddy Klich و David Carrera, أساتذة في معهد القديس يوحنا Berchmans de Liège, الذين وافقوا على المغامرة هذا المسار الجديد مع طلابهم, و إقتراح العديد من التحسينات أيضا . و شكرا خاص ل Christophe Morvan, أستاذ في IUT في مان لا فاليه, لنصائحه القيمة و التشجيعه, و إلى Robert Cordeau, أستاذ في IUT في أورسيه, لنصائحه و شجاعته بتدقيق اللغوي. و شكر كبير ل Florence Leroy محررتي في أورايلي, الذي أصلحت التناقضات بمهارة . و شكرا مرة أخرى لشركائي الحاليين في أورايلي, Muriel Shan Sei Fan, Taï-Marc Le Thanh, Anne-Lise Banéath و Igor Barzilai الذين دعموا بفاعلية هذه الطبعة الجديدة .

و أخير شكر لك سوزل, على صبرها و تفاهمها .

المساهمون

ماديا :

أسامة عقاد

مجلاد السبيعي

ياسر

سطام الحربي

سلطان العنزي

محمد العتيبي

سامي

عبدالله الشاهين

عبد الله الصبي

مساعدة :

محمد أمين

سيف الإسلام البكري

أحمد شريف

صفا الفليج

أنوار بنشقرون

1

في مدرسة السحرة

تعلّم البرمجة شيء مهم في حد ذاته يمكن أن يحفز فضولك الفكري، ليس هذا فحسب بل تعلّمها يفتح أمامك الطريق لإنجاز مشاريع قوية (مفيدة أو مسلّية) والتي ستجعلك في الغالب فخورًا وراضيًا. قبل الدخول في صلب الموضوع سنقترح عليك بعض الملاحظات حول طبيعة البرمجة والسلوك الغريب لممارسيها وشرح بعض مفاهيمها الأساسية. ليس من الصعب تعلم البرمجة لكنها تتطلب منهجًا وقدرًا من المثابرة لأنها علم غير محدود فتواصل التقدم فيه باستمرار .

الصناديق السوداء والتفكير السحري

من الملاحظ في مجتمعنا الحديث أننا نعيش محاطين بالصناديق السوداء بشكل متزايد. من عادة العلماء تحديد أسماء مختلف التقنيات التي نستخدمها بسهولة دون معرفة بنيتها وطريقة عملها بشكل دقيق. مثلا الجميع يعرف كيفية استخدام الهاتف لكن يوجد عدد قليل جدا من التقنيين ذوي درجة عالية من التخصص قادرين على تصميم نموذج جديد.

الصناديق السوداء موجودة في جميع المجالات ومتوفرة للجميع. لا يهمنا هذا عمومًا، بإمكاننا أن نكتفي بمعرفة سطحية لآليتها لاستخدامها دون هواجس . في الحياة اليومية مثلا، لا نهتم فعليًا بمعرفة مكونات البطارية الكهربائية فبمجرد أن نعرف أن البطارية تنتج الكهرباء عن طريق تفاعلات

كيميائية ندرك وبسهولة أنها ستفذ بعد مدة معينة من استخدامها وتصبح بعد ذلك مادة ملوِّثة لا ينبغي رميها في أي مكان، لا حاجة إذن لمعرفة المزيد.

قد يحدث وتصبح بعض الصناديق السوداء معقّدة ولا نستطيع أن نفهمها كفاية لنستخدمها بشكل صحيح تحت أي ظرف, قد نميل إلى التمسك بحججهم ضد التي تربط التفكير السحري, و هذا يعني شكل من أشكال الفكر الذي يشمل على تدخل قوى خارقة للطبيعة أو خصائص لتشرح لعقلنا ما لايمكن أن نفهمه . هذا ما يحدث عندما يظهر ساحر خفيف اليدين, و نحن نميل إلى الإعتقاد بأن له قوة خاصة, مثل تبرع "مشهد مضاعف", أو لقبول وجود اليات خارقة ("السائل المغناطيسي", إلخ), و نحن لا نفهم إستخدامه .

نظرًا لتعقيدها فالحواسيب مثال واضح للصناديق السوداء فحتى لو كنتَ تشعر أنك عشتَ دائما محاطًا بالشاشات ولوحات المفاتيح فغالبًا ليس لديك فكرة كبيرة حول ما يحدث وسط الآلة حقيقة، مثلاً عندما تُحرّك الفأرة سيتحرك على الشاشة وحسب رغبتك رسم صغير يشبه السهم، ما الذي يتحرك بالضبط ؟ هل تشعر أنك قادر على شرحه بالتفصيل دون نسيان (في جملة أمور) المِجَسّات و منافذ الواجهات و الذاكرات و بوابات المقاييس المنطقية و الترانزستورات و البِتات و البايتات و انقطاعات المعالج و شاشات العرض البلوري السائل و شفرة الميكرو و البكسلات و ترميز الألوان... ؟

في زماننا هذا لا يمكن لأحد أن يدعي أنه متمكن من كافة المعارف التقنية والعلمية المستخدمة لتشغيل الحواسيب. عندما نستخدم هذه الآلات فنحن مجبرون على التعامل معها عقليًا أو على الأقل مع أحد أجزائها، كالأغراض السحرية والتي يحق لنا أن نمارس عليها قوة معينة هي الأخرى سحرية.

مثلاً، نفهم جيّدًا تعليمة كتحريك نافذة تطبيق من خلال شريط العنوان ونعرف تمامًا ما يجب فعله لتنفيذها في العالم الحقيقي، كالتعامل مع الأجهزة التقنية (الفأرة ولوحة اللمس ...) التي ستنقل ذبذبات كهربائية من خلال آلية جدّ معقدة ليكون الناتج هو تغيير شفافية أو سطوع جزء من بكسلات الشاشة، لكن في أذهاننا لن تدور أي تساؤلات حول التفاعلات الفيزيائية والدوائر المعقّدة فهذا غرض افتراضي سيتم تفعيله (تحرّك المؤشر على الشاشة) وسيكون بمثابة عصا سحرية للتحكم في الغرض الذي بدوره افتراضي وسحري أيضًا (نافذة التطبيق). التفسير العقلاني لما يحدث فعلاً داخل الجهاز تراجع إذن لصالح إستنتاج متصوّر أكثر سهولة لكنه في الحقيقة مجرّد وهم.

إن كنتَ مهتمًا بالبرمجة فاعلم أنك ستواجه باستمرار أشكال مختلفة من هذا « التفكير السحري » ليس مع الأشخاص الآخرين فحسب (كمَن يَطلب منك إنشاء برنامج معين) بل وأيضًا مع تصوّراتك الذهنية خاصة. يجب عليك أن تزيل هذه الأوهام الزائفة والتي هي في الحقيقة مجرد تخمينات والاعتماد على تفسيرات مجازية مبسطة من الواقع حتى تستطيع تسليط الضوء ولو جزئيًا على الأثار العلمية الحقيقية

وهذا متناقض إلى حد ما ويفسر عنوان هذا الفصل، أي مع تطور مهارتك ستسيطر على الجهاز أكثر وبالتالي ستصبح مع مرور الوقت كساحر في أعين الناس !

أهلا بك إذن في مدرسة السحرة مثل الشهير هاري بوتر !

السحر الأبيض والسحر الأسود

ليست لدينا النيّة بالطبع لمساواة البرمجة بعلم التنجيم وإن كنا نرحب بك هنا كساحر مبتدِأ فهذا فقط لإثارة انتباهك على الأثار المترتبة على هذه الصورة التي قد تعطي نفسك ربما)عن غير قصد لمعاصرتك . قد يكون من المثير للإهتمام ان تقترض بعض الكلمات من المفردات السحر لتوضيح مفاجئة الممارسات .

البرمجة هي فن تلقين جهاز إنجاز مهام جديدة لم يكن قادرًا على تنفيذها سابقًا، تمنحك مزيدًا من السيطرة على كافة الأجهزة المرتبطة بالشبكة وليس على جهازك فحسب. بطريقة أخرى يمكن مساواة هذا بشكل خاص من السحر يُكسب مزاوله قوة، غامضة بالنسبة للكثيرين وحتى مثيرة للقلق عندما ندرك إمكانية استخدماها لأغراض غير شريفة.

في عالم البرمجة نقصد بمصطلح هاكر المبرمجين المحنّكين الذين قاموا بتحسين أنظمة التشغيل الشبيهة بيونكس و وضع تقنيات الإتصال التي كانت أساس التطور المذهل للإنترنت والمستمرين أيضًا في إنتاج وتحسين البرمجيات الحرّة/المصادر المفتوحة، إذن حسب ما سردنا عليه فالهاكرز هم أسياد السحر يمارسون السحر الأبيض.
و لكن هنالك مجموعة أخرى من الناس أطلقت عليهم الصحافة عن طريق الخطأ على أنهم الهاكرز , حين كان يجب عليهم أن يسمونهم كراكرز . و هؤلاء الناس يطلقون على أنفسم إسم الهاكرز لأنهم يريدون أن نعتقد على أنهم مختصون للغاية , و لكن بشكل عام إنهم بالكاد مختصون , و لكنهم ضارين للغاية , لأنهم

يستخدمون معرفتهم للعثور على بعض ثغرات في أنظمة الحاسوب (التي تم صنعها من الهاكرز الحقيقيين) لتنفذ جميع عملياتهم الغير قانونية مثل : سرقة معلومات سرية و الإحتيال , إرسال الرسائل الغير مرغوب فيها (سبام) و الفيروسات و المواد الإباحية و التقليد و تدمير المواقع ...إلخ . طبعا هؤلاء السحرة منحرفين بشكل خطير في السحر الأسود . و هناك أيضا مجموعة أخرى , القراصنة الحقيقيون يسعون إلى تعزيز أخلاقهم التي تقوم أساسا على المحاكاة و تبادل المعلومات (معرفة) والذين يستطيعون بناء برامج تتسم بالكفاءة و هي أيضا بالتأكيد تكون أنيقة و ذات بنية نظامية تماما و لديها وثائقها , ستكتشف بسرعة أن من السهل إنتاج الكثير من البرامج التي تعمل و لكن أغلبها غامضة و مربكة و غير مفهومة لغير مبرمجيها . هذا النوع من البرمجة يكون في الغالب غير واضح و يطلق عليها من قبل الهاكرز بالسحر الأسود .

نهج المبرمج

مثل الساحر , يكون للمبرمج قوة غريبة على سبيل المثال تستطيع تحويل جهاز إلى جهاز اخر و ألة حاسبة إلى الة كتابة أو رسم و قليل من السحر بعد تستطيع تحويل الأمير إلى ضفدع و ذلك بإستخدام لوحة المفاتيح لتدخل بعض التعويذات الغامضة مثل الساحر . و هو قادر على علاج التطبيقات السيئة أو تلقي فترات عن طريق الأنترنات . لكن كيف يكون هذا ممكنا ؟

و قد يبدو هذا متناقضا , كما لاحظنا سابقا , المعلم الحقيقي هو في الواقع الذي لا يؤمن بأي سحر و أي تبرع و لا تدخل خارق . يبقي بارد و عنيد يطلب من المنطق الغير مريح .

تفكير المبرمج يجمع بين البنى الفكرية المعقدة , المماثلة لتلك التي قام بها علماء الرياضيات و المهندسين و العلماء . كما في الرياضيات البرمجة تستخدم لغات رسمية لوصف المنطق (أو الخوارزميات) مثل المهندس , الذي يصنع الأجهزة , فيجمع عناصر لتنفيذ أليات و تقييم أداءها . مثل العالم يلاحظ سلوك النظام المعقد و يصنع النماذج و يختبرها .

النشاط الأساسي للمبرمج هو حل المشاكل

و يجب أن يكون على مستوى عالي من الكفاءة , التي تشمل مختلف المهارات و المعرف و أن يكون قادر على إعادة صياغة المشكلة بطرق عديدة و مختلفة و يجب أن يكون قادر على وضع حلول مبتكرة و فعالة , و قادر على تعبير عن هذه الحلول بوضوع و كمال . كما ذكرنا سابقا , يجب علينا أن نلقي الضوء على الأثار العملية العقلية "سحرية " البسيطة أو الملخصة جدا .

البرمجة هي في الواقع "شرح" بالتفصيل للجهاز ما يجب عليه القيام به , مع العلم أن الجهاز لا يفهم لغة الإنسان , و لكن فقط تنفيذ معالجة الألية لتسلسل من الأحرف ,في الغالب يتم التعبير عن رغبة في تحويل المصدر من حيث " السحرية " إلى منطق صحيح منظم تماما و كل تفاصيلها مفهومة , و هذا ما يسمى بالخوارزمية .

على سبيل المثال أنظر إلى هذه السلسلة من الأرقام بالترتيب : 47,19,23,15,21,36,5,12 ... كيف لنا أن نحصل من الحاسوب على ترتيب هذه الأرقام ؟

الرغبة " السحرية " ليست فقط النقر على الزر , أو إدخال تعليمة واحدة في لوحة المفاتيح ليتم ترتيب الأرقام في مكانه . بل إن عمل المبرمج هو صنع البرنامج لترتيبه و يجب عليه شرح جميع الخطوات لعملية الفرز (في الواقع هنالك طريقة فريدة لهذا أو أن هناك أكثر ؟) و يجب علينا ترجمة جميع الخطوات في سلسلة من التعليمات البسيطة مثلا , على سبيل المثال " قارن أول رقمين , و بدلهما إذا لم يكونا في الترتيب المطلوب , ثم أبدأ مع غيرها ,(الثاني و الثالث ...إلخ) .

إذا كانت التعليمات، وذلك حتى ضوء بسيطة بما فيه الكفاية، يمكن ترميزها في الجهاز وفقا لمجموعة صارمة للغاية من الإتفاقيات المقررة سابقا , و المعروفة بإسم لغة الحاسوب , لفهم هذا يجب توفير جهاز مع وجود برنامج لترجمة هذه التعليمات من خلال ربط كل كلمة مع عمل معين للغة . و هكذا فقط يمكن أن يحقق السحر .

لغة الألة , لغة البرمجة

بالمعنى الدقيق للكلمة , جهاز الحاسوب هو سوى الجهاز الذي ينفذ العمليات بسيطة عن طريق الإشارات كهربائية متسلسلة ,و التي يتم شحنها عن طريق إشارتين لا غير (على سبيل المثال محتمل أقصى أو أدنى جهد) هذه الإشارات المتتالية يستطيع فهمها الحاسوب بالمنطق " كل شيئ أو لا شيئ" و يمكن أن تعتبر تقليدية حسب تسلسل الأرقام أبدأ مع القيمتين 0 و 1 . يطلق عليها بالنظام الرقمي و هي تقتصر على رقمين ثنائيين

لقد عرفنا الأن أن الحاسوب في عملياته الداخلية يتعامل مع الأرقام الثنائية و هو لا يتعامل مع أي شئ أخر . لذا يجب تحويل جميع المعلومات (المشفرة أيضا) إلى الشكل الثنائي . و هذا لا ينطبق فقط على البيانات

التي نود معالجتها (نصوص و صور و أصوات و الأعداد) بل ينطبق حتى على البرامج (التعليمات المتتالية التي ستخبر الحاسوب ماذا يفعل)

و اللغة الوحيدة التي يفهمها الحاسوب هي صعبة بالنسبة لنا , إنها سلسلة طويلة من 1 و 0 (تسمى البت) و يتم التعامل معها على شكل مجموعات (من البايت) 16 أو 32 أو حتى 64 . و هذه اللغة هل غير مفهومة لنا . و للتحدث إلى جهاز الحاسوب, سوف نستخدم أنظمة ترجمة ألية قادرة على تحويل الكلمات التي نفهمه (اللغة الأنكليزية ستكون الأكثر أهمية بالنسبة لنا) إلى سلاسل من الأعداد الثنائية . و يتم إنشاء هذه النظم الترجمة على أساس مجموعة من الإتفاقيات و التي كان من الواضح وجود العديد من الإختلافات . يطلق على نظم الترجمة إسم المترجم أو المفسر حسب الطريقة التي إستخدمتها لتنفيذ الترجمة . ندعو لغات البرمجة أنهم مجموعة من الكلمات الرئيسية (أختيرت تعسفا) مرتبطة مع مجموعة من القواعد المحددة حول كيفية تجميع هذه الكلمات لتصبح عبارات التي يمكن للمفسر أو المترجم ترجمتها إلى لغة الألة (الثنائية) . لكل لغة برمجة مستوى معين فمثلا اللغات ذات مستوى منخفض (على سبيل المثال لغة التجميع) أو ذات مستوى العالي (على سبيل المثال باسكال و بيرل و البايثون ...)اللغات ذات مستوى منخفض تتكون من الإرشادات الأساسية جدا جدا "على مقربة من لغة الجهاز " و اللغات رفيعة المستوى تكون أكثر تجريدا و أكثر "قوة " و بالتالي أكثر "سحر" . و هذا يعني أنه يمكن أن تترجم كل هذه التعليمات من قبل المترجم أو المفسر بعدد كبير من تعليمات الجهاز البدائية . لغة البرمجة التي سنتعلمها أولا هي البايثون , البايثون هي لغة برمجة عالية المستوى , و التي تترجم إلى رمز ثنائي معقد و دائما ما يأخذ بعض الوقت . قد يبدو هذا الوضع يضعف اللغة , لكن الحقيقة أن اللغات ذات مستوى عالي تتميز بمميزات هائلة : إن كتابة برنامج بلغة رفيعة المستوى سهلا كثيرا و يستغرق وقتا أقل لكتابته و إحتمال الأخطاء فيه قليلة و صيانته (معناها تغييرات اللاحقة) و البحث عن أخطاء ("العلل")سهل للغاية . بالإضافة البرنامج المكتوب بلغة عالية المستوى تكون غالبا محمولة هذا يعني أن نغير تغيرات قليلة في البرنامج لتعمل على عدة أجهزة و أنظمة تشغيل مختلفة . و البرنامج المكتوب بلغة ذات مستوى منخفض لا يمكن أن يعمل إلا على نوع واحد فقط و يجب أن يتم إعادة كتابته كاملا ليعمل على جهاز أخر . ما شرحناه في فقرة مختصر : ربما لاحظتم العديد من "الصناديق السوداء " : مترجم و نظام تشغيل و اللغة و تعليمات الجهاز و الكود الثنائي و ما إلى ذلك . إن تعلم البرمجة سوف يسمح لك بفتح جزئيا . و مع ذلك لا يمكنك أن تقوم بفتح جميعها . العديد من كائنات المعلوماتية التي تم صنعها بواسطة من قبل أشخاص آخرين يمكن أن تكون "سحرية" لفترة طويلة (على سبيل المثال, بدءا من لغة البرمجة نفسها) . و يجب عليك أن تثق بأصحابها, و أحيانا سوف تصاب بخيبة أمل من أن النتيجة ليست

دائما ما تستحق الثقة . إذا ابقى يقظا, تعلم كيفية التحقق من الوثائق . في مشاريعك الخاصة, كن حذرا و تجنب إستعمال بأي ثمن "السحر الأسود" (البرامج الممتلئة بالحيل التي أنت فقط تفهمها) : الهاكر الجدير بالثقة ليس لديه ما يخفيه .

تعديل المصدر - المفسر

البرنامج الذي نكتبه في أي لغة برمجة هو نص بسيط, يمكنك البحث عن كل أنواع البرامج الأكثر و الأقل تعقيدا و سوف تجد أنها تنتج فقط نص بسيط و هذا يعني دون تنسيق أو سمة نمط معين (يعني لا مواصفات الخط و يعني لا عناوين و لا خط غامق أو مسطر أو مائل ...إلخ)[1].النص الذي تنتجه يسمى الأن بالشيفرة المصدرية . كما ذكرنا سابقا , لابد من ترجمة تعليمات البرنامج المصدر إلى سلسلة من التعليمات ثنائية مفهومة مباشرة من قبل الجهاز : " شيفرة الكائن في حالة البايثون" و يتم دعم هذه الترجمة بواسطة مفسر تم ترجمته سابقا .هذا الأسلوب الهجين (الذي يستخدم أيضا من قبل لغة جافا) يهدف إلى تعظيم فوائد التفسير و التجميع مع تقليل عيوب كل منها . يرجى منك أن تبحث عن كتاب يشرح هتان التقنيتان في حالة تريد أن تعرف المزيد . أعلم أنه يمكنك صنع برامج عالية الأداء مع البايثون , على الرغم من أنه لا جدال في أن لغة المترجمة بدقة مثل لغة السي يمكنها أن تفعل أفضل من حيث توقيتها .

وضع البرنامج - البحث عن أخطاء (تصحيحها)

البرمجة هي عملية معقدة جدا , كما هو الحال في أي نشاط بشري, في بعض الأحيان قد نقع في أخطاء كثيرة لأسباب عديدة و هي تسمى أخطاء البرمجة "العلل"[2]و جميع التقنيات التي يتم تنفيذها لكشفها و تصحيحها تسمى بالتصحيح في الحقيقة , قد يكون في البرنامج ثلاثة أنواع مختلفة من الأخطاء و ينبغي أن نتعرف عليها لنميزها .

---

[1] و تسمى هذه البرامج برامج معالجة النصوص . على الرغم من أنها توفر مجموعة متنوعة من الأتمتة, و غالبا ما تكون قادرة على إبراز بعض عناصر النص المعالج (تلوين تركيب الجملة, على سبيل المثال), لا تحدث بدقة سوى على النص الغير منسق . فهي مختلفة تماما عن برامج معالجة النصوص, و التي تتمثل مهمته على وجه التحديد تخطيط و تجميل النص مع سمات من أي نوع, بطريقة لجعله قابل للقراءة قدر الإمكان .

[2] bug هي مصطلح أنجيليزي الأصل يستخدم لوصف الحشرات الصغير المزعجة مثل البق . الحواسيب الأولى التي تعمل بمساعدة المصابيح و الراديو التي تتطلب كهرباء ذات فولتية عالية نسبيا, تم حرقها عدة مرات بسبب دخول هذه الحشرات الصغيرة إلى الدوائر المعقدة فتسبب أجسادها دوائر قصير ثم عطب غير مفهوم .

أخطاء التعليمات

يمكن تشغيل برنامج بايثون في حالة كان خاليا من الأخطاء , خلافا لذلك فإن العملية الترجمة ستتوقف و تحصل على رسالة خطأ . "بناء الجملة" هي مصطلح يشير إلى قواعد اللغة التي وضعها مبرمجيها و لقد وضعت ليهكلة البرنامج . كل لغة لديها بناء لغة فمثلا في الفرنسية على سبيل المثال يجب على الجملة أن تبدأ بحرف كبير و تنتهي بنقطة . و هذه الجمل لديها خطأين في بناء الجملة . في النص العادي , وجود بعض الأخطاء في بناء الجملة هنا و هنالك لا يهم . قد يحدث هذا (في الشعر مثلا) كما ترتكب الأخطاء النحوية عن طريق الخطأ و هذا لا يعني أننا لا نستطيع فهم النص . في المقابل في جهاز الحاسوب , أي خطأ لغوي يحدث دائما تحطم , بالإضافة إلى عرض رسالة خطأ , خلال الأسابيع الأولى من مسيرتك للتعلم سوف تجد بالتأكيد العديد من الأخطاء و سوف تبقى وقت طويلا لتصحيحها . و لكن المحترفين سيكونون في وقت أقل من ذلك بكثير . ضع في إعتبارك أن الكلمات و الرموز ليس لها معنى في حد ذاتها بل هي مجرد تسلسل من الرموز التي يجري تحويلها تلقائيا إلى أرقام ثنائية . لذلك يجب أن نكون حذرين للغاية للحفاظ على النحو الصارم في بناء الجملة اللغوية . أخيرا , تذكر أن كل التفاصيل مهم (على سبيل المثال : الحرف الكبير و الحرف الصغير)و و علامات الترقيم و يمكن لأي خطأ (مهما كان صغيرا مثل نسيان فاصلة) يمكن أن يغير الكثير من معنى الجملة و يسبب لك مشاكل كثيرة . فمن حسن حظك أن تتعلم لأول مرة مع البايثون التفسيرية .لأن البحث عن أخطاء سيكون أمرا سهلا للغاية . و لكن مع اللغات التي تستعمل مترجما مثل سي بلس بلس يجب عليك إعادة ترجمة البرنامج بأكملة بعد كل تغيير مهما كان صغيرا .

دلالات الأخطاء

أما النوع الثاني من الأخطاء هو الخطأ المنطقي أو الخطأ الدلالي . إذا كان هناك هذا النوع من الأخطاء في أحد البرامج فيتم تشغيل هذا البرنامج بدون أن تحصل على أية رسالة خطأ و لكن النتيجة هي غير متوقعة , فلقد كنت تقصد شيئا أخر . في الواقع , بقوم البرنامج بالضبط ما طلب منه لكن المشكلة أنه لا يطابق ما تريد أن تفعله أنت . تسلسل التعليمات في البرنامج لا يلبي الهدف . المنطق أو الدلالة خاطئة . إن البحث عن الأخطاء المنطقية مهمة شاقة جدا . من شأنها أن تثبت

قدرتك على إزالة أي شكل من " التفكير السحري " في حججك . و سوف تحتاج إلى الصبر و يجب عليك كتابة سطر وراء سطر في المفسر لتعرف أين الخطأ المنطقي .

أخطاء وقت التشغيل

النوع الثالث من الأخطاء هو خطأ وقت التشغيل و الذي يظهر فقط عندما يكون البرنامج قيد التشغيل , و لكن يجب أن تنشأ ظروف خاصة ليظهر هذا الخطأ (على سبيل المثال , عندما يحاول البرنامج قراءة ملف لم يعد موجودا)و تسمى هذه الأخطاء بالإستثناءات لأنها تشير إلى شيئ إستثائي (ولكنه وقع) و سوف تواجه هذه الأخطاء عند جدولة المشاريع الكبيرة و سوف نتعلم في وقت لاحق من الدورة كيفية التعامل مع هذه الأخطاء .

البحث عن الأخطاء و التجارب

من أهم المهارات للحصول على تدريب خاص بك هو وضع البرنامج في حالة ("دابغينغ") (تصحيح) . هذا النشاط قد يجننك أحيانا لكن إنها دائما غنية بالمعلومات , حيث ستتعلم الكثير من الأشياء و الأفكار . و هذا العمل هو مماثل في كثير من النواحي لتحقيق الشرطة . حيث يمكنك تفحص مجموعة من الحقائق و يجب عليك أن تضع فرضيات عمليات التفسيرية و إعادة بناء الأحداث التي أدت منطقيا إلى حدوث هذه النتائج التي نراها . و يرتبط هذا النشاط أيضا بالعمل التجريبي في العلوم . حيث يمكنك الحصول أولا على ما هو الخطأ , و يمكنك تغيير البرنامجك و المحاولة مرة أخرى و جعله فرضية التي تتيح لك التنبؤ بما هي التغييرات التي سوف تغيرها . إذا كان التنبؤ صحيحا فستتقدم خطوة إلى الأمام نحو البرنامج الذي يعمل بدون مشاكل و إذا كان التنبؤ يثبت الخطأ يجب عليك إجراء فرضية جديدة . كما قال شرلوك هولمز : " عندما تقضي على المستحيل , ما يتبقى , حتى لو كان هذا غير محتمل يحب أن يكون الحقيقة " (أ . كونان دويل , البرج رابع) بالنسبة لبعض الناس , " البرمجة" و "التصحيح " يبدو لهم نفس الشئ , و هذا معناه أن نشاط البرمجة هو في الواقع تعديل و تصحيح البرنامج نفسه بإستمرار, حتى يعمل كما تريده . و الفكرة هي بناء برنامج يبدأ دائما مع الخطوط العريضة التي هي بالفعل شيئا ما (و التي يتم تصحيحها بالفعل), و التي يتم إضافة طبقة طبقة من التغييرات الصغيرة, و تعديل الأخطاء تدريجيا و ذلك يتم في كل مراحل من عملية البرنامج الذي يعمل. على سبيل المثال أنتم تعلمون أن لينكس

هو نظام تشغيل (يعني برنامج كبير) و هو يحتوي على الألاف من الأسطر من التعليمات البرمجية , و لقد بدأ لينوس تورفالدس ببرنامج صغير بسيط لإختبار ملامح من إنتل 80386 . ووفقا للاري غرينفيلد " دليل مستخدم لينكس " إصدار بيتا : "و كان من بين المشاريع الأولى للينوس هو برنامج لتحويل سلسلة AAA إلى BBB و هذا ما أصبح في نهاية المطاف نظام تشغيل لينكس ." ولا تعتقد أننا نريد أن ندفعك لتعلم البرمجة عن طريق التجربة و الخطأ . من فكرة غامضة عندما تبدأ مشروع برمجي من أهمية معينة, يجب أن تقوم بعمل مواصفات بأدق تفصيل ممكن, و التي على أساس متين يبنى للتطبيق المقصود . توجد أساليب مختلفة للتحليل . لكن هذه الدراسة هي خارج هذه الملاحظات , و مع ذلك سنقدم في وقت لاحق (أنظر الفصل 15) بعض الأفكار الأساسية .

2

الخطوات الأولى

البرمجة هي فن قيادة الحاسوب ليفعل ما تريده بالضبط، و بايثون هي واحدة من اللغات القادرة على فهمك من أجل الحصول على أوامرك، سنحاول أن نبدأ فوراً بأوامر بسيطة جدا ألا هي الأرقام؛ لأن الأرقام هي المفضلة لدي، و سنقدم أول "التعليمات"، و نحدد طريقة تعريف بعض مفردات الحاسوب الأساسية، التي سنعمل عليها .

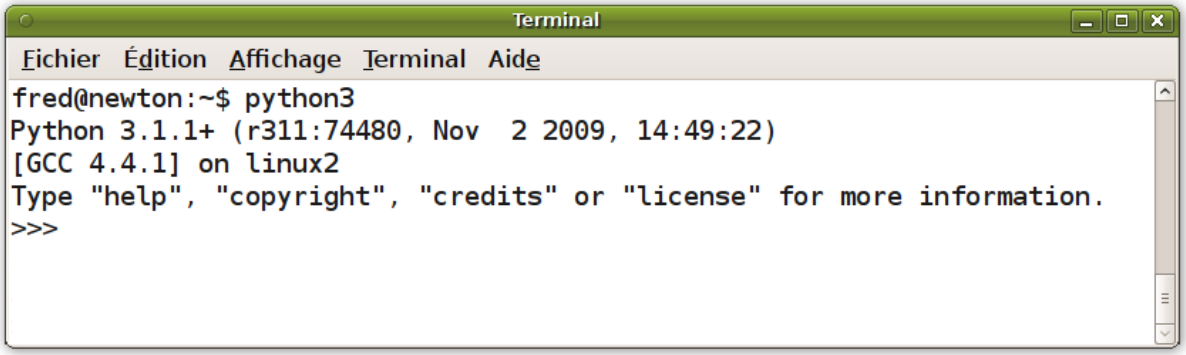
و كما شرحنا في المقدمة (انظر إصدارات اللغة، الصفحة الثانية عشرة) لقد اخترنا استخدام الإصدار الجديد من بايثون، ألا وهو الثالث، و الذي أُدخلت عليه بعض التغييرات النحوية خلافاً للإصدارات السابقة . وسأخبركم -متى كان ممكناً- بالاختلافات بين إصدارات بايثون حتى تتمكنوا من تحليل أو استخدام برامج قديمة مكتوبة ببايثون 1 أو 2.

الحساب مع بايثون

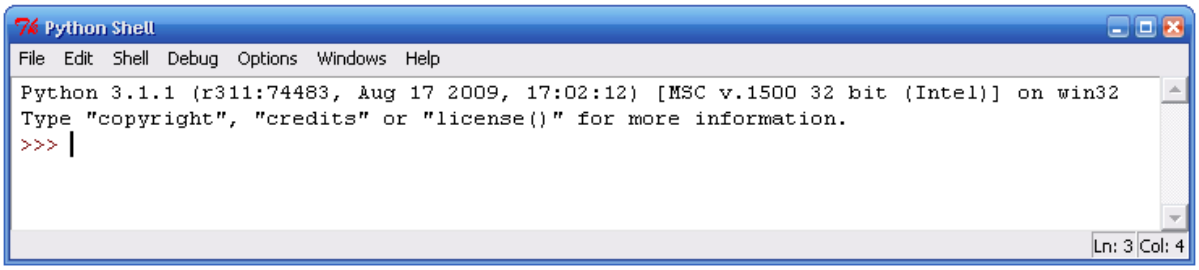
تتميز بايثون بأنها تُستخدم بطرق عدة، و سنستخدم بايثون أولا في وضع تبادلي و هذا يعني التحدث مع بايثون مباشرة عبر لوحة المفاتيح . و هذا سيجعلنا نكتشف سريعا مميزات لغة بايثون . ثم سنتعلم كيفية إنشاء برامجنا الأولى النصية (سكريبتات) و حفظها على القرص.

يمكن تشغيل المترجم من الطرفية مباشرة (في لينكس "shell" أو في نافذة "dos" في نظام التشغيل ويندوز) نحتاج إلى كتابة الأمر python3 فقط (نفترض أن البرنامج كان مثبتاً تثبيتاً صحيحاً و إصدار بايثون هو 3) أو نكتب python فقط (إذا كان إصدار بايثون المثبت على جهازك أقدم من الإصدار 3)

إذا كنت تستخدم واجهة رسومية مثل ويندوز، جنوم أو كدي أو WindowMaker فربما تفضل العمل في ("نافذة الطرفية") أو في محطة عمل مثل IDLE على سبيل المثال, هذه صورة لنافذة الطرفية لواجهة جنوم في نظام تشغيل لينكس (توزيعة أبونتو)[3] :



في IDLE في ويندوز، سيكون مكان عملك كهذا :



[3] في نظام تشغيل ويندوز, في الغالب يجب عليك الإختيار بين بيئة IDLE التي تم تطويرها من قبل غويدو فان روسيم, و التي نفضلنا نحن, و بين PythonWin, واجهة تطوير تم طويرها من قبل مارك هامونك . و توجد أيضا بيئات أخرى أكثر تطورا, مثل Boa Constructor على سبيل المثال (التي تعمل بشكل مماثل لدلفي), لكننا نعتقد أنه غير مناسب جدا للمبتدئين . لمزيد من المعلومات يرجى زيارة موقع البايثون .
في نظام تشغيل لينكس, نحن نفضل شخصيا العمل في نافذة الطرفية البسيطة لتشغيل مفسر البايثون أو تشغيل السكريبتات, و إستخدام محرر نص عاد مثل Gedit أو Kate أو واحد أكثر تخصصا مثل Geany .

الإشارة "أكبر من" المكررة ثلاثا هي إشارة سريعة أو موجِّه فوري يخبرك أن بايثون مستعد لتلقي الأوامر .

على سبيل المثال, تستطيع استعمال المفسر على أنه آلة حاسبة بسيطة لسطح المكتب . أرجو منك أن تختبر الأوامر التالية بنفسك (تعود على استعمال كراس التمارين لكتابة النتائج التي تظهر على الشاشة):

```
>>> 5+3

>>> 2 - 9        #المسافات إختيارية

>>> 7 + 3 * 4    #التسلسل الهرمي للعمليات الحسابية

>>> (7+3)*4

>>> 20 / 3       #انتباه: هذا يعمل بشكل مختلف في بايثون 2

>>> 20 // 3
```

كما ترون، فإن العوامل الحسابية للجمع و الطرح و الضرب و القسمة هي على التوالي + و - و * و \ .وما بين القوسين نتيجته متوقعة:

> في البايثون 3, عامل القسمة / يقوم بقسمة حقيقية (عدد حقيقي) . و إذا أردت أن تحصل على قسمة عدد صحيح, يجب عليك إستخدام المعامل // . و يرجى ملاحظة أن هذه التغييرات أدخلت على تكوين جملة البايثون 3, مقارنة مع الإصدارات السابقة . فإذا كنت تستخدم إحدى هذه الإصدارات, يرجى ملاحظة أن المعامل / يقوم بقسمة عدد صحيح بشكل إفتراضي, إذا قمت بتمرير برامترات أعداد صحيحة و قسمة عدد حقيقي, إذا أدخلت (على الأقل) عدد حقيقي واحد . لحسن الحظ تم التخلي عن السلوك القديم من البايثون, لأنها كانت تؤدي إلى خلل في بعض الأحيان من الصعب إكتشافه .

```
>>> 20.5 / 3

>>> 8,7 / 5        #خطأ!
```

يرجى ملاحظة أن هنالك قاعدة تطبَّق في جميع لغات البرمجة هي الاتفاقيات الرياضيات السارية في البلدان الناطقة باللغة الإنجليزية و من بينها الفاصلة العشرية التي هي دائما نقطة، وليست فاصلة كما عندنا. و سوف نلاحظ أيضا أن في عالم الحاسوب تتم غالبية الأعداد الحقيقية كأرقام "النقطة العائمة".

البيانات والمتغيرات

سيكون لدينا الفرصة لتعلم تفاصيل أكثر لأنواع مختلفة من البيانات الرقمية. و لكن قبل ذلك سنتناول الآن مفهوم أكثر أهمية. العمل الرئيسي الذي يقوم به برنامج الحاسوب هو معالجة البيانات. و يمكن لهذه البيانات أن تكون متنوعة جدًا (في الواقع, كلها رقمية[4]), لكن في ذاكرة الحاسوب تحول دائما في النهاية إلى تسلسل محدد من الأرقام الثنائية . للوصول إلى البيانات يقوم برنامج الحاسوب (بغض النظر عن اللغة المستخدمة في كتابته) باستخدام أعداد كبيرة من المتغيرات مختلفة الأنواع . المتغير يظهر في لغة البرمجة كأي اسم متغير آخر تقريبا (أنظر أدناه) و لكن عند الحاسوب يكون مرجعا يشير إلى عنوان ذاكرة، و هذا معناه موقع محدد في ذاكرة الوصول العشوائي (ram). في هذا الموقع يتم تخزين قيمة محددة (و هي البيانات) في سلسلة من الأرقام الثنائية، و لكن ليس بالضرورة رقم واحد في نظر لغة البرمجة المستخدمة، و يمكن أن يتم هذا عن طريق أي "كائن" و يتم وضعه في ذاكرة الحاسوب. على سبيل المثال : عدد صحيح، عدد حقيقي، سلسلة نصية، ناقل، سلسلة مطبعية، جدول أو وظيفة ... إلخ، سوف نشرح في الصفحات التالية التمييز بين كل هذه المحتويات الممكنة و لغة البرمجة المستخدمة لأنواع مختلفة من المتغيرات (عدد صحيح, عدد حقيقي، سلسلة نصية, قائمة ... إلخ).

4ماذا يمكن فحصه بالضبط ؟ هذا السؤال مهم جدا, و يجب عليك البحث في دراستك في العلوم العامة .

أسماء المتغيرات و الكلمات المحجوزة

أسماء المتغيرات هي الأسماء التي تستطيع تسميتها بحرية (تقريبا)، حاول اختيارهم بشكل جيد ويفضل أن يكون قصيرا جدا و واضحا بقدر الإمكان؛ و ذلك للتعبير بوضوح عما يفترض أن يحتوي المتغير، على سبيل المثال، أسماء المتغيرات مثل الارتفاع أو الارتفاع البديل أكثر ملائمة للتعبير من ارتفاع x.

المبرمج الجيد هو الذي يجعل أسطر برنامجه سهلة القراءة.

في بايثون يجب أن تتبع أسماء المتغيرات بعض القواعد البسيطة :

- اسم المتغير يتكون من سلسلة من الحروف (A إلى Z الكبيرة) و (a إلى z الصغيرة) و الأرقام (0 إلى 9) و يجب أن يبدأ دائما بحرف.
- لا يسمح إلا بالأحرف العادية و يمنع استعمال المساحات و الرموز (أحرف خاصة مثل!@#$ %^&* و ما إلى ذلك) باستثناء الرمز _ (خط تحت السطر).
- من المهم التمييز بين الأحرف الكبيرة و الصغيرة.

  انتبه: Joseph, joseph, JOSEPH متغيرات مختلفة فكن حذرا .

تعود على كتابة معظم الأسماء بأحرف صغيرة . هذه مجرد اتفاقية و لكنها باحترام واسع . إستخدم الأحرف الكبيرة ضمن نفس الاسم، ذلك لزيادة سهولة القراءة، كما هو الحال في TableofContents.

بالإضافة إلى تلك القواعد، يجب أن نضيف أيضا أنه لا يمكننا استخدام أسماء متغيرات محجوزة (33 كلمة) أدناه؛ يتم استخدامها من قبل اللغة نفسها:

and	as	assert	break	class	continue	def
del	elif	else	except	False	finally	for
from	global	if	import	in	is	lambda
None	nonlocal	not	or	pass	raise	return
True	try	while	with	yield		

تعيين قيمة متغير

لقد أصبحنا نعرف الآن كيف نختار اسم المتغير بحكمة، الآن سوف نشرح كيفية تعيين قيمة للمتغير و المصطلح "تعيين قيمة" هي دلالة على عملية إقامة صلة بين اسم المتغير و قيمته (محتوياته). في بايثون، كما هو الحال في العديد من اللغات الأخرى، تمثل هذه العملية بواسطة علامة المساواة (=) :

```
>>> n = 7                    # اعطاء القيمة 7 للمتغير n
>>> msg = "Quoi de neuf ?"   # تعيين قيمة (Quoi de neuf ?) للمتغير msg
>>> pi = 3.14159             # اعطاء قيمة للمتغير pi
```

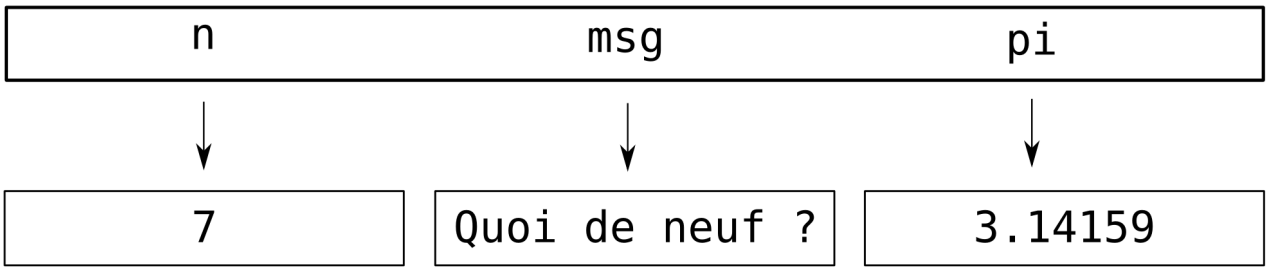
و توضح الأمثلة أعلاه تعيين البيانات في بايثون، بعد أن تم تنفيذها في ذاكرة الحاسوب، في أماكن مختلفة :

- أسماء ثلاثة متغيرات : **n** و **msg** و **pi**
- تسلسل البايتات الثلاثة ، والتي يتم ترميز **7** بعدد صحيح و جملة " **Quoi de neuf ?** " بسلسلة و العدد **3.14159** بعدد صحيح.

البيانات الثلاثة المذكورة أعلاه للتعيين، كان لكل منها أثر في تنفيذ عدة عمليات في ذاكرة الحاسوب :

- إنشاء و تخزين **اسم متغير**.
- تعيين **نوع** محدد جيد (سيتم توضيحه في الصفحة التالية).
- إنشاء و تخزين **قيمة** معينة.
- **ارتباط** (عن طريق مؤشرات داخلية) بين اسم المتغير و موقع الذاكرة من القيمة المطابقة.

أفضل طريقة للشرح هي تمثيل كل هذا برسم تخطيطي:



أسماء المتغيرات الثلاثة مخزن و لديها مراجع في منطقة معينة من الذاكرة تسمى المساحة الأسماء، في حين تقع القيم المناظرة في أماكن أخرى، و أحيانا بعيدة جدا عن بعضها البعض، و سوف نوضح هذا المفهوم أكثر في الصفحات القادمة .

عرض قيمة متغير

بعد التمارين المذكورة أعلاه، سيكون لدينا ثلاثة متغيرات **n** و **msg** و **pi**، لعرض قيمة متغير على الشاشة، هنالك طريقتان، الأولى هي كتابة اسم المتغير ثم النقر على مفتاح الإدخال "Enter"، فسيستجيب بايثون فيعرض القيمة :

```
>>> n
7
>>> msg
'Quoi de neuf ?'
>>> pi
3.14159
```

هذه ميزة ثانية للمفسر، الذي يهدف إلى جعل الحياة أسهل عند القيام بتمارين بسيطة في سطر الأوامر، في داخل البرنامج، سوف تستخدم دائما **print()**:

```
>>> print(msg)
Quoi de neuf ?
>>> print(n)
7
```

لاحظ الفرق الدقيق بين طرق العرض التي تم الحصول عليها مع كل طريقة. وظيفة **print()** لا تظهر بدقة قيمة المتغير كما أنه تم ترميزها، في حين أن الطريقة الأخرى (و هي فقط إدخال اسم المتغير) يعرض مثلا علامات الاقتباس لكي يذكركم أنكم تتعاملون مع متغير مثل سلسلة (سنتعرف إلى هذا في وقت أخر).

في الإصدارات السابقة للبايثون, دور دالة الطباعة ***print()*** هو التعليمة ***print*** الخاصة (مما يجعلها من الكلمات المحجوزة), و هذه التعليمة تستخدم بدون أقواس . في التمارين السابقة, يجب عليك إذا كتابة ***print n*** أو ***print msg*** . و إذا أردت لاحقا إستخدام البايثون 3 في برامج مكتوبة بإحدى نسخ البايثون القديمة, يجب عليك إضافة الأقواس بعد كل تعليمة ***print*** لتحويلها إلى دالة (يمكن للأدوات المساعدة القيام بذلك تلقائيا) .

في هذه الإصدارات القديمة, يتم معالجة السلاسل النصية بشكل مختلف (سوف نتحدث عن هذا بالتفصيل لاحقا) . و إعتمادا على تكوين جهاز حاسوبك, سوف تواجه بعض الرموز الغريبة عندما تتعامل مع سلاسل نصية تحتوي على أحرف معلمة, مثل :

```
>>> msg = "Mon prénom est Chimène"
>>> msg
'Mon pr|xe9nom est Chim|xe8ne'
```

هذا الأشياء الغريبة تنتمي إلى الماضي, و لكن سوف نرى لاحقا كيف أن المبرمج الجدير بهذا الإسم يجب أن يعرف كيف يتم ترميز المحارف التي تواجهه في مصادر مختلفة من البيانات, لأن تحديد هذه الترميزات تغيرت على مر السنين, و يجب أن نعرف التقنيات لتحويلها .

كتابة المتغيرات

في بايثون ليس من الضروري كتابة أسطر معينة من التعليمات البرمجية لتعريف نوع المتغير قبل استخدامها. لأنه ببساطة عند تعيين قيمة إلى اسم المتغير يقوم بايثون بوضع نوع المتغير تلقائيا مع النوع الذي يطابق القيمة المقدمة.

في التمرين السابق، على سبيل المثال، يتم إنشاء أنواع المتغيرات تلقائيا مثلا (**n** : صحيح, **msg** : سلسلة, **pi** : عدد حقيقي).

هذه هي ميزة مثيرة للاهتمام في بايثون، التي تتبع لعائلة معينة من اللغات حيث يوجد من نفس العائلة على سبيل المثال، ليسب و سشيم ...إلخ، طريقة كتابة المتغيرات في بايثون حيوية على عكس الكتابة الثابتة التي هي على قاعدة ثابتة، على سبيل المثال في لغة سي أو جافا الذي يجب الإعلان أول مرة عن اسم و نوع المتغير، عندها فقط يمكنك تعيين المحتوى، و الذي يجب بالطبع أن يكون متوافق مع النوع المعلن.

الكتابة الثابتة هي الأفضل في حالة اللغات المترجمة؛ لأنه يحسن عملية الجمع(التي نتيجتها برنامج بالبيناري).

الطباعة الديناميكية أكثر سهولة لكتابة بنيات المنطقية لمستوى عال (الإنعكاسية, متابروغراميك) و لا سِيَّما في سياق البرمجة الكائنية (تعدد الأشكال) كما يسهل استخدام هياكل البيانات الغنية مثل القوائم و القواميس.

تعدد المهام

في بايثون يمكنك تعيين قيمة لمتغيرات عدة في نفس الوقت، على سبيل المثال:

```
>>> x = y = 7
>>> x
7
>>> y
7
```

يمكنك أيضا إعطاء قيمة لكل متغير من المتغيرات في آن واحد معا:

```
>>> a, b = 4, 8.33
>>> a
4
>>> b
8.33
```

في هذا المثال، تم تعيين قيم مختلفة لكل من **a** و **b** -في وقت واحد- هي **4** و **8.33**

> الفرنكوفونيين (و العرب أيضا) لديهم عادة بإستخدام الفاصلة كفاصل عشري, و أما لغات البرمجة فتستخدم دائما الإتفاق الحالي بين بلدان الناطقة باللغة الإنكليزية, و هذا معناه تستخدم النفطة العشرية . و الفاصلة, التي نستخدمها نحن, تستخدم لفصل مختلف العناصر (برامترات, إلخ) و كما رأينا في مثالنا, لقيم المتغيرات التي قمنا بتعيينها .

تمارين

2.1 قم بوصف بوضوح ماذا يحدث عند كتابة كل واحدة من التعليمات الثلاثة للمثال في الأسفل :

```
>>> largeur = 20
>>> hauteur = 5 * 9.3
>>> largeur * hauteur
930
```

2.2 قم بتعيين 3,5,7 إلى المتغيرات a,b,c . ثم قم بتنفيذ a-b//c و فسر النتيجة المتحصل عليها .

المعاملات والتعابير

قم بالتلاعب بقيم المتغيرات مع المعاملات لتشكيل التعبيرات . على سبيل المثال :

```
a, b = 7.3, 12
y = 3*a + b/5
```

في هذا المثال, بدأنا بتعيين للمتغيرين **a** و **b** القيم **7,3** و **12** .

كما شرحنا سابقا, يقوم البايثون بتعيين تلقائيا نوع "حقيقي" للمتغير **a**, و النوع "صحيح" للمتغير **b** .

السطر الثاني من مثالنا, يقوم بتعيين للمتغير الجديد **y** نتيجة التعبير الذي يحتوي على المعاملات * و + و / مع المعاملين **a**,**b** **3**,**5** . و المعاملات هي رموز خاصة تستخدم لتمثيل العمليات الحسابية البسيطة مثل الجمع و الطرح و المعاملين هي قيم تجمع بمساعدة المعاملات .

يقوم البايثون بتقييم كل تعبير يقدم إليه, و لو كان معقد, و نتيجة هذا التقييم هو دائما قيمة , لهذه القيمة يتم تلقائيا تعيين النوعها, و هي تعتمد على ما يوجد في التعبير . في المثال أعلاه, سيكون **y** نوع حقيقي, لأن التعبير الذي تم تقييمه يحتوي على الأقل عدد حقيقي .

معاملات البايثون ليست فقط عوامل الأربعة الرياضية الأساسية . و لقد رأينا بالفعل وجود عامل القسمة الصحيح **//** . و يوجد أيضا المعامل ****** للأسس (القوة), و يوجد العديد من المعاملات المنطقية, مثل معاملات السلاسل النصية, و معاملات إختبار الهوية أو الإنتماء و إلخ . سوف نتحدث على كل هذا في الصفحات القادمة .

و بالمناسبة يوجد معامل مودولو(modulo), الممثل بالرمز **%** . هذا المعامل يقوم بتوفير ما تبقى من عملية القسمة لعدد صحيح . حاول على سبيل المثال :

```
>>> 10 % 3          #! لاحظ ما يحدث
>>> 10 % 5
```

هذا المعامل سيكون مفيدا في وقت لاحق, و خاصة لإختبار ما إذا كان العدد **a** يقبل القسمة على **b** . و يكفي للتحقق كتابة **a % b** ليعطي نتيجة صفر إذا كان يقبل القسمة .

تمارين

2.3 أختبر أسطر التعليمات، وصِف ما يحدث :

```
>>> r, pi = 12, 3.14159
>>> s = pi * r**2
>>> print(s)
>>> print(type(r), type(pi), type(s))
```

في رأيك، ما فائدة الدالة **type()** ؟

(ملاحظة: سيتم وصف المهام بالتفصيل في الفصلين 6 و 7.)

ترتيب المعاملات

عندما يكون هنالك أكثر من معامل في التعبير، فيجب علينا أن نعرف ترتيب العمليات التي تعتمد على القواعد الأولية، في بايثون، القواعد الأولية هي نفس قواعد الرياضيات التي كنت تدرسها، و يمكنك حفظها بسهولة باستخدام خدعة PEMDAS إختصارا لـ :

- P ما بين قوسين، فَلِما بين القوسين أولوية قصوى، حيث تستطيع أن ترتب التعبير بالترتيب الذي تريده.
  هكذا **2*(1-3) = 4** ،و **(1+1)**(2-5) = 8**.
- E للأسس(القوة)، الأولوية الثانية للأسس قبل غيرها من العمليات
  هكذا **2**1+1 = 3** (ليس 4)، و **3*1**10 = 3** (ليس 59049 !).
- M و D للضرب والقسمة، التي لهما نفس الأولوية. يتم تقيمهما قبل الجمع A والطرح B اللذان يجران آخر الأمر.
  هكذا **2*3-1 = 5** (بدلا من 4)، و **1-2/3 = -0.3333**... (بدلا من 1.0).
- إذا كان هناك أكثر من عاملين لهما نفس الأولوية، يتم التقييم من اليسار إلى اليمين.
  وبالتالي في التعبير **59*100//60** يجب إجراء الضرب أولا، ثم ينفذ الجهاز 60//5900 الذي يعطي 98، إذا أجريت القسمة أولا ستكون النتيجة 59 ().
- (تذكر هنا أن المعامل **//** يقوم بعملية قسمة عدد صحيح, و قم بالتحقق من خلال **59*(100//60)**) .

البنية

حتى الأن إطلعنا على مختلف عناصر لغة البرمجة وهُم : المتغيرات، التعبيرات، التعليمات و لكن دون معالجة، كيف يمكننا أن نوحدها بعضها مع بعض؟

نأتي إلى ذكر واحدة من نقاط القوة الكبيرة لِلُغات البرمجة رفيعة المستوى هي أنها تتيح بناء مجموعة تعليمات عن طرق تجميع أجزاء مختلفة، على سبيل المثال، إذا كنت تعرف كيفية إضافة رقمين و كيفية عرض قيمة، يمكنك الجمع بين هتين التعليمتين :

```
>>> print(17 + 3)
>>> 20
```

هذا قليل من كثير، ستتوضح هذه الميزة عندما تقوم بخوارزميات معقدة بوضوح واختصار، على سبيل المثال :

```
>>> h, m, s = 15, 27, 34
>>> print("عدد الثواني المنقضية منذ منتصف الليل = ", h*3600 + m*60 + s)
```

تنبيه : هناك حد لما يمكن جمعه :

في العبارة، يجب عليك أن تضع إسم المتغير على الجانب الأيسر من علامة المساواة و ليس التعبير و ذلك لأنها لا تمتلك نفس المعنى كما في الرياضيات : كما لاحظنا في وقت سابق و هذه هي مهمة هذا الرمز (وضع بعض المحتويات إلى متغير) و هو ليس رمزا للمساواة . و سوف نتحدث على رمز المساواة (على سبيل المثال في الجمل الشرطية) فيما بعد.

على سبيل المثال ، العبارة **m + 1 = b** خاطئة.

من السلبيات، كتابة **a = a + 1** ، لأنه أمر غير مقبول في الرياضيات، في حين أن هذا النوع من الكتابة شائع جدا في البرمجة. معنى **a = a + 1** هو زيادة قيمة المتغير بقيمة 1.

سوف نتحدث حول هذا الموضوع قريبا. نحن نحتاج أولا إلى فهم مفهوم أكثر أهمية.

3

التحكم في تلقيم التنفيذ

في الفصل الأول، عرفنا أن النشاط الأساسي للمبرمج هو إيجاد حلول للمشاكل، و من أجل حل مشكلة في الحاسوب، يجب دائما القيام بسلسلة من الإجراءات بترتيب معين، و يطلق على هذه الإجراءات و الترتيب الذي ينبغي القيام به بالخوارزمية .

في بايثون يسمى هذا البرنامج بتلقيم التنفيذ، و تسمى الهياكل التي تعدلها بتعليمات التحكم في التلقيم ببنية التحكم.

وهي مجموعات من التعليمات التي تحدد ترتيب الإجراءات التي يتم تنفيذها. في البرمجة الحديثة، توجد ثلاثة أنواع فقط: السلسلة والشروط التي سنناقشها في هذا الفصل والتكرار الذي سنناقشه في الفصل القادم .

تسلسل التعليمات

الذي لم نذكره هو أنه يتم تنفيذ تعليمات البرنامج الواحدة تلو الأخرى بالترتيب كما هي مكتوبة في السكربت.

قد يبدو لك هذا تافها للوهلة الأولى، و مع ذلك التجربة تدل على أن عددا كبيرا من الأخطاء الدلالية في برامج الحاسوب هي نتيجة لتعليمات سيئة (خاطئة). كلما تقدمت أكثر في فن البرمجة، سوف تدرك أكثر حول الحذر بشأن المكان الذي يجب وضع التعليمات الخاصة بك واحدة تلو الأخرى. على سبيل المثال، في تسلسل التعليمات التالية :

```
>>> a, b = 3, 7
>>> a = b
>>> b = a
>>> print(a, b)
```

سوف تحصل على نتيجة عكسية إذا كنت في بدلت بين السطر الثاني و الثالث .

بايثون يدير التعليمات في الوضع الطبيعي من البداية إلى النهاية، إلا إن واجه جملا شرطية مثل if كما هو واضح أدناه (سوف نتعرف على غيرها و لاسيما الحلقات التكرارية)، و هذه التعليمات تسمح للبرنامج بمتابعة المسارات المختلفة تبعا للظروف.

تحديد أو تنفيذ شرط

إذا كنا نريد كتابة تطبيقات مفيدة، فنحن بحاجة إلى توجيه تقنيات تشغيل البرنامج في إتجاهات مختلفة، تبعا للظروف التي تواجه البرنامج، وللقيام بذلك نحن بحاجة إلى تعليمات لإختبار شرط و تعديل سلوك البرنامج وفقا لذلك.

أبسط هذه الجمل الشرطية هي التعليمة if، و لإختبار عملها يجب عليك إدخال هذان السطران إلى محرر بايثون :

```
>>> a = 150
>>> if (a > 100):
...
```

الأمر الأول هو لتعيين قيمة 150 إلى المتغير a. حتى الأن لا شيء جديد.

و عند الإنتهاء من إدخال السطر الثاني، ستجد أن بايثون سيتفاعل بطريقة جديدة، في الواقع إذا لم تنسى الرمز ":" في نهاية السطر، ستجد أنه قد تم إستبدال الموجه الرئيسي (<<<) إلى موجه ثانوي يتكون من ثلاثة نقاط.

إذا كان لديك محرر لاتلقائي يجب عليك الآن إدخال تبويت (أو أدخل أربعة مسافات) قبل الدخول إلى السطر التالي، بحيث يبدء بذلك، و ينبغي أن تظهر الشاشة على النحو التالي :

```
>>> a = 150
>>> if (a > 100):
...    print("a dépasse la centaine ")
...
```

أنقر مرة أخرى زر الإدخال (Enter) سترى أن البرنامج سيعمل و ستحصل على :

```
a dépasse la centaine
```

كرر نفس العملية لكن مع a = 20 في السطر الأول: في هذه المرة، بايثون لم يظهر شيء.

التعبير الذي وضعته بين قوسين نسميه شرط، يُستخدَم if لإختبار صحة هذا الشرط، فإذا كان الشرط صحيحا فسوف يتم تنفيذ ما بعد الرمز ":" و إذا كان الشرط غير صحيح، لا يحدث شيء، لاحظ أن الأقواس المستخدمة هنا مع عبارة if إختيارية، لقد إستخدمناها لتحسين إمكانية القراءة، في اللغات الأخرى، قد تصبح إلزامية

أكرر مرة أخرى, بعد إضافة أول السطرين كما هو مبين في الأسفل . تأكد من أن السطر الرابع بدأ على اليسار (بدون مسافة البادئة), و لكن مرة أخرى نضيف بادئة جديدة في السطر الخامس (و يفضل نفس مسافة بادئة السطر الثالث) :

```
>>> a = 20
>>> if (a > 100):
...    print("a dépasse la centaine ")
... else:
...    print("a ne dépasse pas cent ")
...
```

أنقر مرة أخرى زر الإدخال (Enter) سيعمل البرنامج وسيعرض :

```
a ne dépasse pas cent
```

كما فهمت أنت, العبارة **else** ("إذا") تسمح للمبرمج بتعيين تنفيذ بديل في برنامج من إحتمالين . و يمكننا فعل أفضل من هذا عن طريق إستحدام العبارة **elif** (دمج else if) :

```
>>> a = 0
>>> if a > 0 :
...    print("a est positif ")
... elif a < 0 :
...    print("a est négatif ")
... else:
...    print("a est nul ")
...
```

مقارنة المعاملات

التعليمة if للتحقق من الشروط، قد تحتوي على عوامل المقارنة التالية :

```
x == y     #x يساوي y
x != y     #x لا يساوي y
x > y      #x أكبر من y
x < y      #x أصغر من y
x >= y     #x أكبر من أو يساوي y
x <= y     #x أصغر من أو يساوي y
```

مثال :

```
>>> a = 7
>>> if (a % 2 == 0):
...    print("a est pair")
...    print("parce que le reste de sa division par 2 est nul")
... else:
...    print("a est impair")
...
```

لاحظ أن عامل المساواة بين القيمتين يتكون من رمزي مساواة و ليس واحد. علامة مساواة واحدة هي عامل تعيين وليس عامل مقارنة. و سوف تجد نفس الرمز في لغات أخرى كجافا و سي بلس بلس.

بيانات المشغّل - عبارة الكتل

البناء الذي إستخدمته مع العبارة **if** هو مثالك الأول من مشغّل البيانات. قريبا سوف تجمع آخرون . في بايثون، البيانات المركبة لديهم نفس البنية: نقتطين عموديتان تليهما عبارة أو أكثر البادئة تحت خط العمود، على سبيل المثال :

```
Ligne d'en-tête:
  première instruction du bloc
  ... ...
  ... ...
  dernière instruction du bloc
```

إذا كان هنالك عبارات متعددة مسبوقة بِبادئة تحت الخط العمودي، يجب أن يكونوا على نفس المستوى (على سبيل المثال 4 مساحات فارغة)، بادئة هذه التعليمة هي ما نسميه الآن كتلة العبارات. كتلة العبارات هي سلسلة من التعليمات تُشكِّل وِحدة المَنطق، و التي لا يتم تنفيذها إلا في ظروف معينة يتم تحديدها في الخط العمودي. في المثال في الفقرة السابقة، سطريّ العبارة تحت السطر الذي يحتوي العبارة if هما كتلة المنطق نفسها فسيتم تنفيذ هذان السطرين (على حد سواء) إذا كان إختبار الشرط مع العبارة صحيح و هذا معناه باقي القسمة على 2 لا شيء.

التداخلات

من الممكن تداخل في كل المركبات عدة تصاريح أخرى من أجل تنفيذ المشغّل هياكل صنع القرار، على سبيل المثال :

```
if embranchement == "vertébrés":          # 1
  if classe == "mammifères":              # 2
    if ordre == "carnivores":             # 3
```

```
        if famille == "félins":            # 4
          print("c'est peut-être un chat")   # 5
    print("c'est en tous cas un mammifère")    # 6
  elif classe == "oiseaux":                # 7
    print("c'est peut-être un canari")       # 8
print("la classification des animaux est complexe") # 9
```

تحليل هذا المثال، هذا الجزء من البرنامج لا يطبع عبارة " c'est peut-être un chat " إلا إن كانت نتائج إختبار أول أربعة شروط صحيحة.

ليتم عرض جملة "c'est en tous cas un mammifère" يجب أن يتحققا شرطين. العبارة المطبوعة في هذه الجملة (السطر الرابع) هي في الواقع على مستوى المسافة البادئة لنفس التعليمات: **if ordre == "carnivores"** (السطر الثالث) وبالتالي هما -على حد سواء- جزء من الكتلة نفسها، و التي تُعرض إذا كانت نتائج إختبار الشروط في السطران الأول و الثاني صحيحة.

ليتم عرض عبارة "c'est peut-être un canari" يجب أن يكون المتغير **branch** يحتوي على " vertébrés" ومتغير **class** يحتوي على "oiseaux ".

أما بالنسبة للجملة في السطر التاسع، فيتم عرضها في جميع الحالات، لأنها جزء من كتلة نفس البيانات في السطر الأول .

بعض القواعد لبناء جملة في بايثون

كل ما سبق يقودنا إلى بعض قواعد بناء جملة :

تعريف حدود التعليمات والكتل بالتخطيط

في كثير من لغات البرمجة، يجب عليك إتمام كل سطر من التعليمات برمز خاص (غالبا ما يكون الفاصلة المنقوطة)، في بايثون، هذا الحرف في نهاية السطر يلعب هذا الدور، (سنرى لاحقا كيفية إستعمال هذه القاعدة لتوسيع مشغّل البيانات إلى أسطر متعددة) و أيضا إستكمال خط التعليمة مع التعليق. تعليق

بايثون يبدأ دائما مع الحرف الخاص **#** و يتم تجاهل كل ما يتم تضمينه و الإنتقال إلى السطر التالي من قِبَل المشغّل.

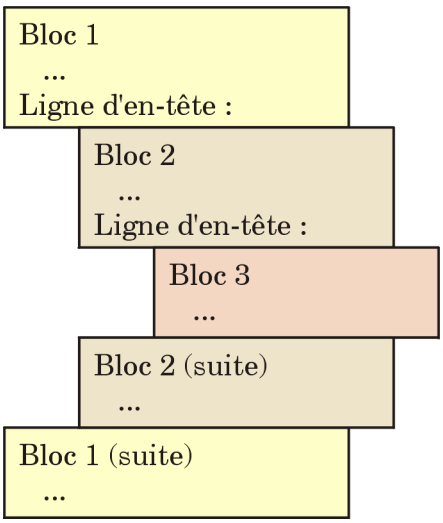
و في معظم لغات البرمجة الأخرى، يجب أن تكون كتلة البيانات مع رموز معينة (حتى في بعض الأحيان تعليمات البداية "**begin**" و النهاية "**end**") في سي بلس بلس و جافا على سبيل المثال، يجب وضع كتلة البيانات داخل أقواس، و هذا يسمح لكتابة كتل من التعليمات واحدة تلو الأخرى، من دون قلق أو فواصل أو سطر المسافة البادئة و لكن هذا يمكن أن يؤدي إلى ارتباك في كتابة البرامج، فنحن البشر من الصعب علينا قراءة هذه الفقرات. و لذلك ينصح جميع المبرمجين الذين يستخدمون هذه اللغات باستخدام فواصل الأسطر و المسافات البادئة لترسم كتلا بصرية.

مع بايثون، يجب إستخدام فواصل الأسطر و المسافات البادئة، و لكن في المقابل لا يوجد ما يدعوا للقلق من رموز كتلة التحديدات الأخرى. في النهاية، بايثون يدفعك لكتابة شفرة مصدرية قابلة للقراءة وأخذ العادات الجيدة التي كنت تحتفظ بها عند إستخدام لغات أخرى.

مشغّل البيانات: الرأس، النقطتان وكتلة بادئة للبيانات

سيكون لدينا فرصا عديدة لإستكشاف مفهوم "تعليمة الكتلة" و القيام بتمارين حول هذا الموضوع في الفصل التالي.

و يلخص الرسم البياني أدناه المبدأ. (fixme)



- ترتبط التعليمات دائما مع الخط العمودي الذي يحتوي على تعليمات محددة للغاية (if ,elif , else, while , def ...إلخ) التي تنتهي بنتقطتين .
- الكتل محددة من قبل مسافة البادئة: يجب أن تكون جميع الأسطر على طريقة (طول البادئة) (و هذا يعني التحول إلى نفس العدد من المسافات) يمكن استخدام أي عدد للمسافات ولكن معظم المبرمجين يستخدمون مضاعفات الرقم 4.
- لاحظ أنه يمكن للكود الكتابة بعيدا (كتلة 1) في حد ذاته يمكن إزالتها من الهامش الأيسر (تداخل في أي شيئ)

> **مهم**
>
> يمكنك أيضا وضع مسافة البادئة من خلال علامات التبويت, و لكن يجب أن تكون بعد ذلك حذرا جدا بعدم إستخدام المسافات في بعض الأحيان, و في بعض الأحيان الأخرى مسافات البادئةلتعريف أسطر نفس الكتلة . و حتى لو تبدو مطابقة على الشاشة, فالمسافات و علامات التبويت هي رموز ثنائية منفصلة : و لذلك ينظر البايثون إلى هذه الأسطر هي جزء من كتل مختلفة . و الذي قد يؤدي إلى أخطاء يصعب تصحيحها . و لذلك يفضل معظم المبرمجين إستخدام علامات التبويت . إذا كنت تستخدم محرر نصوص "ذكي", فمن الأفضل تفعيل خيار "إستبدال علامات التبويت بمسافات" .

المسافات و التعليقات عادة ما يتم تجاهلها

بصرف النظر عن تلك المستخدم لمسافة البادئة(في بداية السطر), المساحات الموضوعة داخل التعليمات و التعبيرات دائما ما يتم تجاهلها (ما لم تكن جزء من سلسلة نصية) . و هذا نفس الشيئ بالنسبة للتعليقات : فهي تبدأ برمز **#** و تمتد إلى نهاية السطر الحالي(يتم تجاهلها) .

4

تعليمات التكرار

واحدة من المهام التي تبذل فيه الألات قصار جهدها هو تكرار المهام المتماثلة من دون خطأ . هنلك العديد من الطرق لبرمجة هذه المهام المتكررة . سنبدأ مع واحدة التي تعتبر الأكثر أساسية و هي حلقة تكرار while .

إعادة التعيين

حتى الأن لم نخبرك بأنه يجوز إعادة تعيين قيمة جديدة لمتغير واحد , مرة واحدة أو عدة مرات على النحو المطلوب .

عندما نقوم بإعادة تعيين , نحن نقوم بإستبدال القيمة القديمة بقيمة جديدة لنفس المتغير .

```
>>> altitude = 320
>>> print(altitude)
320
>>> altitude = 375
>>> print(altitude)
375
```

هذا يجعلنا إلى لفت إنتباهكم مرة أخرى إلى أنه يجب علينا أن لا نخلط بين رمز التعيين في البايثون و رمز المساوات كما يفهم في الرياضيات , لأن هنالك العديد من الأشخاص يفسرون هذه العبارة altitude = 320 كأن هذا الرمز للمساوات , و هي ليست كذلك !

- أولا المساوات هي عملية تبادلية , في حين أن التعيين ليس كذلك فمثلا في الرياضيات كتابة a =7 أو 7 =a نفس الشيئ في حين أن في البرمجة (على سبيل المثال altitude = 374) سيكون غير منطقي .
- ثانيا , المساوات هي دائمة, في حين يمكن إستبدال قيمة المتغير في البايثون بإستخدام رمز التعيين كما رأينا للتو . في الرياضيات نؤكد أن هذه مساوات أ = ب في بداية الحجة, ثم نواصل لتكون مساوية ل ب في جميع التطورات القادمة .

في البرمجة , العبارة التعيين الأولى هي معادلة قيمتين , والعبارة الأخرى لإستبدال قيمة الأولى , على سبيل المثال :

```
>>> a = 5
>>> b = a      # لديهم نفس القيمة a و b
>>> b = 2      # أصبحوا مختلفان الأن a و b
```

تذكروا أن البايثون يسمح لك بتعيين عدة قيم متغيرات في نفس الوقت :

```
>>> a, b, c, d = 3, 4, 5, 7
```

هذه ميزة من ميزات البايثون الأكثر إثارة للإهتمام عند النظرة الأولى .

على سبيل المثال , لنفترض أننا صنعنا المتغيرين a و c و هي تحتوي على القيمتين 5 و 3 , نريد عكس القيمتين) كيف نفعل هذا ؟

تمرين

4.1 أكتب الكود اللازمة لتحقيق هذه النتيجة (فوق).

بعد القيام بالعملية المقترحة أعلاه , سوف تجد بالتأكيد وسيلة , و سيطلب منك المعلم على الأرجح التعليق في الصف . لأن هذه هي عملية مشتركة , إن لغات البرمجة غالبا ما تقدم إختصارات لأداء (تعليمات المتخصصة على سبيل المثال, مثل تعليمة المبادلة الأساسية) في البايثون, يمكن تعيين مساوي في البرمجة التبادل بشكل أنيق و خاص :

```
>>> a, b = b, a
```

(و يمكن بالطبع عكس متغيرات أكثر (مثلا ثلاثة) في نفس الوقت بنفس التعليمة)

حلقات التكرار - العبارة وايل

في البرمجة, نسمي الحلقة التكرار بنظام التعليمة الذي يكرر المهمة المحددة عدة مرات (أو بشكل لا نهائي) جميع العمليات .البايثون يقترح عبارتان لإستعمال الحلقات : العبارة **for** ... **in** , قوية جدا و نحن سندرسها في الفصل العاشر و البيان **while** الذي سندرسه الأن .

الرجاء إدخال الأوامر التالية :

```
>>> a = 0
>>> while (a < 7):      #! لا تنسى النقطين
...   a = a + 1        #! لا تنسى مسافة البادئة
...   print(a)
```

إضغط مرة أخرى على إنتر .

ماذا حدث ؟

قبل قراءة التعليقات التالية , خذ وقتا لفتح دفتر الملاحظات و دون هذه السلسلة من الأوامر . وصف أيضا نتيجة ذلك و حاول تفسر كيف حدث هذا بأكبر قدر ممكن من التفصيل .

التعليقات

تعني كلمة **while** بالإنكليزية "عندما" . هذه العبارة المستخدمة في السطر الثاني تخبر البايثون على أنه يجب عليه تكرار بإستمرار الكتلة التالية من البيانات , عندما يكون المتغير **a** أقل من 7 .

مثل عبارات **if** و التي ناقشناها في الفصل السابق, في حين تبدأ العبارة **while** بعبارة المجمع . النقتطين العاموديتين في نهاية السطر تخبر البايثون أنه سيبدأ بدخول مجمع الذي يجب تكراره و الذي يجب أن يبدأ بمسافة . كما تعلمت في الفصل السابق يجب أن تكون بادئة جميع البيانات داخل الكتلة متساوية بالضبط في نفس المستوى (وهذا يعني البدأ بالعدد نفسه من المسافات) .

لقد قمنا ببناء حلقتنا الأولى , التي تكرر كتلة البادئة للبيانات عدة مرات , و لكن كيف تعمل :

- مع العبارة وايل , يبدأ البايثون بالتأكد من الشرط الموضوع داخل القوسين (القوسين إختياري و نحن إستخدمناها للتوضيح فقط)
- إذا كان الشرط غير صحيح , يتم تجاهل الكتلة كلها و يتم إنهاء البرنامج[5].
- إذا كان الشرط صحيحا , يقوم البايثون بتنفيذ كتلة البايانات و التي تشكل هيئة حلقة و هذا معناه :
- - إستدعاء دالة الطباعة **print()** لإظهار قيمة الحالية للمتغير **a** إذا كان الشرط صحيحا , يقوم البايثون بتنفيذ كتلة البايانات و التي تشكل هيئة حلقة و هذا معناه :

- التعليمة **a = a + 1** معناها زيارة واحد إلى قيمة المتغير (و هذا معناه إعادة تعيين قيمة لمتغيير بقيمته القديمة زائد واحد) .

- - إستدعاء دالة الطباعة **print()** لإظهار قيمة الحالية للمتغير **a**
- و عند تنفيذ هتان التعليمتان يتم الرجوع مرة أخرى إلى عبارة التكرار, و هذا معناه الرجوع إلى عبارة التكرار و إجراء الشرط إذا كان صحيح يكمل و إذا كان غير صحيح يخرج .

في مثالنا هذا , إذا كان الشرط **a < 7** لايزال صحيح , فيتم تنفيذ الحلقة مرت أخرى و تستمر الحلقة .

5.. على الأقل في هذا المثال . سوف نرى فيما بعد أن التنفيذ يستمر مع أول تعليمة التي تلي كتلة البادئة, و التي هي جزء من نفس كتلة العبارة **while** نفسها .

ملاحظات

- يجب أن يكون المتغير الذي يتم إختباره موجود بالفعل (أي أنه يجب صنع المتغير و يجب أن يكون يحتوي على قيمة مثلا : 1)
- إذا كان الشرط غير صحيح من البداية , لن يتم تنفيذ ما داخل الحلقة أبدا (الكتلة)
- إذا كان الشرط صحيحا دائما, يتم تكرار الحلقة إلى اللانهاية (عندما تكون البايثون يعمل) .لذا يجب علينا أن نضع في الكتلة حلقة واحدة على الأقل التي تغير قيمة المتغير الذي يؤثر على الحلقة في الوقت المناسب (حتى أن يصبح هذا الشرط غير صحيح و تنتهي الحلقة)
- مثال على الحلقة اللانهائية (تجنبها !):

```
>>> n = 3
>>> while n < 5:
...   print("hello !")
```

تطوير الجداول

أعد كتابة التمرين الأول مع تغيير طفيف أدناه :

```
>>> a = 0
>>> while a < 12:
...   a = a +1
...   print(a , a**2 , a**3)
```

يجب عليك أن تحصل على قائمة من المربعات و المكعبات من 1 إلى 12 .

إعلم أن تستطيع أن تمرر أكثر من برامتر في دالة الطباعة **print()** لإظهار عدة تعبيرات في نفس الوقت واحدة تلو الأخرى على نفس الخط : فقط ضع فاصل بين كل واحدة و أخرى . البايثون سيضع مسافة بين العناصر المعروضة .

البناء الرياضي

البرنامج الصغير الذي بالأسفل يعرض عشرة أول أرقام من تسلسل يسمى ب " تسلسل فيبواناتشي " . هذه هي سلسلة من الأراقم كل رقم من هذه السلسلة يساوي مجموع رقمين سابقين من نفس السلسلة . جرب تحليل هذا البرنامج (الذي يستخدم التعيين الموازي) و صف أكبر قدر ممكن دور كل سطر من التعليمات .

```
>>> a, b, c = 1, 1, 1
>>> while c < 11 :
...   print(b, end =" ")
...   a, b, c = b, a+b, c+1
```

عندما تشغل البرنامج ستحصل على ما يلي :

```
1 2 3 5 8 13 21 34 55 89
```

يتم عرض تسلسل فيبوناتشي على نفس السطر . هذا ما سنفهمه حتى البارامتر الثاني **end = ''** التي بها دالة الطباعة . إفتراضيا, دالة الطباعة **print()** تقوم بإضافة رمز خاص لا يظهر ليقوم بالقفز إلى السطر التالي عندما نقوم بعرض شيئا ما . و البارامتر **end=''** يخبر البايثون أن يستبدل القفزة ب مسافة صغيرة . إذا حذفت هذا البارامتر, سيتم عرض الأرقام واحد تحت الأخر .

في برامجك المستقبلية, سوف تضع في برامجك في كثير من الأحيان حلقات تكرارية كما التي حللنها هنا .و سيتبادر إلى ذهنك سؤال أساسي, هل ستتعلم البرمجة بإتقان . تأكد أنك ستصل إلى هناك تدريجيا , بفضل التمارين .

عندما تختبر مشكلة من هذا النوع , يجب عليك النظر إلى الأسطر التعليمات , بطبيعة الحال, لكن أنظر خاصة في المتغيرات المختلفة المشاركة في الحلقة . هذا ليس دائما سهل , على العكس ذلك لمساعدتك على النظر بوضوح, بدون تكبد مشقة رسم جدول على الورق وضعنا في الأسفل جدول يشرح برنامجنا " سلسلة فيبوناتشي " :

المتغيرات	a	b	c
القيم الأولية	1	1	1
القيم التي تعيينها خلال التكرارات	1 2 3 5 ...	2 3 5 8 ...	2 3 4 5 ...
عبارة الإستبدال	b	a+b	c+1

في هذا الجدول , الذي صنعناه إلى حد ما "بأيدينا" , مشيرا سطرا بسطر المتغيرات التي تأخذ كل واحدة من هذه المتغيرات كما في التكرارات القادمة . سنبدأ ب بكتابة في أعلى الجدول أسماء المتغيرا المعنية . في السطر التالي, القيم المبدئية لهذه المتغيرات (القيم قبل بداية الحلقة). أخيرا في الجزء السفلي من الجدول ,نضع التعبيرات المستخدمة داخل الحلقة لتغيرر قيمة كل متغير في كل تكرار .

إملأ بضعة سطور المقابلة للتكرار الأول . لصنع متغير لسطر , فقط طبق ما فعناه في السطور السابقة, والتعبير عن الإستبدال موجود أسفل كل عمود . إفحص و إحصل على نتيجة البحث . إذا لم تكن في هذه الحالة جيدة , يجب عليك البحث عن تعبير أخر .

تمارين

4.2 أكتب برنامج يعرض أو 20 نتيجة لعملية الضرب على 7

4.3 أكتب برنامج لعرض جدول لتحويل اليورو إلى الدولار الكندي , تبدأ بالزيادة إلى الجدول ستكون "هندسية" , مثل التالي :

يورو 1 = 1.65 دولار

يورو 2 = 3.30 دولار

يورو 4 = 6.60 دولار

يورو 8 = 13.20 دولار

إلخ ... (توقف عند 16384 يورو)

4.4 أكتب برنامج الذي يعرض 12 رقم كل واحد من هذه الأرقام يساوي 3 مرات الرقم الذي قبله .

سكريبتك الأول, أو كيفية حفظ برامجنا

حتى الأن , نحن نستخدم البايثون في وضع تبادلي (و هذا معناه أنه كل مرة أدخلت الأوامر لم يتم حفظها) . و هذا يسمح لك بتعلم الأساسيات اللغة بسرعة , من خلال التجارب مباشرة هذه الطريقة لديها عيب واحد كبير : كل التعليمات التي تكتبها تختفي عند إغلاق المفسر . قبل مواصلة دراستك, حان الوقت لتعلم كيفية حفظ برامجك في ملفات على القرص الصلب أو مفتاح يو أس بي(USB) , بطريقة لتستطيع إعادة عمل المراحل , و نقلها على حواسيب أخرى , ...إلخ

للقيام بذلك, سوف تكتب الأن التعليمات الخاصة بك على أي محرر (مثل **Kate** و **Geany** و **Gedit** ...في لينكس و **wordpad** و **Komodo** و **Geany** ...على ويندوز, أو أكتب في واجهة التطوير الرسومية **IDLE** التي توزع مع البايثون في ويندوز)

و هكذا تكتب السكريبت, و ثم تستطيع حفظه و تغيره و نقله و إلخ ... مثل أي مستند أخر مكتوب بالحاسوب .

بعدها, عندما ترغب في إختبار تنفيذ البرنامج الخاص بك , يجب عليك فتح مفسر البايثون و أكتب (أكتب كأنه بارامتر) إسم الملف الذي يحتوي على السكريبت . على سبيل المثال, إذا كتبت السكريبت داخل ملف يدعى "**MyScript**", يكفي أن تكتب هذا الأمر ليشتغل :

```
python3 MyScript [6]
```

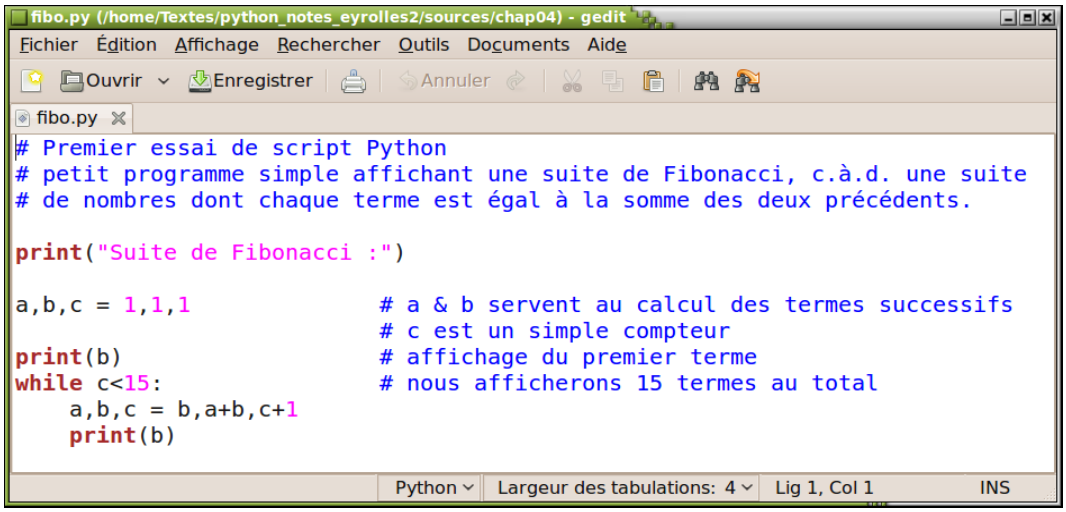
من الأفضل أن تتأكد من أن إسم ملف البرنامج ينهي ب **.py**.

إذا أتبعت هذه النصيحة , يمكنك تشغيل الملف النصي لبرنامج , و ذلك ببساطة عن طريق النقر على إسمه أو رمزه في مدير الملفات(و هو مستكشف في ويندوز, أو **Nautilus** أو **Konqueror** على لينكس ...)

6إذا تم تثبيت البايثون 3 على جهازك كمفسر بايثون إفتراضي, يجب أن تكون قادر على إدخال ببساطة : **python MyScript** . لكن إنتبه : إذا كانت هنالك إصدارات متعددة من البايثون مثبة على جهازك, ربما سيتم إستخدام إصدار سابق من بايثون (الإصدار 2) .

هؤلاء مديري العرض يعرفون أنهم يفتحون البايثون مع أي ملف ينتهي ب **.py** (و هذا بالطبع يجب أن يكون قد تم تكوينها بشكل صحيح) . نفس الشيئ مع المحررات "الذكية" التي تعرف تلقائيا سكريبتات البايثون و تتكيف مع بناء التعليمات .

الشكل التالي يوضح إستخدام محرر Gedit على لينكس "أبنتو" لكتابة السكريبت :



سكريبت البايثون يحتوي على سلسلة تعليمات مماثلة لتلك التي إختبرناها الأن. و سوف تخزنها و بعد مدة سوف تشغلها و تقرأها من قبلك أو من قبل الأخرين , وينصح و بشدة توضيح النصوص الخاصة بك إلى أقصى حد ممكن , و يجب أن تتضمن الكثير من التعليقات , و الصعوبة الحقيقية في تطوير البرامج هي الخوارزميات الصحيحة , بحيث يمكن التأكد من هذه الخوارزميات و تصحيحها و تغييرها و ما إلى ذلك , في أفضل ظروف , و من الضروري أن يصف المبرمج الكلمات جيدا و بأكبر قدر ممكن من الوضوح .

> المبرمج الجيد يدخل دائما أكبر عدد من التعاليق في سكريبتاته . و بذلك, لايسهل فهم الخوارزميات للقراء المحتملين الآخرين فقك, و لكنه يفرض أن سكريبت يكون أكثر وضوحا .

و أفضل مكان لهذا الوصف هو في جسم السكريبت (بحيث لا يضيع)

يمكننا إدراج تعليقات من أي نوع في أي مكان تقريبا في البرنامج النصي . ببساطة ضع قبل التعليق الرمز # حيث يعتبر المفسر هذا الرمز هو دلالة على تجاهل كل ما يأتي بعد هذا الرمز إلى نهاية السطر .

يرجى منك أن تفهم أنه يجب عليك أن تضع التعليقات كلما تقدمت في عملك في البرمجة . لا تنتظر حتى تنتهي من السكريبت الخاص بك . عليك أن تدرك أن المبرمج ينفق الكثير من الوقت لقراءة التعليمات البرمجية الخاصة بك (على سبيل التغيير, و البحث عن الأخطاء , إلخ ...) و سيكون هذا سهلا إذا وضعت العديد من التعليقات و الملاحظات و التفسيرات .

إفتح محرر النصي , و أكتب السيناريو التالي :

```
# أول إختبار لسكريبت بايثون .
# برنامج صغير و بسيط يعرض تسلسل فيبوناتشي .
# كل عدد في هذه السلسلة يساوي مجموع إثنين سابقين .
a, b, c = 1, 1, 1        # تستخدم b و a لحساب الأعداد المتتالية
                         # c هو عداد بسيط
print(b)                 # عرض العدد الأول
while c<15:              # سوف نعرض 15 عدد
  a, b, c = b, a+b, c+1
  print(b)
```

لنظهر لكم هذا الأن المثال جيد , نحن بدأنا هذا السكريبت ب ثلاثة أسطر من التعليقات, التي تحتوي على وصف سريع لعمل البرنامج . إجعل هذه عادة, بفعل نفس الشيئ في سكريباتاتك .

يجب أن تكون الأسطر البرمجية موثق تماما . إذا فعلت مثلما نحن فعلنا هنا , و هذا معناه أنك وضعت التعليقات بجانب الأسطر البرمجية , تأكد أنك أزلت ما يكفي لكي لا تتداخل مع القراءة .

عندما تريد التأكد من نصك, إحفظ البرنامج و شغله .

> و على الرغم من أن هذا ليس ضروريا, نحن نقترح عليك إختيار أسماء الملفات النصية التي تنتهي من .**py** . لأنه سوف يساعدك كثيرا على التعرف إليه . و مدراء

> الملفات الرسومية (مستكشف ويندوز, نوتيلوسو كونكيورر) تستخدم هذا الإمتداد لوضع أيقونة معينة . و يجب عليك أيضا تجنب إختيار الأسماء التي هي أسماء وحدات بايثون بالفعل : مثل أسماء ***math.py*** أو ***tkinter.py*** .

إذا كنت تعمل في الوضع النصي في لينكس أو في نافذة دوس تستطيع تشغيل سكريبتك بمساعدة الأمر **python3** بالإضافة لإسم السكريبت . إذا كنت تعمل في الوضع الرسومي في لينكس , تستطيع فتح نافذة الطرفية و فعل نفس الشيئ :

```
python3 myScript.py
```

في مدير الملفات الرسومي , يمكنك البدء في تشغيل البرنامج النصي بالأضغط على أيقونته . لكن هذا لا يمكن إلا إذا كان البايثون 3 تم تعيينه كالمفسر الإفتراضي لملفات التي تنتهي ب **.py** (قد تحدث مشاكل في الواقع إذا كان يوجد عدة إصدارات من البايثون مثبتة على جهازك . راجع معلم للتفاصيل) .

إذا كنت تعمل مع **IDLE** , يمكنك تشغيل البرنامج النصي الذي كتبته , مباشرة بإستخدام **Ctrl + F5** . في بيئات عمل أخرى متخصصة لتحديد البايثون, مثل **Geany** ستجد رموز أو إختصار للوحة المفاتيح لتشغيل البرنامج(و هذا يكون غالبا المفتاح **F 5**). إستشر شخصا أو معلم عن طرق تشغيل الأخرى على أنظمة تشغيل مختلفة .

مشاكل محتملة مع الرموز

إذا قمت بكتابة السكريبت الخاص بك مع المحرر الأخير (مثل التقارير التي لدينا)ينبغي أن السيناريو المذكور أعلاه ينبغي أن يعمل بدون مشاكل مع الإصدار الحالي مع البايثون 3 . إذا كان البرنامج قديم أو تم تكوينه بشكل غير صحيح , قد تحصل على رسالة مشابه لهذه :

```
File "fibo2.py", line 2
SyntaxError: Non-UTF-8 code starting with '\xe0' in file fibo2.py on line 2, but no encoding
declared; see http://python.org/dev/peps/pep-0263/ for details
```

هذه الرسالة تشير إلى أن السكريبت يحتوي على أحرف مطبعية مشفرة وفقا لمعايير قديمة (ربما ISO-8859-1 أو Latin-1 .)

نحن سنشرح المقاييس الترميز المختلفة في وقت لاحق في هذا الكتاب , في الوقت الراهن , إعلم أنك ستحتاج في هذه الحالة :

- إما بإعادة تكوين محرر النص بحيث يتم الترميز ب رموز **utf-8** (أو الحصول على المحرر يعمل بهذه المعايير) . هذا هو الحل الأفضل, لأنه بعد ذلك يمكنك التأكد في المستقبل أن العمل وفقا للإتفاقيات الحالية لتوحيد المقاييس , و التي سوف عاجلا أو أجل يتم إستبدال جميع القديم .
- أو أن تشمل هذا التعليق في السطر الأول أو الثاني:

  # -*- coding:Latin-1 -*-

  الشبه تعليق الذي فوق يخبر البايثون بإستعمال السكريبت الخاص بك برموز أكسي المقابلة لأهم لغات أروبا الغربية (الفرنسية و الألمانية و البرتغالية ...إلخ) المشفر على بايت واحد بعد **ISO** ..... و غالبا ما يشار إليه في التسمية ب **Latin-1** الشبه تعليق الذي فوق يخبر البايثون بإستعمال السكريبت الخاص بك برموز **ACSII** المقابلة لأهم لغات أروبا الغربية (الفرنسية و الألمانية و البرتغالية ...إلخ) المشفر على بايت واحد بعد **ISO** ..... و غالبا ما يشار إليه في التسمية ب **Latin-1**

يمكن للبايثون التعامل على النحو المناسب مع الأحرف المشفرة للمعايير . لذلك لمساعدة البايثون يجب أن تضع التعليق في البرنامج النصي . من دون هذه الإشارة , البايثون يفترض أنه تم ترميزه ب **utf-8**[7], تحت معيار اليونيكود الجديدة, و التي تم تطويرها لتوحيد التمثيل الرقمي لجميع الرموز من لغات العالم المختلفة بالإضافة لرموز الرياضيات و العلوم ...إلخ . هنالك العديد من الترميزات في هذا المعيار, سنعمل على هذه المسألة , في الوقت الراهن, نعرف أن تماما أن الترميز الأكثر شيوعا في أجهزة الحاسوب الحديثة هو **utf-8** . في هذا النظام , الترميز القياسي (**ACSII**)في بايت واحد, و الذي يوفر بعض التوافق مع ترميز **Latin-1** و لكن الأحرف الأخرى (بما في ذلك الأحرف المعلمة)يمكن ترميزها على 2 أو 3 أو حتى 4 بايت .

---

[7] في الإصدارات السابقة للبايثون, كان الترميز الإفتراضي هو ASCII . و لذلك كان يجب دائما وضع في بداية السكريبت الترميز (بما في ذلك Utf-8) .

سوف نتعلم كيفية إدارة و تحويل هذه الترميزات , عندما ندرسها بالتفصيل في معالجة الملفات النصية (الفصل التاسع)

تمارين

4.5 أكتب برنامج الذي يقوم بحساب متوازي المستطيلات إعتمادا على الطول و العرض و الإرتفاع

4.6 أكتب برنامج يحول الثواني إلى سنوات و أشهر و أيام و ساعات و دقائق و ثواني (أستخدم المعامل موديلو %)

4.7 أكتب برنامج الذي يعرض أول 20 نتيجة لجدول الضرب على سبعة,مشيرا بنجمة مضاعفات العدد 3 .

على سبيل المثال :

7 14 21 * 28 35 42 * 49 ...

4.8 أكتب برنامج لحساب أول 50 نتيجة بجدول ضرب 13 , لكن لن يظهر سوى التي تتقسم على 7 .

4.9 أكتب برنامج يعرض التالي :

```
*
**
***
****
*****
******
*******
```

5

أهم أنواع البيانات

في الفصل الثاني,لقد تعاملنا بالفعل مع العديد من أنواع البيانات : الأعداد الصحيحة أو الحقيقية و السلاسل النصية . حان الوقت الأن لنختبر و نقترب قليلا إلى هذه الأنواع من البيانات , بالإضافة إلى تعرف على أنواع أخرى .

البيانات الرقمية

في تمارينا حتى الأن , لقد إستخدمنا البيانات من نوعين : صحيحة العادية و الأعداد الحقيقية (و تسمى أيضا العائمة و أرقام النقطة الواحدة) سنسعى الأن جاهدين لتسليط الضوء على خصائص (و قيود) هذه المفاهيم .

العدد الصحيح

لنفترض أننا نريد عمل تعديل طفيف على تمارينا السابقة بشأن تسلسل فيبوناتشي ,و ذلك للحصول على عرض أكبر عدد من المصطلحات .مبدئيا , سنغير شرط الحلقة , في السطر الثاني, مع **c < 50**: سنغيرها لنحصل على 49 شرط , دعونا نغير التمرين قليلا , لعرضه كنوع رئيسي للمتغير :

```
>>> a, b, c = 1, 1, 1
>>> while c <50:
    print(c, ":", b, type(b))
    a, b, c = b, a+b, c+1
```

```
...
...
... (affichage des 43 premiers termes)
...
44 : 1134903170 <class 'int'>
45 : 1836311903 <class 'int'>
46 : 2971215073 <class 'int'>
47 : 4807526976 <class 'int'>
48 : 7778742049 <class 'int'>
49 : 12586269025 <class 'int'>
```

التمرين الذي قمنا به للتو , يمكن للعدد الصحيح يمكننا من معرفة التمثيل الداخلي للأرقام في الحاسوب . و ربما كنت تعرف أن في الواقع في قلب الحاسوب يتكون من دوائر متكاملة إلكترونية (رقاقات السيليكون) تم دمجها بطريقة عالية , و التي يمكن أن تؤدي لأكثر من مليار عملية في الثانية , و لكن الأرقام الثنائية محدودة الحجم : حاليا أجهزة 32 بت ,[8]. مجموع القيم العشرية التي يمكن ترميزها في صيغة أرقام ثنائية 32 بت هو من 2147483648- إلى 2147483647+ .

العمليات على الأعداد الصحيحة بين هذين الحدين هي دائما سريعة جدا , و ذلك لأن المعالج قادر على التعامل معهم بسرعة , و مع ذلك عندما يتعلق الأمر مع أكبر الأعداد الصحيحة , أو الأعداد الحقيقية (أرقام ذات فاصلة), ستقوم المفسرات و المجمعات بعمل كبير بترميز\فك ترميز , وهذا موجود في عمليات المعالج النهائية على الأعداد الثنائية , 32 أقصى حد .
لا يوجد شيئ يدعوا للقلق حول هذه الإعتبارات التقنية . عندما تطلب منه معالجة أي عدد صحيح يقوم البايثون بمعالجة هذه الأرقام و تحويلها إلى شكل من أشكال الأرقام الثنائية 32 بت, لتحسين سرعة الحساب و توفير المساحة للذاكرة . و عندما تكون القيم إلى أن التعامل مع الأعداد الصحيحة تكون خارج الحدود

[8] معظم أجهزة الحاسوب المكتبية تحتوي على معالج بسجلات 32 بت (حتى لو كان ثنائي النواة) . سوف تكون معالجات 64 بت قريبا شائعة .

المشار إليها أعلاه , الترميز في الذاكرة الحاسوب يصبح أكثر تعقيدا, و البايثون يعالج هذا الرقم من قبل معالج يتتطلب عمليات متتالية عديدة . و يتم كل هذا تلقائيا , دون الحاجة للقلق [9].

ستستطيع إذا مع البايثون حساب أعداد صحيحة تحتوي مع أي أعداد كبيرة . و يقتصر هذا العدد في الواقع على حجم الذاكرة المتوفرة على الحاسوب المستخدم . إن الحسابات التي تتضمن أعداد كبيرة جدا

سيتم تقسيمها من قبل المترجم على عدة حسابات أكثر بساطة , و هذا قد يتطلب قدرا أكبر من الوقت لمعالجتها في بعض الحالات .

على سبيل المثال :

```
>>> a, b, c = 3, 2, 1
>>> while c < 15:
    print(c, ": ", b)
    a, b, c = b, a*b, c+1

1 :  2
2 :  6
3 :  12
4 :  72
5 :  864
6 :  62208
7 :  53747712
8 :  3343537668096
9 :  179707499645975396352
10 :  6008587943056673222701554251857 92
11 :  10797883156496691381438492294473845785924307043903 0784
12 :  648800305446607527907368373691049776950010342842280428918276494561862 34
582611607420928
13 :  700566989011183200292376413995762169216245450579726979173836923132717 54
883621235064434673400268965204696103008832506249008437424702378475 52
```

[9] في الإصدارات السابقة للبايثون لديها نوعين من الأعداد الصحيحة : **integer** و **long integer**, لكن التحويل بين هذين النوعين أصبح تلقائيا منذ الإصدار 2.2 .

```
14 : 4545280764562657998563629404824935120516823987072294615140165565565839
8
6422276163358151238257824601969802061415367471160941735505142279479530059170
0
9695042269307903824763405582917529683194622450393350175477603300401275836825
6
>>>
```

في المثال أعلاه, فإن قيم الأرقام تم عرضها بشكل سريع جدا , لأن كل لأن كل واحدة منها تساوي إثنين من الشروط السابقة. بالطبع, يمكنك الإستمرار في هذا التسلسل الرياضي إذا كنت تريد , سيزداد نمو الأعداد الهائلة , و لكن سرعة الحساب تنخفض تدريجيا .

> الأعداد الصحيحة للقيمة التي تشمل حدين مشار إليها في الأعلى تحتل كل واحدة 32 بت في ذاكرة الحاسوب . الأعداد الصحيحة الكبيرة جدا تحتل مكان متغير, إعتمادا على حجمها .

الأعداد الحقيقة

لقد تعرفنا في وقت سابق هذا النوع من البيانات الرقمية , عدد "نوع الحقيقي" مشار إليها في اللغة الإنكليزية من قبل أرقام النقطة و ذلك السبب يتم تسميتها الحقيقة في البايثون .

و هذا يسمح القيام بالعمليات الحسابية على أعداد كبيرة جدا أو أعداد صغيرة جدا (البيانات العلمية على سبيل المثال), مع وجود درجة ثابة من الدقة .

للحصول على أعداد رقمية بواسطة البايثون تعتبر من النوع العشري, يكفي أن في صيغته يحتوي على رمز مثل نقطة أو رقم أس(أو قوة) 10 .

على سبيل المثال :

```
3.14    10.    .001    1e100    3.14e-10
```

يتم تفسير العبارات السابقة بأنها أرقام حقيقية تلقائية من البايثون . دعونا إذا نجرب هذا النوع من البيانات في برنامج صغير (مستوحى من المثال أعلاه) :

```
>>> a, b, c = 1., 2., 1          #=> " float - عدد عشري" هي أعداد من نوع a و b
>>> while c <18:
...    a, b, c = b, b*a, c+1
...    print(b)

2.0
4.0
8.0
32.0
256.0
8192.0
2097152.0
17179869184.0
3.6028797019e+16
6.18970019643e+26
2.23007451985e+43
1.38034926936e+70
3.07828173409e+113
4.24910394253e+183
1.30799390526e+297
     Inf
     Inf
```

كما فهمت جيدا , و نعرض لك مرة أخرى سلاسل من الأرقام ,التي تم إنهائها بسرعة كل واحدة منها تساوي نتيجتين سابقتين . في النتيجة التاسعة , تحول البايثون تلقائيا إلى علمي ("**E** + رقم معناه "عشرة أضعاف أس\قوة الرقم) بعد 15, نحن نرى أنها فاقت الحد (بدون رسالة خطأ) , الأرقام في الواقع كبيرة جدا و لاحظ ببساطة عبارة (**Inf**) لتعبر عن اللانهاية .

الأرقام الحقيقية المستخدمة في مثالنا نستخدم لمعالجة الأعداد(الموجبة و السالبة) تكون بين 10**-308 و 10**308 مع دقة . هذه الأرقام يتم ترميزها بطريقة معينة على 8 بايت (64 بت) في ذاكرة الحاسوب : جزء من الشيفرة يتوافق مع 12 رمز كبير و أخر بحجم (أس 10)

تمارين

5.1 أكتب برنامج يحول زاوية بالرديان المقدمة أصلا بالدرجة و الدقائق و الثواني .

5.2 أكتب برنامج يحول الدرجة و الدقائق و الثواني زاوية زودت أصلا بالراديان .

5.3 أكتب برنامج يحول درجة الحرارة من الدرجة المئوية إلى الفهرنهايت أو العكس
صيغة التحويل : $T_F = T_C \times 1{,}8 + 32$.

5.4 أكتب برنامج يقوم بحساب الفؤائد المحققة سنويا لمدة 20 عاما , برأس مال قدره 100 يورو وضعت في البنك بمعدل ربح ثابت ب 4.8%

5.5 تقول أسطورة من الهند القديم أن لعبة الشطرنج أخترعت من قبل رجل يبلغ من العمر الحكمة, فشكره الملك و سيهديه أي هدية كمكافأة . أخبره العجوز أنه يريد ببساطة القليل من الأرز لأيامه السابقة . و على وجه التحديد , عدد من حبات الأرز تكفي لوضعها في أتش جي واحدة على المربع الأول للعبة التي إخترعها , إثنين للثانية و 4 للثالثة و هكذا إلى 64 مربع.

أكتب برنامج لحساب عدد الحبات الأرز في المربعات ال 64 في اللعبة . بطريقتين:

- العدد الدقيقي من الحبوب (عدد كامل)
- عدد الحبوب بالطريقة العلمية (العدد الحقيقي)

المعطيات الأبجدية

حتى الأن لم نتعامل سوى مع الأرقام, و لكن برنامج الحاسوب يحتاج أيضا التعامل مع الحروف الأبجدية و الكلمات و العبارات و السلاسل من الرموز . في معظم لغات البرمجة, هنالك هياكل لهذا الإستخدام و من المعروف بيانات المحددة مثل " السلاسل النصية" .

> سوف نتعلم في الفصل 10 أنه لا نبغي لنا أن نخلط بين مصطلحي "سلسلة نصية" و "سلسلة بايتات" كما تعرضو للإنتهاكات في لغات البرمجة القديمة (بما في ذلك الإصدارات القديمة للبايثون) . و في الوقت الراهن, يقوم البايثون بمعالجة متماسكة لكل السلاسل النصية, التي يمكن أن تكون جزءا من الحروف الأبجدية [10] .

السلسلة

و يمكن تعريف نوع البيانات السلسلة تقريبا مثل أي سلسلة من الحروف . في سكريبت البايثون , يمكن للمرء أن يعرف مثل هذه السلاسل من الأحرف . إما عن طريق علامات التنصيص المفردة أو علامات التنصيص الزوجية (علامة إقتباس) . أمثلة على ذلك :

```
>>> phrase1 = 'les oeufs durs.'
>>> phrase2 = '"Oui", répondit-il,'
>>> phrase3 = "j'aime bien"
>>> print(phrase2, phrase3, phrase1)
"Oui", répondit-il, j'aime bien les oeufs durs.
```

المتغيرات الثلاثة : **phrase1** و **phrase2** و **phrase3** متغيرات من نوع سلسلة .

لاحظ إستخدام علامات الإقتباس لتحديد السلسلة التي توجد فيها علامة تنصيص مفردة, أو إستخدم علامة الإقتباس المفردة , لاحظ أيضا مرة أخرى أن دالة الطباعة تدرج مسافة بين العناصر المعروضة .

الرمز الخاص "\" (الخط المائل) يسمح ببعض المميزات الإضافية :

---

[10] و لذلك فإنها واحدة من المميزات الرئيسية للإصدار الجديد للبايثون (بايثون 3) مقارنة بالإصدارات السابقة . و في هذه النسخة, البيانات من نوع string كانت سلسلة من البايتات و ليس سلسلة من الحروف . و هذا لا يشكل مشكلة كبيرة في التعامل مع نصوص التي تحتوي فقط على الحروف الرئيسية للغات أروبا الغربية, لأنه كان من الممكن ترميز كل هذه الأحرف في بايت واحد (على سبيل المثال, معيار Latin-1) . و هذا أدى إلى صعوبة كبيرة إذا أردنا جميع الأحرف في نص واحد من الحروف الأبجدية المختلفة, أو ببساطة إستخدام الحروف الأبجدية التي تحتوي على أكثر من 256 من الحروق و الرموز الرياضية الخاصة ...إلخ . يمكنك العثور على المزيد من المعلومات حول هذا الموضوع في الفصل 10 .

- أولا, لأنها تتيح لك كتابة أسطر متعددة التي من شأنها أن تأخذ وقتا طويلا لإحتواها على سطر واحد(هذا يتطبق على أي نوع من التعليمات)
- ضمن السلسة يتم إستخدامها لإدخال عدد من الرموز الخاصة(سطر جديد, علامة تنصيص مفردة, علامات الإقتباس, إلخ ...) أمثلة على ذلك :

```
>>> txt3 = '"N\'est-ce pas ?" répondit-elle.'
>>> print(txt3)
"N'est-ce pas ?" répondit-elle.
>>> Salut = "Ceci est une chaîne plutôt longue\n contenant plusieurs lignes \
... de texte (Ceci fonctionne\n de la même façon en C/C++.\n\
...    Notez que les blancs en début\n de ligne sont significatifs.\n"
>>> print(Salut)
Ceci est une chaîne plutôt longue
 contenant plusieurs lignes de texte (Ceci fonctionne
 de la même façon en C/C++.
   Notez que les blancs en début
 de ligne sont significatifs.
```

ملاحظات

- إن رمز **\n** في السلسة معناه القفز إلى سطر جديد
- إن الرمز **\'** لإدراج علامة تنصيص مفردة في سلسلة المحددة بواسطة علامة تنصيص مفردة و نفس الشيئ مع **\"** لإدخال علامات الإقتباس في سلسلة محددة بواسطة علامات الإقتباس .
- تذكر قواعد أسماء المتغيرات(يجب أن نحدد بدقة حالة الأحرف كبيرة أو صغيرة)

الإقتباس الثلاثي

لإدراج أحرف خاصة أو غريبة في السلسلة من دون إستخدام رموز المائل أو لإستعمال رمز المائل نفسه في السلسلة , يمكن للمرء أنه يمكنه إستعمال سلسلة بإقتباسات الثلاثة أو علانات التنصيص المفردة الثلاثية :

```
>>> a1 = """
...  Exemple de texte préformaté, c'est-à-dire
...    dont les indentations et les
...      caractères spéciaux \' " sont
... conservés sans
...    autre forme de procès."""
>>> print(a1)

 Exemple de texte préformaté, c'est-à-dire
   dont les indentations et les
     caractères spéciaux \' " sont
conservés sans
   autre forme de procès.
>>>
```

الوصول إلى الأحرف الفردية في السلسلة

السلاسل تمثل حالة خاصة من نوع من البيانات الأكثر عمومية تدعى مركب . المركب المعين هو الذي يجمع في واحدة مجموعة منه أكثر بساطة في حالة وجود سلسلة , على سبيل المثال, و هذه من الواضع أنه أبسط من الحروف نفسها. تبعا للظروف نحن نرغب بمعالجة السلسلة, و أحيانا الكائن الواحد , و أحيانا مجموعة أحرف . لغة البرمجة بايثون تسمح بالوصول بشكل منفصل على كل من الأحرف في السلسة, كما سوف ترى, و هذه ليست معقدة للغاية .

البايثون تفترض أن السلسلة هي كائن من فئة السلاسل, و هي التي طلبت مجموعات من العناصر . و هذا معناه ببساطة أن أحرف السلسلة ترتب دائما في ترتيب معين . و لذلك , يمكن لكل حرف في السلسلة أن يكون له تسمية في مكانه في السلسلة, و ذلك بإستخدام المؤشر .

للوصول إلى حرف محدد, يجب علينا وضع إسم المتغير الذي يحتوي على السلسلة و نضع رقم المؤشر (و هو رقم موضع الحرف في السلسلة) داخل معقفين .
إنتبه : سوف يكون لديك فرصة للتحقق (لاحقا) من أن الأعداد المرقمة تبدأ دائما من الصفر (و ليس واحد) . و هذا هو الحال بالنسبة لحروف السلسلة .

على سبيل المثال :

```
>>> ch = "Christine"
>>> print(ch[0], ch[3], ch[5])
C i t
```

تستطيع إعادة التمرين في الأعلى , و هذه المرة إستخدم حرف أو حرفين غير أكسي . على عكس ما قد يحدث في بعض الحالات مع الإصدارات البايثون قبل الإصدار 3.0, تحصل هنالك مفاجئة في النتيجة المتوقعة :

```
>>> ch ="Noël en Décembre"
>>> print(ch[1],ch[2],ch[3],ch[4],ch[8],ch[9],ch[10],ch[11],ch[12])
o ë l  D é c e m
```

لا داعي للقلق في الوقت الحالي حول كيفية يقوم البايثون بخزن و التعامل مع الأحرف في ذاكرة الحاسوب . فقط إعلم أن هذه التقنية تستغل المعايير الدولية يونيكود . فيستطيع تمييز أي حرف من الحروف الأبجدية . لذلك يمكنك خلط في نفس السلسلة اللاتينية و اليونانية و السيريلية و العربية ..إلخ و الرموز الرياضية و إلخ ...

> سوف نرى في الفصل 10 (أنظر لصفحة *Error: Reference source not found*) كيفية عرض الأحرف غير التي يمكن الوصول إليها مباشرة من لوحة المفاتيح .

العمليات الأساسية على السلاسل

البايثون يحتوي على العديد من الدوالات التي تؤدي إلى تعاملات مختلفة على سلاسل (الأحرف الكبيرة \ الصغيرة, قطع أجزاء من السلسلة و البحث عن كلمات ... إلخ) . مرة أخرى يجب أن تصبروا لأنه سيتم وضع شرح هذه الدوالات في الفصل 10 (أنظر صفحة 121)

الأن , يمكننا أن نعرف ببساطة أنه يمكن الوصول إلى كل حرف في السلسلة, كما هو موضح في الفقرة السابقة , دعونا نضيف قليلا على ما سبق :

- نجمع العديد من السلاسل الصغيرة لبناء واحدة كبيرة . هذا ما يسمى التسلسل و هذا يتحقق في البايثون بإستخدام الرمز + (هذا الرمز معناه إضافة سلسلة لسلسلة أخر مثل في الرياضيات و هذا يعمل في السلسلة النصية) مثال :

```
a = 'Petit poisson'
b = ' deviendra grand'
c = a + b
print(c)
petit poisson deviendra grand
```

- تحديد طول السلسة (أي عدد الأحرف) و ذلك بإستخدام **len()** :

```
>>> ch ='Georges'
>>> print(len(ch))
7
```

هذه الدالة تعمل بشكل جيد حتى لو كانت السلسلة تحتوي على أحرف أخرى :

```
>>> ch ='René'
>>> print(len(ch))
4
```

- تحويل رقم الذي يمثل سلسلة نصية إلى عدد رقمي , على سبيل المثال :

```
>>> ch = '8647'
>>> print(ch + 45)
```

→ *** خطأ *** : لا يمكن إضافة سلسلة إلى رقم

```
>>> n = int(ch)
>>> print(n + 65)
8712          #نعم : يمكننا إضافة رقم إلى رقم آخر
```

في هذا المثال, دالة **int()** تحول السلسلة إلى عدد صحيح . سيكون ذلك من الممكن أيضا تحويل سلسلة أحرف إلى العدد حقيقي, و ذلك بإستخدام **float()** .

تمارين

5.6 أكتب سكريبت يحدد إذا كانت السلسلة تحتوي على حرف "**a**" أو لا .

5.7 أكتب السكريبت يحسب عدد تواجد الحرف "**a**" في السلسلة .

5.8 أكتب سكريبت يقوم بنسخ سلسلة (في متغير جديد) و إدراج نجمة بين الأحرف, على سبيل المثال , "**gaston**" تصبح "**g*a*s*t*o*n**"

5.9 أكتب سكريبت يقوم بنسخ السلسلة (في متغير جديد) في الإتجاه المعكس على سبيا المثال : هشام تصبح ماشه .

5.10 إستنادا إلى التمارين السابقة , أكتب سكريبت يحدد إذا كانت السلسلة تعطي سياق متناظر أول لا (أي أن سلسلة يمكن قراءتها من الإتجاهين) , مثلا "Radar" أو "SOS" .

القوائم(النهج الأول)

قدمت السلاسل التي ناقشناها في الجزء السابق مثال أولي من البيانات المركبة . هياكل البيانات التي تستخدم لتجميع مجموعة من القيم . و سوف تتعلم تدريجيا لإستخدام غيرها من مركبات عدة أنواع من

البيانات . بما في ذلك , القوائم و القواميس و النفق[11]. و سوف نناقش هنا أول هذه الأنواع الثلاثة, و هذا و هذا مختصر إلى حد ما . لكنه بالفعل موضوع واسع جدا, و التي سنعود إليه مرارا و تكرارا .

في البايثون, يمكننا تحديد قائمة من العناصر مفصولة بفواصل موضوعة كلها داخل نصفي مربع , على سبيل المثال :

```
>>> jour = ['lundi', 'mardi', 'mercredi', 1800, 20.357, 'jeudi', 'vendredi']
>>> print(jour)
['lundi', 'mardi', 'mercredi', 1800, 20.357, 'jeudi', 'vendredi']
```

في هذا المثال, المتغير **jour** هو قائمة .

كما يمكن أن نرى في نفس المثال, فإن العناصر الفردية التي تشكل القائمة قد تكون من أنواع مختلفة . في هذا المثال, في الواقع, أول ثلاثة عناصر السلسلة هي حروف و العنصر الرابع هو عدد صحيح و الخامس هو عدد حقيقي و ما إلى ذلك . (سنناقشها لاحقا , يمكن للقائمة أن يكون أحد عناصرها هو قائمة !) في هذا الشأن , القائمة قد تكون ""مصفوفة" (**array**) أو " متغير إنديسا" حسب لغة البرمجة .

لاحظ أيضا , انه كما في السلاسل , أن القوائم هي سلسلة و هذا يعني أنها مرتبة. مختلف العناصر التي تشكل القائمة هي في الواقع دائما أعدت في نفس الترتيب . و يمكن الوصول إلى كل واحدة منها على حدة إذا كنا نعرف مؤشرها في القائمة . كما كان الحال الأحرف في السلسلة , يجب أن نعرف أن الترقيم يبدأ من الصفر و ليس واحد .

على سبيل المثال :

```
>>> jour = ['lundi', 'mardi', 'mercredi', 1800, 20.357, 'jeudi', 'vendredi']
>>> print(jour[2])
mercredi
>>> print(jour[4])
20.357
```

11 يمكنك إنشاء أنواع من البيانات المركبة بنفسك, عندما تتحكم في مفهوم الصنف. (أنظر إلى صفحة Error: Reference source not found).

على عكس السلاسل , و التي هي من النوع عدم التعديل البيانات (سيكون لدينا العديد من الفرص لزيادتها مرة أخرى على ذلك) , فمن الممكن تغيير عناصر الفردية للقائمة :

```
>>> print(jour)
['lundi', 'mardi', 'mercredi', 1800, 20.357, 'jeudi', 'vendredi']
>>> jour[3] = jour[3] +47
>>> print(jour)
['lundi', 'mardi', 'mercredi', 1847, 20.357, 'jeudi', 'vendredi']
```

يمكننا إستبدال بعض عناصر القائمة بأخرى , كما هو مبين أدناه :

```
>>> jour[3] = 'Juillet'
>>> print(jour)
['lundi', 'mardi', 'mercredi', 'Juillet', 20.357, 'jeudi', 'vendredi']
```

دالة **len()**, التي إستعملناها بالفعل في السلاسل , ينطبق نفس مفهومها على القوائم , لكن تقوم بإظهار عدد العناصر الموجودة في القائمة :

```
>>> print(len(jour))
7
```

دالة أخرى تقوم بحذف عنصر من القائمة (بإستخدام المؤشر) . و هذه الدالة هي **del()** [12] :

```
>>> del(jour[4])
>>> print(jour)
['lundi', 'mardi', 'mercredi', 'juillet', 'jeudi', 'vendredi']
```

و من الممكن إضافة عنصر إلى القائمة, و لكن للقيام بذلك, يجب علينا أن نعتبر القائمة كائن, و التي سوف نستخدم إحدى الطرق, سيتم شرح مفاهيم الحاسوب للكائن والأساليب في وقت لاحق , و لكن ما يهمنا الأن "كيف تعمل" في القائمة :

```
>>> jour.append('samedi')
>>> print(jour)
['lundi', 'mardi', 'mercredi', 'juillet', 'jeudi', 'vendredi', 'samedi']
>>>
```

في السطر الأول من المثال أعلاه , قمنا بطريقة **append()** لإضافة السبت للقائمة **jour** . لمن لا يعرف كلمة "**append**" تعني إضافة في الإنكليزية . و نحن نستطيع أن نفهم أن **append()** هو أسلوب يتم بطريقة أو بأخرى دمج أو إضافة عنصر إلى القائمة . البرامتر الذي يستخدم مع هذه الدالة هو بالطبع العنصر الذي نريد إضافته إلى نهاية القائمة .

سوف نرى لاحقا مجموعة كبيرة من هذه الطرق (و هذا معناه دالات بنيت, أو بالأحرى "غلفت" في نوع قائمة) . لاحظ أنه يتم تطبيق أسلوب الكائن من خلال ربطه مع النقطة. (الإسم الأول للمتغير الذي يشير للكائن , ثم نقطة ثم إسم الإسلوب, و هذا يكون دائما برفقة زوج من الأقواس) .

---

12 في الواقع يوجد عدد متنوع من التقنيات التي تسمح لك بقطع قائمة إلى شرائح, و إدراج مجموعات من العناصر, أو إزالة مجموعات أخرى ...إلخ . و ذلك بإستخدام تكوين جمل خاص الذي يتضمن المؤشر .
و تسمى هذه المجموعة من التقنيات (و التي يمكن أيضا تطبيقها على السلاسل) بالتشريح . بوضع عدة مؤشرات بدلا من الواحد بين قوسين (نصف مربع) ثم نضيف إسم المتغير ؟ مثل jour[1:3] الذي هو ['mardi', 'mercredi'] .
و سيتم شرح بالتفصيل هذه التقنيات لاحقا (أنظر لصفحة Error: Reference source not found و ما يليها).

كالسلاسل, سيتم التعمق في القوائم في وقت لاحق (أنظر الصفحة 141) . نحن لا نعرف ما يكفي للبدء في إستخدام برنامجنا . يرجى قراءة المثال . لتحليل السكريبت الصغير أدناه و التعليق على كيفية عمل ذلك :

```
jour = ['dimanche','lundi','mardi','mercredi','jeudi','vendredi','samedi']
a, b = 0, 0
while a<25:
    a = a + 1
    b = a % 7
    print(a, jour[b])
```

السطر الخامس في هذا المثال يستخدم المعامل "موديلو" الذي درسنا في وقت سابق و يمكن أن يكون ذا فائدة كبيرة في البرمجة . و يتم تمثيله ب **%** في العديد من لغات البرمجة (بما في ذلك البايثون) . حسنا , ما هي العملية التي يقوم بها هذا المعامل ؟

تمارين

5.11 أنظر في القوائم التالية :

```
t1 = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
t2 = ['Janvier', 'Février', 'Mars', 'Avril', 'Mai', 'Juin',
   'Juillet', 'Août', 'Septembre', 'Octobre', 'Novembre', 'Décembre']
```

أكتب برنامج صغير يقوم بإنشاء قائمة جديدة **t3** . و هذه القائمة يجب أن تحتوي على جميع العناصر من القائمتين بالتناوب , بحيث يتبع كل إسم عدد أيام الشهر .

```
['Janvier',31,'Février',28,'Mars',31, etc...].
```

5.12 أكتب برنامج الذي يعرض بشكل صحيح جميع عناصر القائمة . إذا طبقت على سبيل المثال إلى تي 2 الأوامر المذكورة أعلاه يجب أن تحصل على :

```
Janvier Février Mars Avril Mai Juin Juillet Août Septembre Octobre Novembre
Décembre
```

5.13 أكتب البرنامج الذي يبحث في القائمة على أكبر عنصر . فمثلا إذا طبقته على قائمة

[15 ,2 ,75 ,3 ,8 ,12 ,5 ,32]

يجب عليك أن تحصل على :

```
le plus grand élément de cette liste a la valeur 75.
```

5.14 أكتب برنامج الذي يقوم بفحص كل عناصر واحد تلو الأخر لقائمة أرقام (على سبيل المثال من التمرين السابق) لإنشاء قائمتين جديدتين .تحتوي الأولى على الأرقام الزوجية و الثانية على الأرقام الفردية , نصيحة : إشتخدم المعامل موديلو المذكور أعلاه .

5.15 أكتب برنامج الذي يحلل واحد تلو الأخر كل عناصر القائمة من الكلمات على سبيل المثال: ['جون'، 'ماكسيميليان'، 'بريجيت'، 'سونيا'، 'جان بيير'، 'ساندرا']) لتولد قائمتين جديد, واحد للكلمات أقل من 6 أحرف، و واحدة أخرى 6 أحرف أو أكثر.

6

الدالات المعرفة مسبقا

واحدة من أهم مفاهيم في البرمجة هي الدالة[13]. الدالة تجعلك تحلل برنامج معقد إلى سلسلة من البرامج بسيط, و التي بدورها يمكن تقسيمها إلى أجزاء أصغر, و هكذا . و إضافة إلى فإن الدالة قابلة لإعادة الإستخدام فمثلا إذا كان لدينا دالة تحسب الجذر التربيعي, يمكن أن نستخدمها في كل مكان في برنامجنا دون الحاجة إلى إعادة الكتابتها كل مرة .

دالة *print*()

لقد تعرفنا سابقا بهذه الدالة . هنا أردت أن أشير أنه يسمح عرض أي عدد من القيم المتوفرة كبرامترات (و هذا يعني ما بين أقواس) . إفتراضيا, سيتم فصل هذه القيم عن بعضها البعض بمسافة, و في النهاية يتم القفز إلى سطر جديد .

يمكننا إستبدال المسافة الإفتراضية بأي شيئ أخر (أو حتى بلا شيئ) من خلال البارامتر **sep** .
مثال :

```
>>> print("Bonjour", "à", "tous", sep ="*")
Bonjour*à*tous
 >>> print("Bonjour", "à", "tous", sep ="")
Bonjouràtous
```

و يمكنك إستبدال القفز إلى سطر جديد بإستخدام البارامتر **end** :

```
>>> n =0
```

13في البايثون, إن مصطلح "دالة" يستخدم للإشارة إلى الدالات الحقيقية و لكنه يستخدم أيضا للإشارة إلى الإجراءات . و سوف نشرح لاحقا التمييز بين هذين المفهومين المتشابهين .

```
>>> while n<6:
...     print("zut", end ="")
...     n = n+1
...
zutzutzutzutzut
```

التفاعل مع المستخدم : دالة input()

حاليا معظم مدخلات المستخدم تتم عن طريق (إدخال برامترات, النقر بواسط الفأرة, الضغط على زر في لوحة المفاتيح, إلخ ...) . في سكريبت الوضع النصي (مثل التي صنعناها حتى الأن) , أبسط طريقة هي إستخدام الدالة **input()** . هذه الدالة تسبب توقف البرنامج لتدعو المستخدم لإدخال حروف من لوحة المفاتيح و يجب أن ينتهي مع الضغط على زر الإدخال (Enter) . و عندما يضغط المستخدم زر الإدخال تقوم الدالة بأخذ ما كتبه المستخدم و يتم تعيينه لأي متغير و يمكن لهذه الدالة حتى تحويله .

يمكن للمبرمج إستدعاء دالة **input()** , و ترك الأقواس فارغة و يمكنه أيضا أن يضع بارامتر به رسالة تفسيرية للمستخدم , على سبيل المثال :

```
prenom = input("Entrez votre prénom : ")
print("Bonjour,", prenom)
```

أو :

```
print("Veuillez entrer un nombre positif quelconque : ", end=" ")
ch = input()
nn = int(ch)              # تحويل السلسلة إلى عدد صحيح
print("Le carré de", nn, "vaut", nn**2)
```

لاحظ أن دالة **input()** تقوم دائما بإرجاع سلسلة نصية[14]. فإذا كنت تريد أن يقوم المستخدم بإدخال قيمة رقمية, سوف تحتاج إلى تحويل قيمة المدخلات (و التي ستكون سلسلة نصية) إلى نوع الرقمي الذي يناسبك, من خلال وضع دالة **int()** (إذا كنت تتوقع عدد صحيح) أو **float()** (إذا كنت تتوقع عدد حقيقي) .على سبيل المثال :

[14] في الإصدارات السابقة للبايثون, القيم التي يتم إرجاعها من خلال **input()** كانت من نوع متغير, إعتمادا على ما قام المستخدم بإدخاله . السلوك الحالي كان سابقا دالة **raw_input()** , و الذي يفضله معظم المبرمجين .

```
>>> a = input("Entrez une donnée numérique : ")
Entrez une donnée numérique : 52.37
>>> type(a)
<class 'str'>
>>> b = float(a)        # تحويل السلسلة إلى عدد حقيقي
>>> type(b)
<class 'float'>
```

إستدعاء وحدة دالات

لقد تعرفت بالفعل على العديد من دالات اللغة , مثلا دالة **len()** , الذي يسمح بمعرفة طول السلسلة . مع ذلك , ليس من الممكن دمج جميع الدالات في البايثون القياسية , لأنه يوجد عدد لامتناهي من الدالات : و التي سوف تتعلم قريبا كيفية صنع دالات جديدة بنفسك . الدالات في البايثون القياسية هي قليلة نسبيا : فيوجد بها فقط الدالات التي يتم إستخدامها بشكل متكرر جدا , و البعض الأخر يتم وضعها في ملفات خاصة تدعى وحدات .

> الوحدات هي ملفات التي تحتوي على مجموعات من الدالات[15].

سترى في وقت لاحق أن تقسيم البرنامج إلى عدة ملفات أمر مريح لسهولة الصيانة . تطبيق البايثون يتألف من برنامج رئيسي, يرافقه وحدة واحدة أو أكثر , كل منها يحتوي على تعريفات عدد من الدالات الإضافية .

هنالك العديد من وحدات البايثون التي يتم توفيرها تلقائيا مع البايثون . تستطيع العثور على غيرها من مختلف المصادر . كثيرا ما نحاول جمع في نفس الوحدة مجموعة من دالات التي لها ذات الصلة , التي نسميها مكتبات .

وحدة **math** على سبيل المثال, تحتوي على تعريفات الدالات الرياضية مثل الجذر التربيعي ... إلخ . لإستخدام هذه المميزات, يمكنك ببساطة إدراج السطر التالي في بداية السكريبت الخاص بك :

```
from math import *
```

[15] بالمعنى الدقيق للكلمة, يمكن للوحدة أن تحتوي أيضا على معرفات المتغيرات و كذلك الأصناف . و سوف نترك هذه التفاصيل جانب (لبعض الوقت) .

هذا السطر يخبر البايثون أن يتم إدراج في البرنامج الحالي جميع الدالات (هذا ما معناه الرمز * "الجوكر") لوحدة الرياضيات, و التي تحتوي على دوالات رياضية مبرمجة مسبقا .

في داخل السكريبت , سوف تكتب على سبيل المثال

racine = sqrt(nombre) لتعيين للمتغير **racine** الجذر التربيعي ل **nombre**,

sinusx = sin(angle) لتعيين للمتغير **sinusx** جيب **angle** (بالراديان!), إلخ ...

على سبيل المثال :

```
# math تجربة : إستخدام دالات وحدة

from math import *

nombre = 121
angle = pi/6          # 30° إذا كانت
                      # (pi مكتبة الرياضيات تتضمن أيضا تعريف )
print("racine carrée de", nombre, "=", sqrt(nombre))
print("sinus de", angle, "radians", "=", sin(angle))
```

عندما تشغل هذا السكريبت سوف يظهر التالي :

```
racine carrée de 121 = 11.0
sinus de 0.523598775598 radians = 0.5
```

هذا المثال القصير يوضح بشكل جيدة بعض الخصائص المهمة للدالات :

- يجب كتابة إسم الدالة بجانب قوسين مثال :

  sqrt()

- داخل القوسين يمكننا كتابة برامتر واحد أو أكثر مثال :

  sqrt(121)

- تقوم الدالة بإرجاع قيمة مثال :

  11.0

. سنضع كل هذا في الصفحات القادمة . يرجى ملاحظة أن دوالات المستخدمة هنا ليست سوى مثال أولي . فإن نظرت سريعا في وثائق مكتبات البايثون سوف تجد العديد من دوالات لتنفيذ العديد من المهام , بما في ذلك خوارزميات الرياضية المعقدة (و تستخدم على نطاق واسع في الجامعات التي

تستخدم البايثون لحل المشاكل العلمية ذات مستوى عالي) . ليس هنالك شك هنا لتوفير قائمة مفسلة . مثل هذه القائمة التي يمكن الوصول إليها بسهولة في النظام بإستخدام البايثون :

Documentation HTML → Python documentation → Modules index → math

وثائق HTML – وثائق بايثون – فهرست الوحدات – math (رياضيات)

في الفصل القادم سوف نتعلم كيفية صنع دوال خاص بنا .

تمارين

> في جميع التمارين إستخدم الدالة ***input()*** لإدخال البيانات

6.1 أكتب برنامج لتحويل ميل/ساعة إلى المتر/ الثانية و إلى كم/ثانية (لا تنسى أن 1 ميل : 1609 متر) .

6.2 أكتب برنامج الذي يقوم بحساب محيط و مساحة أي مثلث . (سيدخل المستخدم أضلاع المثلث الثلاثة) .

(تذكير يتم حساب مساحة أي مثلث بإستخدام هذه الصيغة :

$S= \sqrt{d\cdot(d-a)\cdot(d-b)\cdot(d-c)}$

حيث d هو طول نصف المحيط و a , b , c هي أطراف الثلاثة)

6.3 أكتب برنامج يقوم بحساب فترة من بندول بسيط لمدة معينة

الصيغة لحساب فترة البندول البسيط هو : $T=2\pi\sqrt{\frac{l}{g}}$

L تمثل قيمة البندول و G تمثل قيمة التسارع الناتج عن الجاذبية بدل الخبرة .

6.4 أكتب برنامج الذي يسمح لك بوضع القيم في القائمة . البرنامج يجب أن يحتوي على حلقة ,و يتم طلب من المستخدم الحصول على القيم و عند إنتهاءه يضغط على زر

الإدخال دون أن يكتب شيئا و يتم إنهاء البرنامج . مع إنهاء يتم عرض القائمة, مثال على العملية :

```
Veuillez entrer une valeur : 25
Veuillez entrer une valeur : 18
Veuillez entrer une valeur : 6284
Veuillez entrer une valeur :
[25, 18, 6284]
```

الإسترخاء قليلا مع وحدة turtle

كما رأينا سابقا, واحدة من أهم مميزات البايثون هو أنه من السهل للغاية إضافة العديد من الدالات عن طريق إستيراد الوحدات .

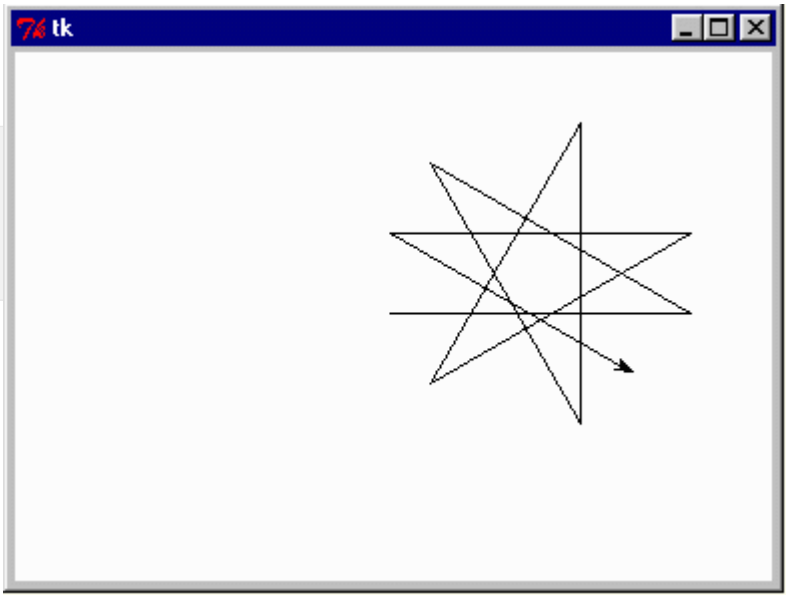
لتوضيح هذا, و للحصول على بعض المتعة مع الكائنات الأخرى (بدل الأرقام) , سوف نستكشف وحدة بايثون التي تسمح ب " رسومات السلحفات " و هذا يعني, رسوم هندسية مناظرة إلى المسار التي خلفها "سلحفاة" صغيرة ظاهرية , التي سنراقب تحركاتها على شاشة الحاسوب بإستخدام تعليمات بسيطة .

بتفعيل هذه الساحفاة فهي لعبة حقيقية للأطفال . بدل من إعطاء تفسيرات طويلة , نحن ندعوك لمحاولتها على الفور :

```
>>> from turtle import *
>>> forward(120)
>>> left(90)
>>> color('red')
>>> forward(80)
```

التمرين سيصبح أكثر غنى
إذا تم إستخدام الحلقات

```
>>> reset()
>>> a = 0
>>> while a <12:
        a = a +1
        forward(150)
        left(150)
```



تحذير : قبل أن تبدأ تشغيل هذا السكريبت, تأكد دائما من أنه لا يتضمن حلقة بدون إنهائها (راجع الصفحة 29), لإنه إذا كان به حلقة غير منتهية قد تكون غير قادر على إستعادة السيطرة على العمليات(و خاصة على نظام تشغيل ويندوز) .

إستمتع بكتابة السكريبتات التي تجعل من الرسوم التالية علامة لتقدمك . و الدالات الرئيسية المتاحة لك في وحدة turtle هي :

reset()	إزالة كل شيئ و البدء من جديد
goto(x, y)	الذهاب إلى مكان الإحداثية إكس, واي
forward(distance)	التقدم إلى الأمام لمسافة معينة
backward(distance)	الرجوع إلى الخلف
up()	رفع القلم (المضي قدما دون الرسم)
down()	إخفض قلم الرصاص (لبدء الرسم)
color(couleur)	لون القناة محدد مسبقا
left(angle)	الإتجاه يسارا بزاوية معينة (بالدرجة)
right(angle)	إتجه إلى اليمين
width(épaisseur)	حدد سمك الخط

fill(1)	تعبئة محيط مغلق بإستخدام لون محدد
write(texte)	يجب على النص أن يكون سلسلة نصية

تعبير حقيقي\مزيف

عندما يحتوي البرنامج على عبارات معينة مثل **while** و **if**, يجب على الحاسوب الذي يشغل البرنامج فحص صحة الشرط , فهو يتأكد إذا كان شرط التعبير صحيح أو خاطئ . على سبيل المثال : الحلقة التي تبدأ ب **c < 20** : ستبقى تعمل مادام **c** أصغر من **20** .

لكن كيف يمكن لجهاز الحاسوب تحديد ما إذا كان الشرط صحيحا أو خاطئ ؟

في الواقع أنت تعرف مسبقا أن الحاسوب يعالج الأرقام بدقة . يجب أولا على الحاسوب أن يحول المعلومات إلى قيم رقمية . و هذا ينطبق على مفهوم الصح /خطأ . في البايثون , كما هو الحال في السي و العديد من لغات البرمجة الأخرى , , يعتبر الحاسوب أي قيمة رقمية أخرى غير الصفر هي "صحيحة" . فقط الصفر هو "الخطأ" , على سبيل المثال :

```
ch = input('Entrez un nombre entier quelconque')
n =int(ch)
if n:
    print("vrai")
else:
    print("faux")
```

السكريبت الصغير في الأعلى سيظهر خطأ إذا أدخلت أي قيمة بخلاف الصفر , و إذا أدخلت قيمة أخرى ستحصل على واحد .

الوسائل المذكورة أعلاه يجب أن يتم فحص التعبير, مثل **a > 5** , أولا سيحول الحاسوب هذه العبارة إلى قيمة رقمية (1 إذا كان التعبير صحيح , و صفر إذا كان التعبير خاطئ) . هذا ليس واضح كثيرا, و ذلك لأن مفسر البايثون يترجم هتان القيمتين إلى **True** أو **False** .

على سبيل المثال :

```
>>> a, b = 3, 8
>>> c = (a < b)
>>> d = (a > b)
>>> c
True
>>> d
False
```

سيتم تخزين نتيجة التعبير **a < b** (صحيح) في متغير **c** . و بالمثل لنتيجة لمتغير عكسي , يتم التسجيل في المتغير **d**[16].

بإستخدام القليل من الحيل , يمكننا التحقق إذا كانت هذه القيم صحيحة أو خاطئة (هي في حقيقة الرقمين 1 و 0) .

```
>>> accord = ["non", "oui"]
>>> accord[d]
non
>>> accord[c]
oui
```

بإستخدام المتغيرين **c** و **d** مؤشرات لإسترداد العناصر من قائمة أكورد نحن نتأكد أن خطأ = 0 و صحيح = 1 .

السكريبت الصغير التالي يشبه كثيرا السكريبت السابق . فهو يسمح لنا بإختبار الحرف صحيح أو خطأ لسلسة نصية :

```
ch = input("Entrez une chaîne de caractères quelconque")
if ch:
    print("vrai")
else:
    print("faux")
```

سوف تحصل على "خطأ " لكل سلسلة فارغة , و "صحيح" لكن سلسلة تحتوي على الأقل على حرف . فهل تستطيع أن تختبر بنفس الطريقة إذا كانت السلسلة فارغة تظهر "خطأ" و إذا كانت تحتوي على أي شيئ تكون "صحيحة".[17].

العبارة **if ch :** , في السطر الثالي من هذا المثال ,هو ما يعادل **If ch =!' ' :** من وجهة نظرنا كبشر , أما بالنسبة للحاسوب فهي ليست كذلك , فالعبارة **if ch :** هي للتحقق مباشر من أن قيمة المتغير **ch** هو متغير فارغة أو لا , كما نراه نحن, اما في العبارة **If ch =!' ' :** يتطلب البدء في مقارنة محتوى **ch** بقيمة المقدمة التي وضعناها في برنامجنا (سلسلة فارغة), ثم إختبار نتيجة هذه

---

[16] هذه المتغيرات من نوع صحيح خاص : نوع "منطقي - Boolean" . المتغيرات من هذا النوع لا يمكن أن تأخذ سوى قيمتين **True** و **False** (في الواقع, 1 و 0) .

[17] هياكل البيانات الأخرى تتصرف بطريقة متشابهة . سوف تدرس الأنفاق و القواميس في الفصل 10, و التي ستكون "خطأ" في حالة أنها فارغة, و صحيحة إذا كان لديها محتوى .

المقارنة صحيحة أو خاطئة (أو بعبارة أخرى, يتأكد إذا كانت النتيجة صحيحة أو خاطئة) . لذا فهو يتطلب عمليتين متتاليتين, العبارة الأولى هي الأكثر كفائة .

للأسباب نفسها, في هذا السكريبت :

```
ch =input("Veuillez entrer un nombre : ")
n =int(ch)
if n % 2:
     print("Il s'agit d'un nombre impair.")
else:
     print("Il s'agit d'un nombre ir.")
```

و هو أكثر فاعلية ما فعلناه لبرمجة السطر الثالث , كما فعلنا في الأعلى, أو بالأحرى كتابة هذا بشكل واضح **if n % 2 != 0** , لأن هذه الصيغة تتطلب من جهاز الحاسوب أداء عمليتي مقارنة متتالية بدل من واحدة .

هذا التعليل "قريب من الحاسوب", ربما سيبدو لك خفي في البداية, و لكن نعتقد أن هذا الشكل من الكتابة سوف تتعلمه بسرعة .

مراجعة

في ما يلي, لن نتعلم مفاهيم جديدة, و لكن مجرد إستخدام كل ما تعلمناه سابقا لصنع برامج صغيرة حقيقية .

التحكم في تلقيم التنفيذ - بإستخدام قائمة بسيطة

دعونا نبدأ مع عودة صغيرة لفروع الجمل الشرطية (ربما هذه مجموعة التعليمات الأكثر الأهمية في أي لغة):

```
#إستخدام قائمة مع شروط متفرعة

print("Ce script recherche le plus grand de trois nombres")
print("Veuillez entrer trois nombres séparés par des virgules : ")
ch =input()
#ملاحظة : إن ربط الدالتين list() و eval() يسمح بتحويل
# في القائمة جميع سلاسل مفصولة بفواصل: [18]
```

[18] في الحقيقة, إن دالة **eval()** تقوم بفحص محتوى السلسلة التي تم توفيرها على شكل برامتر كتعبير بايثون الذي يجب أن يتم إرجاع نتيجته, على سبيل المثال : **eval("7 + 5")** يجب أن يتم إرجاع العدد الصحيح **12** . فإذا قمت بتوفير سلسلة من القيم مفصولة بفواصل, و هذه تتوافق مع النفق . الأنفاق هي متسلسلات متصلة بقوائم . و سيتم شرحها في الفصل العاشر (أنر إلى صفحة)

```
nn = list(eval(ch))
max, index = nn[0], 'premier'
if nn[1] > max:                          # لا تنسى النقطتين !
    max = nn[1]
    index = 'second'
if nn[2] > max:
    max = nn[2]
    index = 'troisième'
print("Le plus grand de ces nombres est", max)
print("Ce nombre est le", index, "de votre liste.")
```

في هذا التمرين, نجد مرة أخرى مفهوم "تعليمة الكتلة" , التي بدأنها بالفعل في الفصلين 3 و 4 , و التي كان يجب عليك إستيعابها (للتذكير, كتل التعليمات محددة بمسافة) . بعد أول عبارة **if**, على سبيل المثال , هنالك نوعان من بادئة أسطر تحديد كتلة البيانات . سيتم تنفيذ هذه البيانات إذا كنا الشرط **nn[1] > max** صحيح .

السطر التالي (إستعمال عبارة **if** ثانية) . لم يبدأ ببادئة , و لذلك فإن هذا السطر يتم تعريفه كجزء من جسم البرنامج. فيتم تنفيذ هذا الجزء دائما , في حين أن السطرين التاليين (و التي يتم تنفيذها حتى الأن ككتلة) لن يتم تنفيذه إلا إذا كان الشرط **nn[2] > max** صحيحا .

نتقدم بنفس المنطق, و نرى أن على السطرين الأخيرين هما جزء من الكتلة الرئيسية و التي يتم تنفيذها دائما .

حلقة وايل - التداخل

نواصل السير في نفس المسار عن طريق دمج هياكل أخرى :

```
1#     # <while> - <if> - <elif> - <else> تعليمة مركبة
2#
3#     print('Choisissez un nombre de 1 à 3 (ou 0 pour terminer) ', end=' ')
4#     ch = input()
5#     a = int(ch)              # تحويل السلسلة المدخلة إلى عدد صحيح
6#     while a:                 # < while a != 0: > :تعادل
7#         if a == 1:
8#             print("Vous avez choisi un :")
9#             print("le premier, l'unique, l'unité ..."
10#        elif a == 2:
11#            print("Vous préférez le deux :")
12#            print("la paire, le couple, le duo ...")
13#        elif a == 3:
14#            print("Vous optez pour le plus grand des trois :")
15#            print("le trio, la trinité, le triplet ...")
16#        else :
17#            print("Un nombre entre UN et TROIS, s.v.p.")
18#        print('Choisissez un nombre de 1 à 3 (ou 0 pour terminer) ', end =' ')
19#        a = int(input())
```

```
20#   print("Vous avez entré zéro :")
21#   print("L'exercice est donc terminé.")
```

نجد هنا أن حلقة **while**, مرتبط لمجموعة عبارة **if** و **elif** و **else** . لاحظ مرة أخرى بنية المنطقية للبرنامج المصنوع بمساعدة التبويتات (... لا تنسى الرمز ":" في نهاية السطر)

في السطر السادس, يتم إستخدام عبارة **while** كما هو موضح في الصفحة 55 : لتفهمها فقط تذكر أن كل القيم الرقمية غير الصفر تعتبر صحيحة من قبل مفسر البايثون .تستطيع تغيير هذا الشكل من الكتابة ب **while a != 0** إذا كنت تفضل هذا (لنتشكر هنا أن المعامل المقارنة != "تختلف عن "), لكن أقل فاعلية .

هذه "حلقة while" تحفز بإستجواب بعد كل إجابة من المستخدم (على الأقل حتى قرر الخروج و لم يدخل أي قيمة)

في داخل الحلقة, نجد مجموعة من العبارات **if** و **elif** و **else**(من السطر 7 إلى السطر 17), التي توجه تدفق البرنامج إلى أماكن مختلفة , ثم تعليمة **print()** و التعليمة **input()** (السطر 18 و 19) يتم تشغيله في جميع الحالات : يرجى ملاحظة مستوى مسافة البادئة, و الذي هو نفس كتلة , **if** **elif** و **else** . بعد هذه التعليمات, تستأنف حلقة البرنامج تنفيذها مع العبارة **while** (السطر 6)

في السطر 19, إستخدامنا تركيبة الكتابة لكتابة الكود أكثر إيجازا, و هو ما يعادل السطرين 4 و 5 مجتمعة .

يتم تنفيذ عبارتي **print()** الأخيرتين (السطر 20 و 21) اللتان ستعملان عند الخروج من الحلقة .

تمارين

6.5 ما يفعل هذا البرنامج (في الأسفل), في هذه الحالات الأربعة : إذا كان **a** يساوي 1 أو 2 أو 3 أو 15 ؟

```
if a !=2:
    print('perdu')
elif a ==3:
    print('un instant, s.v.p.')
else :
    print('gagné')
```

6.6 ماذا تفعل هذه البرامج ؟

```
a)   a = 5
```

```
        b = 2
        if (a==5) & (b<2):
            print('"&" signifie "et"; on peut aussi utiliser\
                  le mot "and"')
b)      a, b = 2, 4
        if (a==4) or (b!=4):
            print('gagné')
        elif (a==4) or (b==4):
            print('presque gagné')
c)      a = 1
        if not a:
            print('gagné')
        elif a:
            print('perdu')
```

6.7 جرب البرنامج **c)** مع a = 0 بدل من a =1 . ماذا حدث ؟ هل دخل !

6.8 أكتب برنامج الذي يقوم بتحديد عددين صحيحين a و b, ثم يضيف رقمي الضرب ل 3 و 5 بين هذين الحدين . على سبيل المثال a = 0 و b = 32 و يجب أن يكون الناتج 0 + 15 + 30 = 45 .

قم بتعديل طفيف على البرنامج لإضافة أرقام ضرب 3 أو 5 بين حدي a و b . و بحدي 0 و 32 يكون الناتج : 0 + 3 + 5 + 6 + 9 + 10 + 12 + 15 + 18 + 20 + 21 + 24 + 25 + 27 + 30 = 225.

6.9 أكتب برنامج الذي يحدد إذا كانت السنة كبيسة أو لا , و السنة الكبيسة هي السنة التي تقبل القسمة على 4 . و لن تكون كبيسة إذا كان A من مضعفات 100 (على الأقل A ليس من مضعفات رقم 400) .

6.10 أكتب البرنامج الذي يطلب من المستخدم إسمه و جنسه (ذكر أو أنثى) و بناءا على هذه البيانات, البرنامج سيعرض " عزيز السيد " أو عزيزيتي السيدة " متبوعة بإسم الشخص .

6.11 أطلب من المستخدم إدخال 3 أطول (a و b و c) . حدد ما إذا كان من الممكن إنشاء مثلث بمساعدة هذه الأطوال الثلاثة . ثم حدد ما إذا كان هذا المثلث قائمة الزاوية أو متساوي الضلعين أو متساوي الأضلاع أو إلخ . إنتبه : يمكن أن يكون المثلث متساوي الضلعين .

6.12 أكتب برنامج الذي يطلب من المستخدم إدخال رقم : ثم ثم يعرض له الجذر التربيعي لهذا العدد , فإذا لا يوجد جذر تربيعي لذلك الرقم تظهر له رسالة تخبره بذلك .

6.13 أكتب برنامج الذي يحول علامة المدرسة أن, التي أدخلها المستخدم بشكل نقاط (على سبيل المثال 27 من 85) ,إلى درجة قياسية مثل التالي :

```
Note                Appréciation
N >= 80 %                 A
80 % > N >= 60 %          B
```

```
60 % > N >= 50 %          C
50 % > N >= 40 %          D
N < 40 %                  E
```

6.14 أنظر في القائمة التالية

```
['Jean-Michel', 'Marc', 'Vanessa', 'Anne', 'Maximilien',
['Alexandre-Benoît', 'Louise'
```

أكتب برنامج الذي يعرض كل هذه الأسماء مع عدد الحروف التي تتكون منها

6.15 أكتب حلقة برنامج التي تطلب من المستخدم إدخال نتائج الطلاب . الحلقة لا تتوقف إلا عندما يدخل المستخدم قيمة سالبة . مع النتائج التي تم إدخالها , يتم وضعها في قائمة . بعد كل دخول نتيجة (بالتالي كل تكرار للحلقة), يظهر البرنامج عدد النتائج التي تم إدخالها, الدرجة الأكثر تقدير و الدرجة الأقل , و معدل جميع النتائج .

6.16 أكتب سكريبت الذي يظهر قيمة قوة الجاذبية التي تعمل بين كتلتين 10 000 كلغم, لمسافة التي التي تزيدهندسيا عن 2 , بداية من 5 سم (0,05 متر) .

تخضع قوة الجاذبية لهذه الصيغة : $F = 6{,}67 . 10^{-11} . \frac{m \cdot m'}{d^2}$

مثال

```
d = .05 m :  la force vaut  2.668 N
d = .1 m  :  la force vaut  0.667 N
d = .2 m  :  la force vaut  0.167 N
d = .4 m  :  la force vaut  0.0417 N
```

etc.

7

دالات أصلية

البرمجة هي فن تعليم الحاسوب على أداء مهام التي لم يكن قادر على أدائها سابقا . أحد الأساليب الأكثر إثارة للإهتمام لتحقيق هذا هو إضافة تعليمات جديدة للغة البرمجة التي تستخدمها, في شكل دالة أصلية .

تعريف دالة

السكريبتات التي كتبتها حتى الأن قصيرة للغاية , و ذلك لأن هدفهم هو تعلم أساسيات اللغة , بمجرد أن تبدأ بتطوير مشاريع حقيقية, سوف تواجه الكثير من المشاكل المعقدة, و تبدأ أسطر البرنامج بالتراكم ...

الطريقة الفعالة لحل الكثير من المشاكل هي تقسيم المشاكل إلى مجموعة مشاكل صغيرة أكثر بساطة لدراسة كل واحدة على حدة (يمكن لهذه المشاكل الصغيرة أن تحلل نفسها بنفسها بدورها) و من المهم أن يتم تقسيم هذه المشاكل بشكل صحيح في خوارزميات[19] و يجب أن تكون واضحة .

و من ناحية أخرى, فإنه غالبا ما تستخدم نفس تسلسل التعليمات مرارا و تكرارا في أحد البرامج, و سيكون ليس جيدا إعادة كتابة الكود كل مرة .

الدالات[20] و الكائنات هي هياكل مختلفة من الوظائف الفرعية التي تم تخيلها من قبل المبرمجين للغات عالية المستوى لحل الصعوبات المذكورة أعلاه . سنقوم بشرح هنا للمرة الأولى تعريف الدالات في البايثون . و سوف نناقش الكائنات و الصفوف في وقت لاحق .

---

19 الخوارزمية هي سلسلة مفصلة من العمليات المطلوبة من أجل حل مشكلة .

20 و يوجد في لغات البرمجة الأخرى روتينات (و التي تسمى أيضا برامج فرعية) و الإجراءات . لا يوجد روتينات في البايثون . و بالمعنى الدقيق للكلمة الدالة (تقوم بإرجاع قيمة) و الإجراءات (لا تقوم بإرجاع قيمة) .

لقد إلتقيت بالفعل مع الدالات المبرمجة سابقا لأداء مهام مختلفة . سوف نتعلم الأن كيف صنع دالات بأنفسنا .

تركيب الجملة في البايثون لتعريف الدالة هو :

```
def nomDeLaFonction(liste de paramètres):
    ...
    bloc d'instructions
    ...
```

- يمكنك إختيار أي إسم للدالة التي تقوم بإنشاءها, مع إستثناء الكلمات المحجوزة للغة[21], و يجل عليك أن لا تستعمل أي رموز (بإستثناء هذه _) كما هو الحال لأسماء المتغيرات, و أنصح بإستخدام الأحرف الصغيرة في معظم الأوقات, و في بداية إسم المتغير (الأسماء التي تبدأ بحرف كبير محجوز للصفوف و التي سوف نتحدث عنها لاحقا) .
- مثل العبارات **if** و **while** التي قد عرفتها سابقا, و العبارة **def** هي عبارة مجمع . السطر الذي يحتوي على هذه العبارة ينتهي بالضرورة بنقطتين, و الذي يحتوي على مجموعة من التعليمات التي يجب أن لا تنسهى فيها مسافة البادئة .
- قائمة البرامترات هي المعلومات التي تريد إعطائها للدالة لإستعمالها (القوسين قد يكونا فارغين إذا كانت الدالة لا تحتاج إلى بارمترات)
- يتم إستخدام الدالة مثل أي تعليمة تقريبا . في قلب البرنامج . ستم إستدعاء الدالة عن طريق إسمها يليها قوسين . و إذا كان ضروريا, يتم وضع البارمترات التي تريد أن تحيلها إلى الدالة . و يجب عادة إدخال بارامتر واحد في تعريف الدالة, على الرغم من أنه يمكنك أن تحدد القيم الإفتراضية لهذه البرامترات(سنتعرف إلى هذا لاحقا)

دالة بسيطة بدون برامترات

لأول برنامج ملموس لنا بالدوال , سوف نستعمل مرة أخرى البايثون في الوضع التبادلي . الوضع التبادلي للبايثون هو في الواقع المثالي إجراء إختبارات صغيرة مثل التالية . هذه الميزة لا تجدها في جميع لغات البرمجة !

```
>>> def table7():
...     n = 1
...     while n <11 :
...         print(n * 7, end =' ')
...         n = n +1
...
```

21تجد قائمة الكلمات المحجوزة في صفحة .***

عن طريق إدخال هذه الأسطر القليلة, عرفنا وظيفة بسيطة جدا و هي تحسب و تظهر أول عشرة نتائج لجدول الضرب على سبعة . لاحظ جيدا الأقواس,[22]النقتطين, و العبارة و مسافة البادئة (هي الكتلة التي تشكل جسم الدوال نفسه) .

لإستخدام دوال محدد الذي صنعناه, يمكن دعوته بواسطة إسمه :

```
>>> table7()
```

الذي يؤدي إلى عرض :

```
7 14 21 28 35 42 49 56 63 70
```

يمكننا الأن إعادة إستخدام هذه الدالة مرارا و تكرارا, مرات عديدة كما نرغب . يمكننا أيضا دمجها في تعرف وظيفة أخرى , كما في المثال التالي :

```
>>> def table7triple():
...     print('La table par 7 en triple exemplaire :')
...     table7()
...     table7()
...     table7()
...
```

يمكننا إستخدام هذه الميزة الجديدة عن طريق إدخال الأمر :

```
>>> table7triple()
```

عرض الناتج سيكون كالتالي :

```
La table par 7 en triple exemplaire :
7 14 21 28 35 42 49 56 63 70
7 14 21 28 35 42 49 56 63 70
7 14 21 28 35 42 49 56 63 70
```

و يمكن للدالة الثانية إستدعاء الأولى و نفس الشيئ مع الدالة الثالثة و إلخ ... في هذه المرحلة التي وصلنا إليها, قد لا تعرف بعد ما فائدة هذه و لكن يمكن ملاحظة إثنين من الفوائد :

22إسم الدالة يجب أن يكون دائما مصحوبا بقوسين, حتى لو كانت الدالة لا تأخذ أي برامترات . و هذه هي نتيجة إتفاقية كتابة التي تنص على أن في أي نص يتعامل مع برمجة الحاسوب, يجب أن يصاحب إسم الدالة قوسين فارغين . و نحن نلتزم بهذه الإتفاقية في النص التالي .

- إنشاء دالة جديدة تسمح لنا بإعطاء إسم لمجموعة من التعليمات . و بهذه الطريقة, يمكنك تبسيط جسم الرئيسي للبرنامج, عن طريق إخفاء خوارزمية ثانوية معقدة تحت أمر واحد, و التي يمكن إعطائها إسم واضح جدا, بالفرنسية إذا أردت .
- صنع دالة جديد يمكننا من إنشاء تصغير حجم البرنامج, من خلال إزالة الأجزاء المتكررة . على سبيل المثال, إذا أردت إظهار جدول سبعة مرات في نفس البرنامج, يمكنك أن لا تكتب كل مرة الخوارزمية التي تقوم بهذا العمل .

الدالة هي تعليمات جديدة خاصة بك, التي تستطيع إضافتها بحرية إلى لغة البرمجة الخاصة بك .

دالة مع بارمترات

في الأمثلة السابقة, لقد عرفنا و إستعملنا دالة التي تظهر جدول الضرب على 7 . الأن نفترض أنه يجب علينا أن نفعل نفس الشيئ مع جدول 9 . هل يجب علينا أن نصنع دوال جديد لهاذا ؟ و لنفترض أيضا أننا أردنا بعد ضلك أن نفعل نفس الشيئ مع جدول 13 و هل يجب علينا البدء من جديد . أليس من الأفضل أن نعرف دالة قادرة على عرض أي جدول, على الطلب ؟

عندما نسمي هذه الدالة, يجب أن نكون طبعا قادرين على الإشارة إلى أي جدول نريد عرضه . هذه المعلومات يجب تمريرها للدالة و التي تسمى بارمتر . لقد إلتقينا عدة مرات مع دالات متكاملة التي تأخذ بارمتر . الدالة **sin(a)** على سبيل المثال , يحسب جوف الزاوية **a** . الدالة **sin()** إستعملت إذا قيمة عددية كبرامتر للقيام بالعمل .

في تعريف الدالة, يجب أن يكون هنالك متغير خاص لتلقي البارامتر . هذا المتغير يسمى بارمتر . نختار له إسم لبناء قواعد التعليمة كالعادة, و نضع الإسم في ما بين قوسين المصاحبة لتعريف الدالة .

هذا الذي يجب كتابته في حالتنا :

```
>>> def table(base):
...     n = 1
...     while n <11 :
...        print(n * base, end =' ')
...        n = n +1
```

الدالة **table()** كما هو معرف أعلاه تستعمل البرامتر **base** لحساب أول عشرة نتائج لجدو الضرب الموافق .

لأختبار هذه الميزة الجديدة, يجب علينا إستدعائها مع البارامتر . مثال على ذلك :

```
>>> table(13)
13 26 39 52 65 78 91 104 117 130

>>> table(9)
9 18 27 36 45 54 63 72 81 90
```

في هذه الأمثلة, يتم تلقائيا تعيين قيمة بين قوسين عندما نستدعي الدالة(و بالتالي فهو بارامتر) إلى البرامتر القاعدة .

في جسم الدالة, **base** تلعب دورا ليس كأي متغير أخر . عندما ندخل الأمر **table(9)** , نحن هنا نقصد الألة التي تنفذ الدالة **table()** عن طريق إعداد القاعدة لمتغير 9 للمتغير **base** .

إستعمال المتغير كبرامتر

في المثالين أعلاه, البرامتر الذي إستخدمناه للدالة **table()** هو في كل مرة عدد ثابت (المتغير 13 ثم المتغير 9) . هذا ليس إلزاميا . لكن البرامتر الذي إستخدمناه عند إستدعاء الدالة يمكن أن يكون متغير أيضا, مثل المثال أدناه . حلل المثال جيدا, حاول تشغيله, و صف في كراس التمارين الخاص بك ما يحدث . موضحا بشكل جيد . هذا المثال بعطيك دراسة أولية لإستعمال الدالات للقيام ببساطة مهام معقدة :

```
>>> a = 1
>>> while a <20:
...     table(a)
...     a = a +1
...
```

ملاحظة مهمة

في المثال أعلاه, البرامتر الذي مررناه ل **table()** هو القيمة التي يحتويه المتغير a . داخل الدالة, هذا البرامتر يتم تعيينه لبرماتر القاعدة, الذي هو متغير أخر . إذا لاحظ الان أن :

> إسم المتغير الذي تم تمريره كبرامتر ليس له أي علاقة بإسم البرامتر المقابل في الدالة .

قد تكون هذه الأسماء هي نفسها إذا أردت, لكن يجب أن تعرف أنهم لا يعبرون على نفس الشيئ (على الرغم من أنهم قدي يحتويان على نفس القيمة الإختيارية) .

تمرين

7.1 إستدعي وحدة **turtle** لأداء رسوم بسيطة . يمكنك رسم مجموعة من المثلثات متساوية الأضلع مختلفة الألوان . و للقيام بذلك, يجب عليك تعريف متغير **triangle()** الذي يقوم برسم مثلث بلون محدد (و هذا يعني أن تعريف الدالة يجب أن يحتوي على برامتر لأخذ اللون) . ثم إستعمل هذه الدالة لرسم المثلث نفسه في أماكن مختلفة, مع تغيير اللون كل مرة .

دالة مع عدة برامترات

الدالة **table()** هي مثيرة للإهتمام بالتأكيد, لكنها لا تظهر سوى أول عشرة نتائج لجدول الضرب, ربما نحن نرغب أن نظهر أكثر من ذلك . الأن, سوف نحسن الكود بإضافة برامترات أخرى في النسخة الجديدة التي سنطلق عليها هذه المرة **tableMulti()** :

```
>>> def tableMulti(base, debut, fin):
...     print('Fragment de la table de multiplication par', base, ':')
...     n = debut
...     while n <= fin :
...        print(n, 'x', base, '=', n * base)
...        n = n +1
```

هذه الميزة الجديدة سوف تستخدم ثلاثة برامترات : الأولى هي قاعدة الجدول مثل المثال السابق, الثانية هي أول عدد ضرب يبدأ به , و الثالث هي أخر عدد ضرب ينتهي به .

جرب هذه الدالة عن طرق إدخال هذا على سبيل المثال :

```
>>> tableMulti(8, 13, 17)
```

سوف يظهر :

```
Fragment de la table de multiplication par 8 :
13 x 8 = 104
14 x 8 = 112
15 x 8 = 120
16 x 8 = 128
17 x 8 = 136
```

ملاحظات

- لتحديد دالة مع عدة برامترات , يجب عليك كتابتهم في ما بين القوسين التي تلي إسم الدالة, مفصولة بفواصل .

- عند إستدعاء الدالة, يجب أن يكون البرامترات في نفس أماكنهم الصحيحة (و يتم فصلهم أياضا بفاصلة) . سيتم تعيين البرامتر الأول للبرامتر الاول للدالة , و البرامتر الثاني للبرامتر الثاني في الدالة و إلخ ...
- كتمرين, حاول تسلسل التعليمات التالية و صف في كراس التمارين النتيجة :

```
>>> t, d, f = 11, 5, 10
>>> while t<21:
...     tableMulti(t,d,f)
...     t, d, f = t +1, d +3, f +5
...
```

متغير محلي, متغير عام

عندما نحدد متغيرات داخل جسم الدالة, هذه المتغيرات هي فقط للوصول إلى الدالة نفسها . نقول أن هذه المتغيرات محلية للدالة . هذا على سبيل المثال حالة من المتغيرات base, debut, fin و n للتمرين السابق .

في كل مرة يتم إستدعاء فيها **tableMulti()** البايثون تحجز لها (في ذاكرة الحاسوب) مساحة جديدة للإسم[23]. محتويات المتغيرات base و debut و fin و n يتم تخزينها في مساحة الإسم التي لا يمكن الوصول إليها من خاج الدالة . على سبيل المثال, إذا أردت إظهار محوى الدالة base بعد القيام بالتمرين السابق, سوف تحصل على رسالة الخطأ التالية :

```
>>> print(base)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'base' is not defined
```

23 هذا المفهوم لمساحة الأسماء سوف نتعمق به تدريجيا . و سوف نتعلم أيضا في وقت لاحق أن الدالات هي في الواقع كائنات التي يتم إنشاء في كل مرة مثيل عندما يتم إستدعائها .

الالة تقول لنا بوضوح أن الرمز base غير معروف, في حين أنه تم طباعته من قبل الدالة **tableMulti()** نفسها . مساحة الأسماء التي يحتويها الرمز base مقتصرة فقط للدالة **tableMulti()**, و يتم حذف أوتوماتكيا عندما تنتهي الدالة من عملها .

المتغيرات المعرفة خارج الدالة هي متغيرات عامة . محتواه " مرئي " للدالة, لكن الدالة لا تستطيع تغيير محتواها . على سبيل المثال :

```
>>> def mask():
...     p = 20
...     print(p, q)
...
>>> p, q = 15, 38
>>> mask()
20 38
>>> print(p, q)
15 38
```

حلل بعناية هذا المثال :

سوف نبدأ من خلال تعريف دالة بسيطة جدا (و التي لا تستعمل أي برامترات) . في هذه الدالة, يتم تعريف الدالة p مع القيمة الأولية لهذا المتغير هي 20 . هذا المتغير سوف يكون داخل الدالة أي محلي .

بمجرد الإنتهاء من تعريف الدالة, نعود للمستوى الأساسي و نعرف متغيرين p و q الذان يحتويان على القيمتين 15 و 38 . هذان المتغيران يتم تعريفهما في المستوى الأساسي سيكونا إذا متغيرات عامة .

و هكذا تم إستخدما نفس المتغير p مرتين, لتحديد إثنين من المتغيرات المختلفة : واحد عام و الثاني محلي . يمكن النظر إلى هذان المتغيران على أنهما متغيران مستقلان و منفصلان عن بعضهما البعض, حسب القاعدة أن في الدالة(حيث تكون المنافسة), يكون للمتغيرات المحلية أولوية .

في الواقع يبدوا أنه عندما يتم تشغيل دالة **mask()** , المتغيران العامان q و y يبدو أنه يمكن الوصول إليهما, منذ يتم طباعتهم بشكل جيد, ل p على العكس, القيمة المحلية هي التي يتم إظهارها .

قد يعتقد المرء في البداية أن دالة **mask()** ببساطة تغير محتوى المتغير العام p (بما أنه يمكن الوصول إله) . الأسطر التالية تظهر أنه ليس كذلك : عند الخروج من الدالة **mask()** , يعود المتغير العام p إلى قيمته الأولية .

يبدوا هذا كله معقد في البداية, لكن سرعان ما سوف نعرف كم هو مفيد تعريف المتغيرات كمحلية, و هذا معنى بطريقة أخرى أنها تقتصر فقط على الدالة . هذا يعني أنك تستطيع دائما إستخدام عدد لا نهائي من الدالات دون الحاجة إلى القلق من أسمائها إذا كانت مستخدمة سابقا أو لا : هذه المتغيرات لا تستطيع أبدا أن تتداخل مع تلك التي عرفتها بنفسك في مكان أخر .

إذا أردت تستطيع تغيير هذا . يمكن أن يكون على سبيل المثال أنك عرفت دالة التي يجب عليها تغيير محتوى متغير عام . لفعل هذا, ببساطة إستعمل التعليمة global . هذه التعليمة تمكنك من الإشارة داخل التعريف الدالة - تمكنك من لتعامل مع المتغيرات بشكل شامل .

في المثال أدناه, يستخدم المتغير a داخل الدالة ()monter ليس فقط للوصول, بل حتى تغيير محتواها, لأنه تم تعريفه على أنه متغير بإستعمال شامل . و على سبيل المقارنة, إعمل نفس التمرين لكن هذه المرة إخذف التعليمة global : المتغير لا يزداد عن كل إستدعاء للدالة .

```
>>> def monter():
...     global a
...     a = a+1
...     print(a)
...
>>> a = 15
>>> monter()
16
>>> monter()
17
>>>
```

الدالات الحقيقية والإجراءات

للمتمرسين, الدالات التي وصفناها هنا هي في المعنى الدقيق للكلمة ليست كذلك, و لكن بشكل أكثر دقة, الدالة "الحقيقية" (بالمعنى الدقيق) يجب عليها أن تعود قيمة واحدة عندما تنتهي . الدالة "الحقيقية" تستطيع إستعمال علامة المساوات في التعبيرات مثل **y = sin(a)** . نفهم من هذا أن هذه العبارة, الدالة **()sin** تعيد قيمة(في داخل البرامتر a) الذي تم تعيينه مباشرة إلى المتغير y .

دعونا نبدأ مع مثال بسيط للغاية :

```
>>> def cube(w):
...    return w*w*w
...
```

العبارة return تحدد القيمة التي يجب إرجاعها من الدالة . في هذه الحالة, هذا هو مكعب البرامتر الذي تم تمريره عند إستدعاء الوظيفة . على سبيل المثال :

```
>>> b = cube(9)
>>> print(b)
729
```

على سبيل المثال أكثر قليلا تعقيدا, سوف نقوم الأن بتغيير صغير على دالة **table()** الذي عملنا به عمل لا بأس به, سوف نجعله يعود بقيمة . هذه القيمة في هذه الحالة هي قائمة (قائمة متكونة من أول عشرة نتائج لجدول الضرب المختار . هذه فرصة جيدة لإعادة الحديث عن القوائم . في هذه العملية, يجب علينا أن ننتهز هذه الفرصة لتعلم شيئا جديد .

```
>>> def table(base):
...     resultat = []              # النتيجة الأولى هي قائمة فارغة
...     n = 1
...     while n < 11:
...         b = n * base
...         resultat.append(b)     # إضافة قيمة إلى القائمة
...         n = n +1               # (أنظر إلى الشرح أدناه)
...     return resultat
...
```

لتجربة هذه الدالة, نستطيع أن ندخل على سبيل المثال :

```
>>> ta9 = table(9)
```

و بالتالي نحن خصصنا للمتغير **ta9** أول عشرة نتائج لجدول الضرب على 9, في شكل لائحة :

```
>>> print(ta9)
[9, 18, 27, 36, 45, 54, 63, 72, 81, 90]
>>> print(ta9[0])
9
>>> print(ta9[3])
36
>>> print(ta9[2:5])
[27, 36, 45]
>>>
```

(تذكر: العنصر الأول في الدالة هو المؤشر 0)

ملاحظات

- كما رأينا في المقال السابق, العبارة return تحدد ما هي القيمة التي يجب أن تعود من الدالة . في هذه الحالة, هذا هو هنا محتوى المتغير resultat , هذا معناه قائمة الأرقام التي تم صناعتها من قبل الدالة[24].
- العبارة **resultat.append(b)** هي المثال الثاني لنا لإستخدام مفهوم الإستدعاء الذي لا يزال الكثير من الأشياء منه لم نضعها : هذه العبارة, نحن نطبق طريقة **append()** للكائن resultat .
- سوف نشرح الأن خطوة بخطوة ما المقصود ببرمجة الكائن . الأن, نعترف ببساطة بأن هذا المصطلح عام جدا ينطبق بشكل خاص على قوائم البايثون . الأسلوب هو ليس أكثر من مجرد دالة (و يمكنكم أيضا معرفتها بوجود الأقواس), لكن هنا الدالة مرتبطة بكائن . و هي جزء من تعريف هذا الكائن, أو بشكل أكثر دقة فئة معينة تنتمي لهذا الكائن(سوف ندرس مفهوم الطبقة في وقت لاحق) .
- يتم تنفيذ الأسلوب المرتبط بالكائن بطريقة أو بأخرى "بتشغيل دالة" هذا الكائن بطريقة معينة . على سبيل المثال يتم تطبيق الأسلوب **methode4()** لكائن objet3 بمساعدة تعليمة النوع : **()** **objet3.methode4** , و هذا معناه إسم الكائن, ثم إسم الأسلوب, متصلة ببعضها البعض بواسطة نقطة . هذه النقطة لها دور أساسي : يمكن إعتباره معامل حقيقي .
- في مثالنا, نحن نطبق الأسلوب **append()** إلى كائن resultat, الذي هو قائمة . في البايثون, القوائم هي فئة معينة من الكائنات, التي يمكن أن تطبق بشكل فعال على مجموعة متنوعة من الأساليب . في هذه الحالة, الأسلوب **append()** مجموعة كائنات "قوائم" يستخدم لإضافة عنصر إلى النهاية . الكائن الذي سيتم إضافته سيكون داخل أقواس, مثل جميع البرامترات .
- كنا قد حصلنا على نتيجة مماثلة إذا إستخدمنابدلا من هذه العبارة التعليمة "resultat = resultat + b]]" (معامل السلسلة يعمل في الواقع أيضا مع القوائم), هذه الطريقة هي أقل كفاءة و فعالية, لأنها تقوم بإعادة تعريف قائمة جديدة عند كل تكرار جديد للحلقة . حيث القائمة الكاملة السابقة تقوم بكل مرة بإعادة النسخها مع إضافة عنصر إضافي .
- عند إستخدام أسلوب **append()**, من سلبياته, عوائد الحاسوب في الواقع على تعديل قائمة موجودة بالفعل (دون نسخه في متغير جديد). إذا هذا الأسلوب هو الأفضل, لأنه يستخدم موارد أقل, بالإضافة إلى أنه أسرع (و خاصة عند التعامل مع القوائم الكبيرة) .

24يمكن إستخدام **return** بدون أي برامتر داخل الدالة, مما يتسبب في إغلاق البرنامج مباشرة . و القيم التي يتم إرجاعها في هذه الحالة كائن **None** (كائن خاص, "لا شيئ") .

- إنه ليس ملزم علينا أن كل قيمة يتم إرجاعها من لبدلبو يجب أن تكون بواسطة المتغير (كما فعلنا حتى الأن في هذه الأمثلة). و بالتالي, يمكننا إختبار الدوالان **cube()** و **table()** عن طريق إدخال هذه الأمرين :

```
>>> print(cube(9))
>>> print(table(9))
>>> print(table(9)[3])
```

أو بشكل أكثر بساطة :

```
>>> cube(9)...
```

إستعمال الدالات داخل سكريبت

لتقدمنا في دالات, نحن إستخدمنا لحد هذه اللحظة الوضع التفاعلي للبايثون .

فمن الواضح أنه يمكننا إستخدام الدالات في البرامج النصية (سكريبت) كذلك . الرجاء حاول فعل ذلك بنفسك مع البرامج الصغيرة في الأسفل, و التي تحسب حجم الكرة بإستخدام الصيغة التي ربما قد تعرفها :

$$V=\frac{4}{3}\pi R^3$$

```
def cube(n):
   return n**3

def volumeSphere(r):
   return 4 * 3.1416 * cube(r) / 3

r = input('Entrez la valeur du rayon : ')
print('Le volume de cette sphère vaut', volumeSphere(float(r)))
```

ملاحظات

كيف فكرة في هذا, هذا البرنامج يتكون من ثلاثة أجزاء رئيسية : الدالتان **cube()** و **volumeSphere()**, و الجسم الأساسي للبرنامج .

في الجسم الأساسي في البرنامج, إستدعينا الدالة **volumeSphere()**, و سوف نمرر لها القيمة المدخلة من قبل المستخدم لقطر نصف الدائرة, و يتم تحويلها إلى عدد حقيقي بمساعدة الدالة المدمج **float()** .

داخل الدالة **volumeSphere()**, هنالك إستدعاء للدالة **cube()**.

لاحظ أن الأجزاء الثلاثة في البرنامج مرتبة بترتيب معين : أولا نبدأ بتعريف الدالات, ثم نقوم بكتابة الجزء الأساسي للبرنامج . هذا الترتيب ضروري, لأن المفسر يقوم بتنفيذ أسطر التعليمات واحدة تلو الأخرى, وفقا لترتيب ظهورها في السورس كود (شفرة البرنامج) .

> في السكريبت, يجب على تعريف الدالات أن تسبق إستخدامها (يجب عليك تعريفها في البداية) ..

لتقتنعوا, أعكس هذا النظام(على سبيل المثال, ضع جسم البرنامج في البداية), ثم لاحظ ظهور رسالة خطأ عند تشغيل السكريبت المعدل .

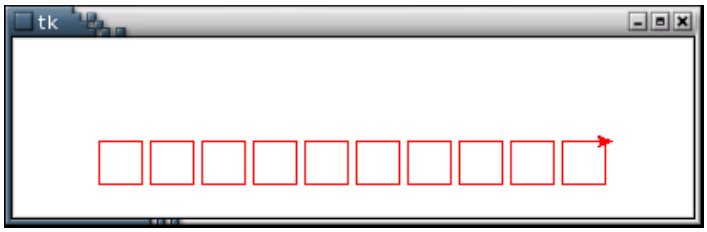
في الواقع, الجسم الأساسي لبرنامج المكتوب بالبايثون هو جزء خاص إلى حد ما, الذي يعرف دائما عند الأعمال الداخلية للمفسر تحت إسم المحجوز __main__ (الكلمة "main" تعني أساسي باللغة الأنكليزية . يكون الإسم محدد بعلامتي خط على جانبي الإسم, لتجنب الخلط بينه و بين غيره من الرموز) . و عند تشغيل السكريبت, يبدأ دائما بعبارة هذا الجزء __main__, و قد يكون هذا موجود في القائمة . و يتم تنفيذ هذه التعليمات واحدة الأخرى, في نظام, حتى إستدعاء الدالة الأولى . عند إستدعاء دالة يكون مثل إلتفاف في تدفق تنفيذ البرنامج : بدلا من الإنتقال إلى التعليمة التالية, يقوم المفسر يتنفيذ الدالة التي تم إستدعاءها, ثم يعود البرنامج إلى السطر الذي كان فيه ليكمل العمل الذي إنقطع عنه . لهذه ألية في العمل, لذلك فمن الضروري أن يكون المفسر قادرا على قراءة و تعرف الوظيفة قبل __main__, وهذا الأخير يجب وضعه في نهاية سكريبت البرنامج .

في مثالنا, جزء __main__ يستدعي الدالة الأولى و هي تستدعي الدالة الثانية . هذه الطريقة هي شائعة جدا في البرمجة . إذا أردت أن تفهم بشكل صحيح ماذا يحدث في أحد البرامج, يجب عليك إذا تعلم قراءة السكريبت, ليس من السطر الأول إلى السطر الأخير, لكن بإتباع مسار مشابه لما يحدث عند تشغيل البرنامج. هذا يعني بالضبط أنه يجب عليك تحليل السكريبت بدأ من الأسطر الأخيرة !

وحدات الدالات

حتى تتمكن من فهم جيدا التمييز بين تعريف دالة و إستخدامها في البرنامج, نقترح عليك بوضع في الكثير من الأحيان تعريفات الدالات في وحدة بايثون, و البرنامج الذي يستخدمها في مكان أخر .

مثال ::



مطلوب منك إنتاج السلسلة من الرسوم على الجانب, و ذلك بمساعدة وحدة turtle :

أكتب الأسطر البرمجية التالية, و قم حفضها في ملف بإسم **dessins_tortue.py** :

```
from turtle import *

def carre(taille, couleur):
    "fonction qui dessine un carré de taille et de couleur déterminées"
    color(couleur)
    c =0
    while c <4:
        forward(taille)
        right(90)
        c = c +1
```

قد تلاحظ أن تعريف الدالة **carre()** يبدأ ب سلسلة نصية . هذه السلسلة لا تلعب أي دور وظيفي في السكريبت : تتم معالجة هذه السلسلة كتعليق بسيط من قبل البايثون, لكن يتم تخزينها داخل جزء نظام الوثتئق الداخلي التلقائيا , ثم يمكن إستغلالها من قبل المستخدمين و الناشرين (ذكي) .

إذا كنت في بيئة IDLE, على سبيل المثال, تستطيع هذه السلسلة النصية التوثيق "تلميح", تستطيع في كل مرة يتم إستدعاء دالات موثقة جيدا .

في الواقع, البايثون يضع هذه السلسلة النصية داخل متغير خاص تحت إسم __doc__ (الكلمة "doc" بجانبها خطين من كل جهة), و يرتبط بكائن دالة مثل خصائصه(سوف نتعلم عن هذه الصفات عندما نناقش طبقات الكائنات صفحة Error: Reference source not found). و لتتمكن من الهثور على سلسلة التوثيق لدالة معينة أعرض محتوى هذ المتغير . مثال :

```
>>> def essai():
...     "Cette fonction est bien documentée mais ne fait presque rien."
...     print("rien à signaler")
```

```
...
>>> essai()
rien à signaler
>>> print(essai.__doc__)
Cette fonction est bien documentée mais ne fait presque rien.
```

خذ إذا عناء كتابة السلاسل و إبذل كل جهدك لتعرف الدالات في المستقبل : هذه الممارسة موصى بها كثيرا . الملف الذي صنعته الأن هو وحدة بايثون صحيحة, تماما مثل الوحدات turtle أو math التي قد عرفتها في وقت سابق . تستطيع الأن إستعمالها في أي سكريبت أهر, مثل هذا, على سبيل المثال, التي تؤدي العمل المطلوب :

```
from dessins_tortue import *

up()                    # إزالة القلم
goto(-150, 50)          # العودة إلى أعلى اليسار

# رسم عشرة مربعات حمراء بمحاذات بعضها البعض
i = 0
while i < 10:
    down()              # إنزال القلم
    carre(25, 'red')    # رسم مربع
    up()                # إزالة القلم
    forward(30)         # الإبتعاد
    i = i +1
a = input()             # الإنتظار
```

-

Résumé : structure d'un programme Python type

```
# -*- coding:Utf8 -*-

####################################
# Programme Python type            #
# auteur : G.Swinnen, Liège, 2009  #
# licence : GPL                    #
####################################


#####################################
# Importation de fonctions externes :

from math import sqrt


##################################
# Définition locale de fonctions :

def occurrences(car, ch):
    "Cette fonction renvoie le \
     nombre de caractères <car> \
     contenus dans la chaîne <ch>"

    nc = 0

    i = 0

    while i < len(ch):

        if ch[i] == car:
            nc = nc + 1

        i = i + 1

    return nc


################################
# Corps principal du programme :

print("Veuillez entrer un nombre :")
nbr = eval(input())

print("Veuillez entrer une phrase :")
phr = input()
print("Entrez le caractère à compter :")
cch = input()

no = occurrences(cch, phr)
rc = sqrt(nbr**3)

print("La racine carrée du cube", end=' ')
print("du nombre fourni vaut", end=' ')
print(rc)

print("La phrase contient", end=' ')
print(no, "caractères", cch)
```

Un programme Python contient en général les blocs suivants, dans l'ordre :
- Quelques instructions d'initialisation (importation de fonctions et/ou de classes, définition éventuelle de variables globales).
- Les définitions locales de fonctions et/ou de classes.
- Le corps principal du programme.

Le programme peut utiliser un nombre quelconque de fonctions, lesquelles sont définies localement ou importées depuis des modules externes.
Vous pouvez vous-même définir de tels modules.

La définition d'une fonction comporte souvent une liste de PARAMÈTRES.
Ce sont toujours des VARIABLES, qui recevront leur valeur lorsque la fonction sera appelée.

Une boucle de répétition de type 'while' doit toujours inclure au moins quatre éléments :
- l'initialisation d'une variable 'compteur' ;
- l'instruction while proprement dite, dans laquelle on exprime la condition de répétition des instructions qui suivent ;
- le bloc d'instructions à répéter ;
- une instruction d'incrémentation du compteur.

La fonction "renvoie" toujours une valeur bien déterminée au programme appelant.
Si l'instruction 'return' n'est pas utilisée, ou si elle est utilisée sans argument, la fonction renvoie un objet vide : 'None'.

Le programme qui fait appel à une fonction lui transmet d'habitude une série d'ARGUMENTS, lesquels peuvent être des valeurs, des variables, ou même des expressions.

> إنتبه
>
> يمكنك تسمية وحدات دالاتك على النحو الذي تراه مناسبا . و لكن يجب أن تدرك أنه لا يمكن إستدعاء وحدة إذا كان إسمها محجوز للبايثون (الموجودة في الصفحة), لأن إسم الوحدة المستدعية ستصبح متغير في سكريبتك, و الكلمات المحجوزة لا يمكن أن تستخدم كأسماء متغيرات . تذكر أيضا أنه لا يمكنك إعطاء إسم لوحداتك (و لكل سكريبتاتك بصفة عامة) نفس إسم لوحدة بايثون موجودة, و إلا سوف تحدث مشاكل . على سبيل المثال, إذا أعطيت إسم **turtle.py** لنمريت الذي وضعت فيها تعليمة لإستدعاء وحدة .**turtle.py**, فيتم إستدعاء هذا التمرين نفسه !

تمارين

7.2 عرف الدالة **ligneCar(n, ca)** التي تقوم بإرجاع سلسلة نصية n ل ca

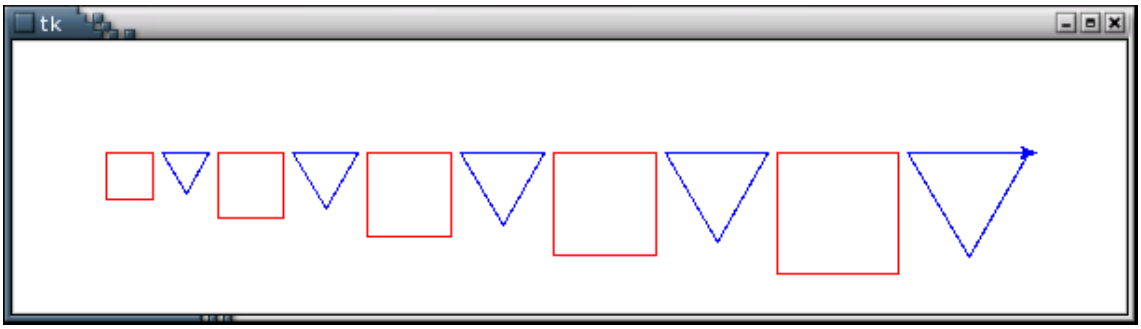
7.3 عرف الدالة **surfCercle(R)** هذه الدالة تقوم بإرجاع السطح (المنطقة) لدائرة التي قدمنا لك نفس قطرها **R** في برامتر . على سبيل المثال, عند تنفيذ التعليمة : **print(surfCercle(2.5))** يجب أن يكون الناتج : 19.63495...

7.4 عرف الدالة **volBoite(x1,x2,x3)** التي تقوم بإرجاع حجم علبة متوازية التي تم وضع ثلاثة أبعادها **x1, x2, x3** كبرامتر . على سبيل المثال, عند تنفيذ التعليمة **print(volBoite(5.2, 7.7, 3.3))** يجب أن يكون الناتج 132.132.

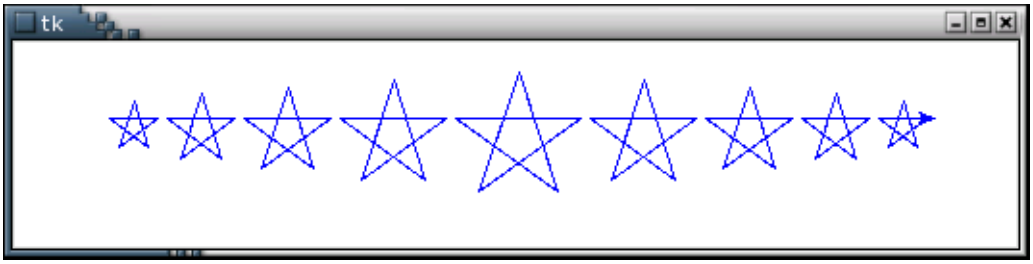
7.5 عرف الدالة **maximum(n1,n2,n3)** التي تقوم بإرجاع أكبر عدد بين 3 أعداد **n1, n2, n3** التي هي في البرامتر . على سبيل المثال عند تنفيذ التعليمة التالية : **print(maximum(2,5,4))** يجب أن يكون الناتج : 5 .

7.6 أكمل وحدة الدالات الرسومية **dessins_tortue.py** و التي تم وصفها في الصفحة Error: Reference source not found.

إبدأ بإضافة البرامتر angle إلى دالة **carre()** , بحيث يمكن وضع المربعات في إتجاهات مختلفة . ثم حدد الوظيفة **triangle(taille, couleur, angle)** القادرة على رسم مثلث متوازي الأضلاع بلون و إتجاه الموضوع كبرامتر . إختر الوحد الخاص بك بمساعدة برنامج التي يقوم بإستدعاء هذه الدالات عدة مرات, مع مجموعة من البرامترات المتنوعة التي تقوم برسم مجموعة مربعات و المثلثات :

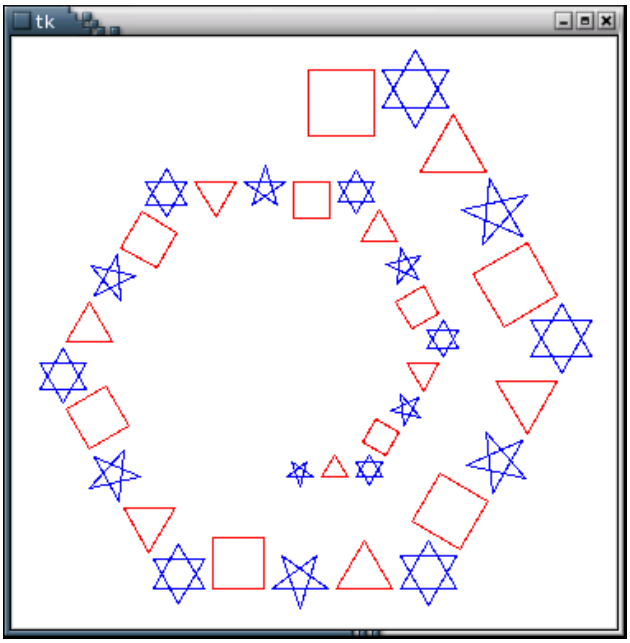


7.7 أضف إلى وحدة التمرين السابق دالة **etoile5()** عرف الدالة **maximum(n1,n2,n3)** التي تقوم بإرجاع أكبر عدد بين 3 أعداد n1, n2, n3 التي هي في البرامتر . على سبيل المثال عند تنفيذ



التعليمة التالية : **print(maximum(2,5,4))** يجب أن يكون الناتج : 5 . المتخصصة برسم نجمة مع خمسة أفرع . داخل البرنامج الأساسي, أضف حلقة التي ترسم مجموعة من 9 نجمات صغيرة بأحجام مختلفة :

7.8 أضف إلى وحدة التمرين السابق دالة **etoile6()** التي تقوم برسم تجمة ب 6 أفرع, و هي تتكون من مثلثين متداخلين في بعضهما البعض . هذه الدالة الجديدة تقوم بإستدعاء الدالة **triangle()** التي تم تعريفها سابقا . و برنامجك يجب أن يقوم برسم سلسلة من هذه النجوم :



7.9 عرف دالة **compteCar(ca,ch)** التي تقوم بإرجاع عدد مرات التي يتكرر فيها الحرف **ca** داخل السلسلة النصية **ch** . على سبيل المثال, عند تنفيذ التعليمة : **print(compteCar('e', 'Cette phrase est un exemple'))** يعطينا الناتج : 7

7.10 عرف الدالة **indexMax(liste)** التي تقوم بإرجاع مؤشر العنصر القيمة الأعلى داخل السلسلة على شكل برامتر . مثال للتشغيل :

serie = [5, 8, 2, 1, 9, 3, 6, 7]

print(indexMax(serie))

4

7.11 عرف الدالة **nomMois(n)** التي تقوم بإرجاع إسم شهر للسنة على سبيل المثال, عند تنفيذ التعليمة :

print(nomMois(4))

تقوم بإعطاء الناتج : أفريل, نيسان

7.12 عرف الدالة **inverse(ch)** التي تقوم بعكس ترتيب حروف في أي سلسلة . السلسلة المعكوسة سيتم إرجاعها للبرنامج الذي إستدعى الدالة .

7.13 عرف الدالة **compteMots(ph)** التي تقوم بإرجاع عدد الكلمات التي تحتويها الجملة ph . و نعتبر الكلمة هي مجموعة من الحروف و يكون بين الكلمات مسافات .

كتابة البرامترات

لقد تعلمت كتابة المتغيرات في البايثون بشكل ديناميكي, و هذا معناه أنه يتم تعريف نوع المتغير في نفس الوقت الذي تقوم بوضع قمته . هذه الألية تعمل أيضا للبرامترات للدالة . نوع البرامتر يصبح تلقائيا عندما يتم تمرير القيمة كبرامتر للدالة . على سبيل المثال :

```
>>> def afficher3fois(arg):
...     print(arg, arg, arg)
...

>>> afficher3fois(5)
5 5 5

>>> afficher3fois('zut')
zut zut zut

>>> afficher3fois([5, 7])
[5, 7] [5, 7] [5, 7]
```

```
>>> afficher3fois(6**2)
36 36 36
```

في هذا المثال, قد تجد أن الدالة **afficher3fois()** تقبل جميع أنواع البرامترات التي يتم تمريرها على مختلف أنواعها, وهي رقم, سلسلة نصية, قائمة أو حتى تعبير . في الحالة الأخيرة, البايثون يقوم بفحص التعبير, و يقوم بتمرير ناتج عملية التعبير كبرامتر للدالة .

قيم إفتراضية للبرامترات

في تعريف الدالة, من الممكن (و مرغوب في الكثير من الأحيان) تعريف قيمة برامتر إفتراضية لكل برامتر . و هذا يعطي الدالة التي نستطيع تسكيتها مع مجموعة فقط من البرامترات المنتظرة . على سبيل المثال :

```
>>> def politesse(nom, vedette ='Monsieur'):
...     print("Veuillez agréer ,", vedette, nom, ", mes salutations cordiales.")
...

>>> politesse('Dupont')
Veuillez agréer , Monsieur Dupont , mes salutations cordiales.

>>> politesse('Durand', 'Mademoiselle')
Veuillez agréer , Mademoiselle Durand , mes salutations cordiales.
```

عند إستدعاء هذه الدالة, القيمة الأولى قد وضعناها أما القيمة الثانية ستأخذ القيمة الإفتراضية . و إذا أدخلنا قيمتان, القيمة الإفتراضية الثانية سوف تلغى .

يمكن تعيين قيمة إفتراضية لكن البرامترات, أو جزء منها فقط . في هذه الحالة , و مع ذلك , البرامترات بدون قيم يجب أن تسبق بقية القيم . على سبيل المثال, المثال في الأسفل غير صحيح :

```
>>> def politesse(vedette ='Monsieur', nom):
```

مثال أخر :

```
>>> def question(annonce, essais =4, please ='Oui ou non, s.v.p.!'):
...     while essais >0:
...         reponse = input(annonce)
...         if reponse in ('o', 'oui','O','Oui','OUI'):
...             return 1
...         if reponse in ('n','non','N','Non','NON'):
...             return 0
...         print(please)
...         essais = essais-1
...
>>>
```

يمكن إستدعاء هذه الدالة بطرق مختلفة, على سبيل المثال :

```
rep = question('Voulez-vous vraiment terminer ? ')
```

أو :

```
rep = question('Faut-il effacer ce fichier ? ', 3)
```

أو :

```
rep = question('Avez-vous compris ? ', 2, 'Répondez par oui ou par non !')
```

خذ وقت في تشريح هذا المثال .

برامترات مع علامات

في معظم لغات البرمجة, البرامترات التي نضعها عند إستدعاء الدالة تكون في نفس مكانها في تعريف الوظيفة .

البايثون تسمح بقدر كبير من المرونة . إذا حصلت البرامترات في تعريفها في الدالة على قيمة , كما هو موضح أعلاه, يمكننا إستدعاء الدالة عن طريق تقديم البرامترات على أي ترتيب, على شرط أن نكتب إسم البرامتر بشكل صحيح, على سبيل المثال :

```
>>> def oiseau(voltage=100, etat='allumé', action='danser la java'):
...     print('Ce perroquet ne pourra pas', action)
...     print('si vous le branchez sur', voltage, 'volts !')
...     print("L'auteur de ceci est complètement", etat)
...

>>> oiseau(etat='givré', voltage=250, action='vous approuver')
Ce perroquet ne pourra pas vous approuver
si vous le branchez sur 250 volts !
L'auteur de ceci est complètement givré

>>> oiseau()
Ce perroquet ne pourra pas danser la java
si vous le branchez sur 100 volts !
L'auteur de ceci est complètement allumé
```

تمارين

1.3 عدل الدالة **volBoite(x1,x2,x3)** التي تم تعريفها في التمرين السابق, بحيث يمكن إستدعائها ب براماتر واحد أو إثنين أو ثلاثة برامترات, أو بدون برامترات . إستخدم القيم الإفتراضية للقيم هي 10, على سبيل المثال :

print(volBoite)() نتيجته : 1000

print(volBoite(5.2)) نتيجته: 520.0

print(volBoite(5.2, 3)) نتيجته: 156.0

2.3 عدل الدالة **volBoite(x1,x2,x3)** التي في الأعلى بطريقة بحيث يمكننا إستدعائها مع برامتر واحد أو إثنين أو ثلاثة برامترات . في حالة إستخدام برامتر واحد, الصندوق يكون على شكل مكعب (البرامترات يجب أن تكون تعبر عن الحافة) . إذا تم إستخدام برامترين, يبدوا كأنه مربع منشور (في هذه الحالة البرامتر الأولى للجانب و الثانية لإرتفاع المنشور) . و إذا ثلاثة, تكون على شكل متوازي, على سبيل المثال :

print(volBoite()) (نتيجته: -1 (يشير إلى خطأ

print(volBoite(5.2)) نتيجته: 140.608

print(volBoite(5.2, 3)) نتيجته: 81.12

print(volBoite(5.2, 3, 7.4)) نتيجته: 115.44

3.3 عرف دالة **changeCar(ch,ca1,ca2,debut,fin)** التي تبدل كل حروف ca1 ب حروف ca2 في سلسلة نصية ch, بداية من المؤشر debut و إلى المؤشر fin, هذان البرامتران الأخيران يمكننا تركهم (وفي هذه الحالة يتم التعامل مع سلسلة واحدة من البداية إلى النهاية), أمثلة على الدالة المتوقعة :

```
>>> phrase = 'Ceci est une toute petite phrase.'
>>> print(changeCar(phrase, ' ', '*'))
Ceci*est*une*toute*petite*phrase.
>>> print(changeCar(phrase, ' ', '*', 8, 12))
Ceci est*une*toute petite phrase.
>>> print(changeCar(phrase, ' ', '*', 12))
Ceci est une*toute*petite*phrase.
>>> print(changeCar(phrase, ' ', '*', fin = 12))
Ceci*est*une*toute petite phrase.
```

4.3 عرف الدالة **eleMax(liste,debut,fin)** لبتي تقوم بإرجاع القيمة الأعلى في السلسلة التي تم تمريرها , البرامتران debut و fin تشير إلى المؤشرات التي ينبغي البحث عنها, و يمكن حذفهم(كما في التمرين السابق). أمثلة على الدالة المتوقعة :

```
>>> serie = [9, 3, 6, 1, 7, 5, 4, 8, 2]
>>> print(eleMax(serie))
9
>>> print(eleMax(serie, 2, 5))
7
>>> print(eleMax(serie, 2))
8
>>> print(eleMax(serie, fin =3, debut =1))
6
```

8

استخدام النوافذ والرسومات

حتى الأن , إستخدمنا البايثون فقط في "الوضع النصي" لأنه يجب علينا أن نتعلم أولا عدد من المفاهيم الأساسيةو البنية الأساسية للغة, قبل أن نبدأ التعلم أشياء أكثر صعوبة و تطورا(مثل النوافذ و الصور و الأصوات, إلخ ...)يمكننا الأن التوغل في البايثون و الدخول إلى حقل واسع من الواجهات الرسومية , لكن هذا لن يكون سوى البداية : على الرغم من أننا لم نتعلم الكثير و مازال أمامنا الكثير من الأساسيات يجب أن نتعلمها, و ربما أصبح "الوضع النصي " محبوبا لدى الكثير منكم .

واجهات المستخدم الرسومية (GUI)

إن كنت تجهل هذا حتى الأن , إعلم أن مجال الواجهات الرسومية في غاية التعقيد و الصعوبة . لكل نظام تشغيل يتوفر عدة "مكتبات" لوظائف الرسم الأساسية , التي تضاف إلى (في كثير من الأحيان) العديد من المكملات , (أكثر أو أقل بحسب لغات البرمجة)و تعرض جميع هذه المكونات بشكل عام فئات للكائن (كلاس أوبجيكت) و التي سندرس سماتها و أساليبها .

مع البايثون , المكتبة الرسومية الأكثر إستخداما حتى الأن (هذا الكتاب قديم) مكتبة تكنتر الذي هو تكييف لمكتبة تاكا و ضعت أصلا للغة برمجة Tcl و wxPython : وهناك أيضا عدة مكتبات رسومية للغة برمجة بايثون مثل PyQT و Pygtk ...إلخ

و هناك إمكانية لإستخدام مكتبات جافا و مكتبات ميكروسوفت أم أف سي لنظام ويندوز. إضافة إلى هذا نحن سنتعلم فقط البرمجة بإستخدام تكنتر التي توجد لحسن الحظ نسخ لعدة أنظمة تشغيل (وبشكل مجاني) منها ويندوز و لينكس و ماك

الخطوات الأولى مع Tkinter

للمزيد من الإيضاح , نحن نفترض بالطبع أن وحدة Tkinter[25] مثبة مسبقا على نظامك . لتكون قادرا على إستخدام مميزات تكنتر يجب عليك أن تستدعيه (بسطر واحد فقط) بإضافة هذا السطر إلى ملف البرنامج :

```
from tkinter import *
```

كالعادة , على البايثون ليس من الضروري كتابة سكريبت بل تستطيع فعل هذا من خلال سطر الأوامر (بعد تشغيل البايثون) في مثالنا التالي سوف نقوم بإنشاء نافذة بسيطة , ثم نضيف فيها أداتين[26]: أداة جزء من النص (لابل) و زر .

```
>>> from tkinter import *
>>> fen1 = Tk()
>>> tex1 = Label(fen1, text='Bonjour tout le monde !', fg='red')
>>> tex1.pack()
>>> bou1 = Button(fen1, text='Quitter', command = fen1.destroy)
>>> bou1.pack()
>>> fen1.mainloop()
```

> إعتمادا على هذه النسخة من البايثون , سوف نرى نافذة التطبيق تظهر مباشرة بعد إدخال الأمر الثاني في مثالنا هذا أو بعد السطر السابع فقط[27].

---

[25] في إصدارات البايثون السابقة (قبل الإصدار الثالث) تبدأ إسم الوحدة بحرف كبير

[26] الويدجت هي نتيجة لإنكماش عبارة نافذة الأداة . في بعض لغات البرمجة , هذه ليست ما يطلق علها السيطرة أو المكون الرسومي هذا المصطلح يشير إلى أي شئ يمكن وضعه في إطار التطبيق : مثل الزر و الصور و إلخ ... و أحيانا النافذة نفسها .

[27] إذا قمت بإجراء هذه العملية تحت نظام ويندوز , يجب عليك إستخدام و يفضل أن يكون الإصدار القياسسي من البايثون في إطار دوس في إي دي أل إي أو بايثون وين بدلا من ذلك . يمكنك أن ترى أفضل , مايحدث بعد إدخال كل أمر .

دعونا الأن نبحث عن المزيد في كل سطور الأوامر المنفذة 1. كما سبق شرحه أعلاه, فإنه من السهل بناء وحدات البايثون المختلفة, و التي تحتوي على سكريبتات, تعريفات الدالات, أصناف الكائنات, إلخ ... يمكننا إذا إستدعاء جزء أو كل من هذه الوحدات لأي برنامج, حتى لو كنا داخل مفسر يعمل بالوضع التفاعلي(هذا معناه مباشرة إلى سطر الأوامر). هذا ما فعلناه في السطر الأول للمثالنا : **from tkinter import *** , معناه إستدعاء جميع الأصناف في وحدة tkinter.

سيكون لدينا المزيد حول هذه الفئات. في البرمجة, تسمي مولدات الكائنات, و هي جزء من البرنامج يمكن إعادة إستخدامه . نحن لا نريد أن نعطيك التعريف محدد و الدقيق للكئنات و الأصناف, لكن أقترح أن نستخدمهم بشكل مباشر و ليس جزئي . سوف نفهم هذا تدريجيا .

2. في السطر الثاني من مثالنا : **fen1 = Tk()**, نحن إستخدمنا صنف للوحدة tkinter, و الصنف **Tk()**, و نحن أنشأنا مثيل (إسم أخر يصف كائن محدد), أي النافذة **fen1** .

هذه عملية تمثيل كائن من العمليات الأساسية في التقنيات الحالية للبرمجة . هذه الطريقة في الواقع الأكثر إستخداما و نعرف بإسم البرمجة الشيئية (أو OOP أي البرمجة الموجهة) .

صنف هو نموذج عام يبدأ من أن نطلب من الألة بناء كائن حاسوبي معين . الصنف يحتوي على مجموعة من التعريفات للخيارات المختلفة, نحن لن نستخدم سوى جزء من الكائن الذي صنعناه إبتداءا منها . و بالتالي الصنف **Tk()** , الذي يعد من الفئات الرئيسية لمكتبة tkinter, و يحتوي على كل ما هو مطلوب لتوليد أنواع مختلفة من نوافذ التطبيقات, مختلفة الأحجام و الألوان, مع أو بدون شريط أوامر ... إلخ.

نحن نستخدمها هنا لصناعة كائن غرافيكي أساسي, أي نافذة تحتوي على كل ما تبقى . في أقواس **Tk()** , يمكننا تحديد خيارات مختلفة, لكن سنترك هذا إلى وقت أخر .

تجسيد التعليمة يشبه تعيين بسيط لمتغير . إفهم من ذلك أنه يحدث هنا شيئين في وقت واحد :

* إنشاء كائن جديد, (و الذي قد يكون معقد للغاية في بعض الحالات, و بالتالي يحتل مساحة كبيرة في الذاكرة)

* تعيين المتغير, و الذي سيعمل الأن كمرجع لمعالجة الكائن .

3. في السطر الثالث :

tex1 = Label(fen1, text='Bonjour tout le monde !', fg='red')

نحن سنصنع كائن أخر (ودجيت), و هذه المرة من الصنف **Label()** . كما يوحي لنا إسمه, هذا الصنف يعرف جميع أنواع التسميات(أو العلامات) . في الواقع, هو ببساطة هو جزء من النص, يستخدم لعرض معلومات و رسائل مختلفة داخل النافذة .

سنسعى جاهدين لتمرير الطريقة الصحيحة للتعبير عن الأشياء, نقول هنا أننا صنعنا الكائن **tex1** بواسطة مثيل الصنف **Label()** .

لاحظ أننا قمنا بإستدعاء الصنف, بنفس الطريقة التي إستدعينا فيها الدالة : و هذا معناه تقديم عدد من البرامترات داخل الأقواس . سوف نرى لاحقا أن الصنف هو نوع من أنواع "الحاويات, و التي تم تجميع فيها مجموعة من الدوالات و المعطيات .

ما هي البرامترات التي قدمناها لهذا المثيل ؟

* البراماتر الأول الذي تم تمريره هو **fen1**, يشير إلى أن الويدجت الأول الذي قمنا بصنعه داخل وجدت سابق, الذي وضعناه هنا مثل "سيده" : الكائن **fen1** هو الويدجت السيد للكائن **tex1** . نستطيع أن نقول أن الطائن **tex1** هو ويدجت عبيد للكائن **fen1** .

هذان البرامتران يستخدمان ليصفان بالضبط ماذا يجب أن يأخذ الويدجت . هذا في الواقع إختيارين للصنع, قدم لكل واحد في شكل سلسلة نصية / في البداية النص التسمية, ثم اللون(foreground أو بإختصار **fg**) . نحن نريد أن يظهر النص بشكل جيد, لذلك لوناه باللون الأحمر .

و يمكننا أيضا تحديد المزيد من الخصائص الأخرى : مثل الخط أو اللون الخلفي على سبيل المثال . كل هذه الخصائص لديها قيم إفتراضية في تعريف الصنف **Label()** . لا يمكننا تحديد جميع الخيارات المتاحة لخصائص المختلفة عن النموذج القياسي .

في السطر الرابع من مثالنا : **tex1.pack()** , فعّلنا الأسلوب المرتبط بالكائن **tex1** : الأسلوب **pack()** . لقد إلتقينا بالفعل مع هذا الأسلوب(عن القوائم خاصةً) . و هنالك أسلوب الدالة مضمنة في الكائن (نقول أيضا كما يتم تغليف الكائن) . و نحن نعلم عما قريب أن الكائن الحاسوبي هو في الواقع عنصر لبرنامج يحتوي دائما على :

* عدد من البيانات(رقمية أو غيرها), تحتوي في داخل المتغيرات من أنواع مختلفة : نسميها خصائص الكائن .

و يطلق على مجموعة من الإجراءات و الدوالات(والتي هي خوارزمية) : أساليب الكائن .

الأسلوب **pack()** هو مجموعة من الأساليب التي تطبق ليس فقط على ويدجت الصنف **Label()** , بل تطبق في معظم الويدجات الأخرى ل tkinter, و التي تؤثر على ترتيبها في إطار الهندسي في النافذة . كما يمكنك أن ترى بنفسك إذا قمت بإدخال الأوامر مثالنا واحدة تلو الأخرى, الأسلوب **pack()** يقلل تلقائيا حجم نافذة - السيد - بحيث تكون كبيرة لإضافة ما يكفي من الويدجات - العبيد - المحددة مسبقا .

5. في السطر الخامس :

bou1 = Button(fen1, text='Quitter', command = fen1.destroy)

صنعنا الويدجت الثاني - عبيد - : زر

كما فعلنا مع الويدجت السابقة, نحن إستدعينا الصنف **Button()** مصحوبة بقوسين بداخلها البرامترات . لأنه في هذه الحالة من الكائن التفاعلي, يجب علينا أن نضع خيار ماذا سيحدث عندما يقوم المستخدم بالضغط على الزر . في هذه الحالة, نحن وضعنا خيار إغلاق مرتبط بالكائن fen1, الذي ينبغي أن يتسبب بإغلاق النافذة .

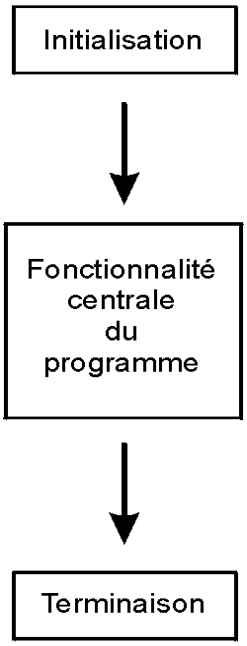
6. في السطر السادس إستخدمنا الأسلوب **pack()** حتى يتكيف هندسيا في النافذة مع الكائن الجديد لدمجه .

7. في السطر السابع: **fen1.mainloop()** مهم للغاية, لأنه يتسبب ببدء الأحداث المرتبطة بالنافذة . هذه التعليمة ضرورية للغاية لتطبيقنا سواء ل - الإطلاع - على نقرات الفأرة, أو للضغطات على لوحة المفاتيح, إلخ ... إذا هذه التعليمة بتعبير أخر - تجعله يعمل - .

كما يوحي إسمها (mainloop), هو أسلوب للكائن **fen1**, الذي يفعّل حلقة البرنامج, الذي يعمل في الخلفية بشكل مستمر, في إنتظار رسائل من قبل نظام التشغيل المثبت على الحاسوب . ينتظر في الواقع بشكل مستمر في بيئته, أجهزة الإدخال (الفأرة, لوحة المفاتيح, إلخ ...). عندما يتم الكشف عن أي حالة, يتم إرسال رسائل مختلفة التي تصف الحالة إلى البرنامج . سنتعرف على التفاصيل قريبا .

برامج تتوجه حسب الأحداث

لقد صنعت برنامجك الأول مستخدما الواجهة الرسومية . هذا النوع من البرامج يتنظم بطريقة مختلفة عن السكريبتات التي درسناها سابقا .

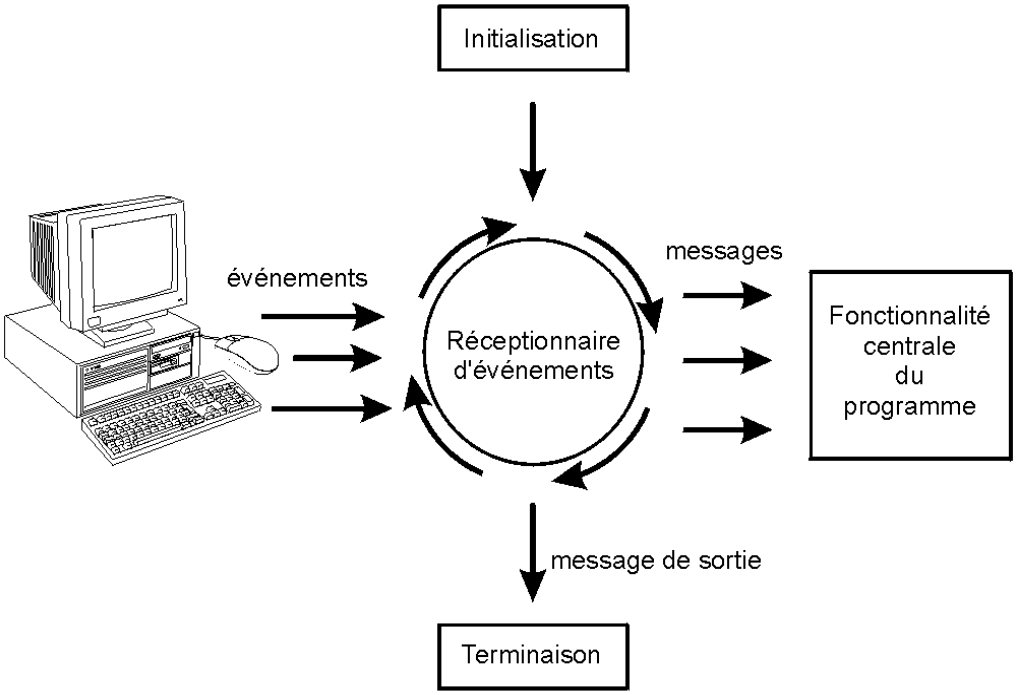


جميع برامج الحاسوب لديك تعمل بثلاثة مراحل رئيسية : مرحلة التهيئة, التي تحتوي على التعليمات التي تعّد العمل المطلوب (إستدعاء وحدات خارجية اللازمة,فتح ملفات, الإتصال بخادم قواعد البيانات أو في شبكة الأنترنات, إلخ ..), مرحلة الوسطى(المركزية) حيث نجد هناك الدوالات الرئيسية للبرنامج(هذا معناه كل شيئ من المفترض أن يفعله البرنامج : عرض البيانات على الشاشة, تنفيذ العمليات الحسابية, تحرير محتويات لملف, طباعة, إلخ ...), و في النهاية مرحلة الإنتهاء و الذي تعمل على إغلاق المعاملات - بشكل صحيح - (هذا معناه إغلاق الملفات المفتوحة, قطع الإتصالات الخارجية, إلخ ...)

في البرنامج - بالوضع النصي - , يتم ترتيب هذه المراحل الثلاثة ببساطة في نمط خطي كما تم توضيحه . و بناءً على ذلك, تتميز هذه البرامج بتفاعلها المحدود جدا مع المستخدم . هذه م الناحية العملية ليس لديك أي حرية : يطلب منك من وقت لأخر إدخال بعض البيانات من لوحة المفاتيح, لكن دائما في ترتيب محدد سابقا لسلسلة من تعليمات البرنامج .

في حالة أن البرنامج يستخدم الواجهة الرسومية, يكون التنظيم الداخلي هو المختلف . نقول أن البرنامج يتوجه بواسطة الأحداث . بعد مرحلة التهيئة, البرنامج من هذا النوع يبقى ينتظر, و يمرر السيطرة على برنامج أخر, و التي هي أكثر أو أقل إندماجا مع نظام التشغيل الموجود على الحاسوب .

هذا متلقي الأحداث يقوم بإستمرار بفحص الملحقات(لوحة المقاتيح, الفأرة, الساعة, المودم, إلخ ...) و يتفاعل فور الكشف عن حصول حدث . و يمكن أن يكون هذا الحدث من المستخدم : تحريك الفأرة, الضغط على مفتاح في لوحة المفاتيح, إلخ ... أو يمكن حدث خارجي أو تلقائي(إنتهاء المؤقت, على سبيل المثال) .



عندما يتم كشف حدث, يرسل المتلقي رسالة معينة إلى البرنامج[28], الذي هو مصمما ليرد وفقا لذلك .

مرحلة التهيئة لبرنامج يستخدم واجهة رسومية تتضمن مجموعة من التعليمات التي تضع مكونات الواجهة التفاعلية في مكانها(النوافذ, الأزرار, الخانات, إلخ ...). المزيد من تعليمات التي تعرف رسائل الأحداث تكون مدعومة : في الواقع, يمكن للمرء أن يقرر ردت فعل البرنامج على أحداث معينة و يتجاهل البقية .

28هذه الرسائل غالبا ما يتم تدل على WM (رسائل النافذة) في بيئة رسومية تتكون من نوافذ (مع مناطق فعالة كثيرة : الأزرار, خانات الإختيار, القوائم, إلخ) . في وصف الخوارزميات, كما يحدث في كثيرة من الأحيان تختلط هذه الرسائل مع الأحداث نفسها .

بينما تكون المرحلة الوسطى في البرنامج النصي, تتكون من سلسلة من التعليمات التي تصف الترتيب المهام التي ينبغي أن يؤديها البرنامج(حتى لو تم تقديمها في مسارات مختلفة إستجابة للظروف التي تواجه), لا توجد مرحلة وسطى في البرنامج الذي يستخدم الواجهة الرسومية بل تكون مجموعة من الدوالات المستقلة. و تستدعى كل دالة خاصة عندما يتم الكشف عن حدث معين من قبل نظام التشغيل : يتم تنفيذ الدالة لتقوم بالعمل المتوقع للبرنامج في إستجابة لهذا الحدث, ثم لا شيئ أخر .[29].

من المهم أن نفهم هنا في كل هذا الوقت, المتلقي يبقى يعمل لينتظر حدوث أحداث أخرى محتملة .

إذا حَدَثَ حدْث أخر, يمكن أن يكون الرد من الدالة الثانية(أو الثالثة, أو الرابعة, إلخ...)التي سوف يتم تفعيلها لتبدأ عملها بالتوازي مع الدالة الأولى التي لم تكمل عملها بعد .[30]. يمكن للأنظمة التشغيل و الغات البرمجة الحديثة أن تعمل بالتوازي و التي نسميها أيضا تعدد المهام .

في الفصل السابق, لاحظنا بالفعل أن بنية الملف النصي لبرنامج غير مشابه لبنية الملف عندما يتم تنفيذه . هذه الملاحظة تطبق أيضا على البرنامج مع الواجهة الرسومية, حيث ترتيب الدالات التي يتم إستدعائها غير مسجلة بأي جزء من البرنامج . الأحداث هي التي تتحكم !

كل هذا قدي يبدو معقدا قليلا . سوف نوضح هذا في بعض أمثلة .

مثال غرافيسكس : خطوط رسم على اللوحة

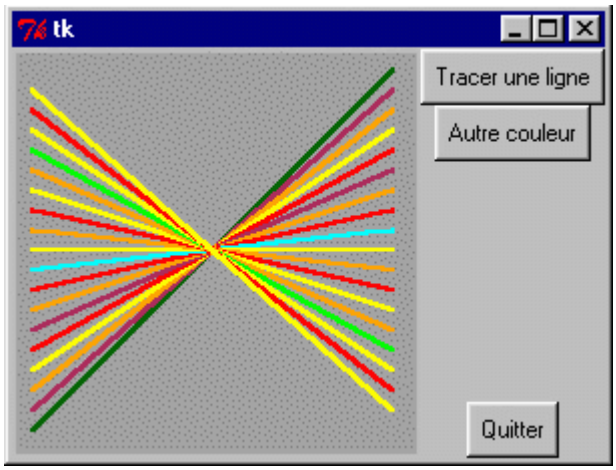
السكريبت الذي بالأسفل يصنع نافذة مع ثلاثة أزرار و لوحة . بمصطلحات tkinter, اللوحة - canevas - هي مساحة مستطيلة محددة, يمكن أن يوضع بها مختلف التصاميم و الصور بإستخدام أساليب محددة[31].عند الضغط على زر "رسم خط" - "Tracer une ligne", سيظهر سطر ملون جديد على اللوحة, كل واحد لديها ميل مختلف عن سابقتها .

---

29 بالمعنى الدقيق للكلمة, أي الدالة لا ترجع أية قيمة هي إجراء (أنظر إلى صفحة :) .

30 نفس الدالة يمكن أن يتم إستدعائا عدة مرات ردا على وقوع بعض الأحداث نفسها, ثم يتم تنفيذ نفس المهمة بنسخ مختلفة , سوف نرى لاحقا أنه يمكن أن يؤدي إلى "آثار حافة" مزعجة .

31 في النهاية سيتم تحريك هذه الرسوم في مرحلة لاحقة .



إذا تم الضغط على زر "لون أخر" - " Autre couleur ", سيتم إختيار لون جديد من سلسلة الألوان المحددة . هذا اللون سيتم إستخدامه في الرسم القادم .

زر "خروج" - " Quitter " لإنهاء الطبيق عن طريق غلق النافذة .

```
# tkinter تمرين صغير يستخدم مكتبة الرسومية

from tkinter import *
from random import randrange

# --- : تعريف دالات لمعالجة الأحداث ---
def drawline():
    "Tracé d'une ligne dans le canevas can1"
    global x1, y1, x2, y2, coul
    can1.create_line(x1,y1,x2,y2,width=2,fill=coul)

    # : تعديل الإحداثيات للسطر التالي
    y2, y1 = y2+10, y1-10

def changecolor():
    "Changement aléatoire de la couleur du tracé"
    global coul
    pal=['purple','cyan','maroon','green','red','blue','orange','yellow']
    c = randrange(8)          # => توليد رقم عشوائي بين 0 و 7
    coul = pal[c]

#------ البرنامج الرئيسي -------

# : سيتم إستخدام المتغيرات التلية بشكل عام
x1, y1, x2, y2 = 10, 190, 190, 10       # إحداثيات السطر
```

```
coul = 'dark green'                    # لون السطر

# إنشاء الويدجت الرئيسي ("السيد") :
fen1 = Tk()
# إنشاء الويدجت ("العبيد") :
can1 = Canvas(fen1,bg='dark grey',height=200,width=200)
can1.pack(side=LEFT)
bou1 = Button(fen1,text='Quitter',command=fen1.quit)
bou1.pack(side=BOTTOM)
bou2 = Button(fen1,text='Tracer une ligne',command=drawline)
bou2.pack()
bou3 = Button(fen1,text='Autre couleur',command=changecolor)
bou3.pack()

fen1.mainloop()   # بدء إستقبال الأحداث

fen1.destroy()    # تدمير (غلق) النافذة
```

وفق لما شرحناه في صفحات السابقة, وظيفية هذا البرنامج تقوم على دالتين أساسيتين **drawline()** و **()changecolor** , التي يتم تفعيلها من خلال الأحداث, لأنها تم تفعيلها في مرحلة التهيئة .

في هذه المرحلة - مرحلة التهيئة -, نحن نبدأ بإستدعاء وحدة tkinter بالإضافة إلى وحدة random التي تقوم بإختيار رقم عشوائي . ثم صنعنا الويدجات المختلفة مثل **Tk()** و **Canvas()** و **Button()** . لاحظ أن بالتمرير للصنف **Button()** صنعنا مجموعة من الأزرار, التي هي مشابهة لبعضها جدا, مع خيارات لكل واحدة منها لصناعتها, و الأزرار تستطيع أن تعمكل واحدة بشكل مستقل عن الأخرى .

مرحلة التهيئة تنتهي مع التعليمة **fen1.mainloop()** التي تبدأ بتلقي الأحداث . التعليمات التي تأتي بعدها ستعمل عندما يتم الخروج من الحلقة, ستخرج من خلال أسلوب **fen1.quit()** (أنظر أدناه) .

خيار الأمر المستخدم في عبارة تجسيد الأزرار التي ترسم الدالة االتي سيتم إستدعائها عندما يعمل هذا الحدث " ضغط على الأزر الأيسر للفأرة على الويدجت " . في الواقع يجب عمل إختصار لهذا الحدث حاصة, و الذي يتم تقديمه من tkinter لراحتك لأن هذا الحدث يرتبط بشكل طبيعي مع الويدجت من نوع زر . سوف نرى لاحقاأن هنالك تقنيات أخرى أكثر عمومية لربط أي نوع من الأحداث إلى أي قطعة .

يمكن للدالات في هذا السكريبت تعديل - تحرير - قيم لمتغيرا المعرفة في الجزء الرئيسي من البرنامج . و لقد أصبح هذا ممكن مع التعليمة global المستخدمة لتعريف هذه الدالات . نحن نسمح لأنفسنا أن نفعل ذلك لبعض الوقت (حتى لو كان فقط للتعود على التمييز بين المتغيرات المحلية و العامة), لكن كما ستفهم في

وقت لاحق, هذه الممارسة غير مستحسنة, خاصة عندما تكتب برامج كبيرة . سوف نتعلم أفضل التقنيات عندما نصل لدراسة الأصناف (بداية من الصفحة Error: Reference source not found).

في دالتنا **changecolor()**, يتم إختيار لون عشوائي من القائمة . نحن إستخدمنا للقيام بذلك الدالة **()randrange** التي تقوم بإستدعاء الوحدة random. التي يتم إستدعائها مع البرامتر **N**, هذه الدالو تقوم بإرجاع عدد صحيح, ما بين 0 و N-1 .

زر الأمر مرتبط ب - Quitter - خروج - التي تستدعي الأسوب **()quit** لنافذة **fen1** . ويستخدم هذا الأسلوب لإغلاق - خروج - من متلقي الأحداث (**mainloop**) المرتبط بهذه النافذة . عندما سيتم تفعيل هذا الأسلوب, تنفيذ البرنامج لايزال مع التعليمات التي بعد إستدعاء ال **mainloop** . في مثالنا, سيتم إزالة النافذة .

تمارين

8.1 كيف يتم تغيير البرنامج لكي تكون ألوان الخطوط : *cyan* و *maroon* و *green* ?

8.2 كيف يتم تغيير البرنامج لكي تكون جميع الخطوط أفقية و عمودية ؟

8.3 كبر حجم اللوحة لتصبح عرضها 500 وحدة و إرتفاعها 650 وحدة . و غير أيضا حجم الخطوط, لتكون حوافهم تتساوى مع حواف اللوحة .

8.4 أضف دالة **drawli ne2** التي تتبع خطين أحمرين بعلامة أكس في وسط اللوحة, واحدة أفقية و الأخرى عمودية . و أضف أيضا زر "منظار", عند الضغط عليه سوف تظهر علامة أكس .

8.5 كرر كتابة البرنامج الأول . أستبدل الأسلوب **create_line** ب **create_rectangle** . ماذا سيحدث ؟

و بنفس الطريقة حاول مع **create_arc** و **create_oval** و **create_polygon** . لكل أسلوب, أكتب خياراته في البرامترات . (ملاحظة : بالنسبة للمضلع, فمن الضروري القيام بتعديل صغير على البرنامج ليعمل !)

8.6 أحذف السطر **global x1, y1, x2, y2** في دالة **drawline** في البرنامج الأصلي .

ماذا حدث ؟ و لماذا ؟

إذا وضعت "x1, y1, x2, y2" داخل الأقواس, في سطر تعريف الدالة drawline, بطريقة لتمرير المتغيرات للدالة كبرامترات, هل يعمل البرنامج ؟ لا تنسى أيضا تغيير السطر الذي يستدعي هذه الدالة !

إذا عرّفت **x1, y1, x2, y2 = 10, 390, 390, 10** بدل **global x1, y1**... , ماذا سيحدث ؟ و لماذا ؟ ماذا إستنتجت من هذا ؟

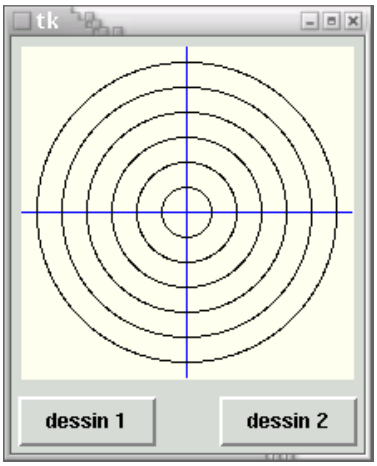
8.7 أ) أكتب برنامج قصير الذي يرسم الحلقات الأولمبية الخمسة في مستطيل أبيض (white) . بالإضافة إلى زر Quitter للخروج من المافذة .

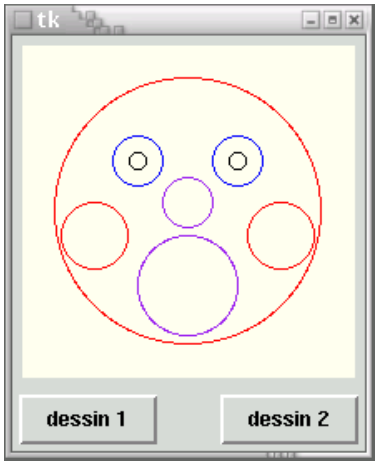
ب) عدل البرنامج أعلاه بإضافة خمسة أزرار كل زر يرسم حلقة من الحلقات الخمسة .

8.9 في دفتر الملاحظات, خطط جدول من عمودين . ستكتب على اليسار تعريفات الأصناف التي قد درسناها(مع قائمة البرامترات), و على اليمين الأساليب المرتبطة بهذه الأصناف (مع برامتراتها) . أترك بعض المجال لإكماله في وقت لاحق .

مثال غرافيكي : رسمان مناوبين

المثال التالي يظهر لك كيفية الإستفادة من المعلومات و المعرفة التي قد حصلت عليها للقوائم الحلقات و الدالات, لرسم العديد من الرسومات بأسطر قليلة . هذا البرنامج الصغير يظهر واحد من الرسمان الموجودان بالأسفل, على حسب الزر المضغوط :






```
from tkinter import *

def cercle(x, y, r, coul ='black'):
    "tracé d'un cercle de centre (x,y) et de rayon r"
    can.create_oval(x-r, y-r, x+r, y+r, outline=coul)

def figure_1():
    "dessiner une cible"
    # أحذف أولا أي رسم سابق:
```

```
    can.delete(ALL)
    # أرسم خطين (أفقي و عمودي) :
    can.create_line(100, 0, 100, 200, fill ='blue')
    can.create_line(0, 100, 200, 100, fill ='blue')
    # أرسم عدة دوائر متحدة المركز :
    rayon = 15
    while rayon < 100:
        cercle(100, 100, rayon)
        rayon += 15

def figure_2():
    "dessiner un visage simplifié"
    # أحذف أولا أي رسم سابق :
    can.delete(ALL)
    # خصائص كل دائرة
    # موضوعة في قائمة من القوائم :
    cc =[[100, 100, 80, 'red'],      # الوجه
         [70, 70, 15, 'blue'],       # العينان
         [130, 70, 15, 'blue'],
         [70, 70, 5, 'black'],
         [130, 70, 5, 'black'],
         [44, 115, 20, 'red'],       # الخدين
         [156, 115, 20, 'red'],
         [100, 95, 15, 'purple'],    # الأنف
         [100, 145, 30, 'purple']]   # الفم
    # يتم رسم جميع الدوائر بمساعدة حلقة :
    i =0
    while i < len(cc):        # تدوير القائمة
        el = cc[i]            # كل عنصر هو في حد ذاته قائمة
        cercle(el[0], el[1], el[2], el[3])
        i += 1

##### البرنامج الرئيسي : ############

fen = Tk()
can = Canvas(fen, width =200, height =200, bg ='ivory')
can.pack(side =TOP, padx =5, pady =5)
b1 = Button(fen, text ='dessin 1', command =figure_1)
b1.pack(side =LEFT, padx =3, pady =3)
b2 = Button(fen, text ='dessin 2', command =figure_2)
b2.pack(side =RIGHT, padx =3, pady =3)
fen.mainloop()
```

أبدأ بتحليل البرنامج الرئيسي, في نهاية السكريبت :

لقد قمنا بإنشاء نافذة, بتمثيل كائن للصنف **Tk()** في المتغير **fen** . ثم قمنا بوضع ثلاثة ويدجات في هذه النافذة : لوحة و زران . و لقد أنشأنا اللوحة في المتغير **can**, الزران في المتغير **b1** و **b2** . كما في السكريبت السابق, الويدجات تم وضعهم في أماكنهم في النافذة بمساعدة الأسلوب **pack()**, لكنهذه المرة إستخدمنا هذه الخيارات :

* الخيار side الذي يقبل القيم **TOP** أو **BOTTOM** أو **LEFT** أو **RIGHT**, لوضع الويدجت في الجانب المناسب في النافذة . هذه الأسماء تكتب بأحرف كبيرة و هم جزء من متغيرات التي تم إستدعاءها مع الوحدة tkinter, و يمكن أن تعتبره ك " شبه ثوابت " .

* الخياران **padx** و **pady** الذان يقومان بحجز مساحة صغيرة حول الويدجت . هذه المساحة تعبر عن عدد البيكسلات : **padx** تحجز المساحة على يمين و يسار الويدجت, و **pady** تقوم بحجز المساحة فوق و تحت الويدجت .

الأزرار تتحكم في إظهار الرسمين, بإستدعاء الدالتين **figure_1()** و **figure_2()** .

بما أننا سنرسم العديد من الدوائر في هذه الرسومات, ففكرنا أنه من المديد أن نبدأ بتعريف دالة **cercle()** لرسم الدوائر . في الحقيقة, و ربما لم تكن تعرف سابقا أن اللوحة في tkinter لديها أسلوب **()** **create_oval** التي تستطيع من خلالها رسم أية أشكال بيضوية (و بالطبع دوائر أيضا), لكن هذا الأسلوب يجب أن يحتوي عل أربعة برامترات التي هي إحداثيات أعلى و أسفل و يمين و يسار مستطيل وهمي, في أي شكل بيضوي تريد أن ترسمه . و هذا ليس عمليا في حالات معينة من الدائرة : و الأكثر طبيعية هو أن يتم تمرير طول المركز و نصف قطر الدائرة و هذا ما سنحصل عليه في دالتنا **cercle()**, و التي تستدعي الأسلوب **creat_oval()** عن طريق إجراء تحويل للتنسيق . لاحظ أيضا أن هذه الدالة يجب إعطائها لون الدائرة التي تريدها (اللون الإفتراضي هو الأسود) .

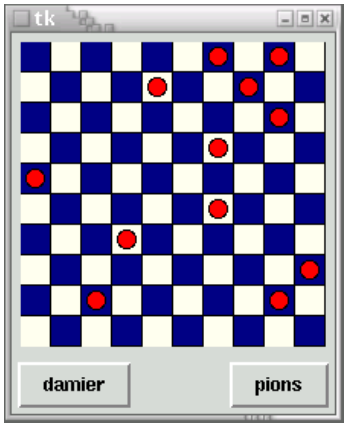
و عمل هذه لطريقة هو سهل وواضح في الدالة **figure_1()** , لقد صنعنا دالة بسيطة لتكرار رسم جميع الدوائر (بنفس المركز و بنفس قطر متزايد) . ملاحظة أخر و هي إستخدام العامل += التي تزيد قيمة المتغير (في مثالنا, المتغير **r** يزداد 15 قيمة في كل تكرار) .

الرسم الثاني هو أكثر تعقيدا نوعا ما, لأنها تتكون من دوائر مختلفة الأحجام في أماكن مختلفة . نستطيع رسم كل هذه الدوائر بمساعدة حلقة تكرار واحدة, إذا عرفنا كيفية الإستفادة من القوائم .

في الحقيقة أن الفرق بين الدوائر التي رسمناهاتتلخص في ثلاثة خصائص :

إحداثيات **x** و **y** للوسط, و المركز و اللون . لكن دائرة, نستطيع أن نضع هذه الخصائص في قائمة صغيرة, ثم نقوم بجمع كل هذه القوائم الضغيرة في قائمة أكبر . هذا سيتيح لنا قائمة من القوائم, و بمساعدة حلقة التكرار سيتم رسم الدوائر في الأماكن الصحيحة .

تمارين



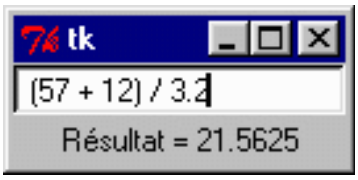
8.10 أستوحي بعض الأفكار من السكريبت السابق لكتابة برنامج لإظهار لوحة لعبة الداما (الرسم يتكون من مربعات سوداء و بيضاء) عندما نضغط على الزر

8.11 من برنامج التمرين السابق, أضف زر الذي سوف يظهر بيادق بشكل عشوائي في لوحة الداما(كل ضغطة على الزر سوف تظهر بيدق بشكل عشوائي) .

مثال غرافيكي : الة حاسبة بسيطة

على الرغم من أن الكود قصير للغاية, السكريبت الذي بالأسفل مثل الة حاسبة كاملة أي أنه يمكنك الحساب حتى مع الأقواس و الرموز العلمية . لا يوجد أي شيئ غير عادي . كل هذه الوضائف تستخدم مفسر بدل من مترجم لتنفيذ برامجك .

كما تعلم, فإن المترجم لا يأتي إلا لمرة واحدة, لترجمة الكود المصدري لبرنامجك إلى برنامج قابل للتنفيذ . أي أن دوره ينتهي قبل تنفيذ البرنامج . أما المفسر يبقى نشطا خلال تنفيذ البرنامج و بالتالي هو متوفر لترجمة أي كود مصدري جديد, مثل عبارة رياضية يتم إدخالها عن طريق لوحة المفاتيح من المستخدم .



أي أن لغات البرمجة التي تعتمد على المفسر, يكون مفسرها موجود دائما ليفحص سلسلة نصية مثل تعليمة من اللغة نفسها . يصبح من الممكن بناء بضعة أسطر برمجية من الهيكل البرامج ديناميكي للغاية . في المثال بالأسفل,

نحن إستخدمنا الدالة **eval()** لفحص التعبير الرياضي الذي تم إدخاله من قبل المستخدم, ثم نظهر نحن نتيجة .

```
# تمرين يستخدم مكتبة رسومية tkinter و وحدة math

from tkinter import *
from math import *

# تعريف الحركة التي يجب إتخاذها عندما يفعله المستخدم
# مفتاح الإدخال عة الذي يقوم بتحرير حقل الإدخال :

def evaluer(event):
    chaine.configure(text = "Résultat = " + str(eval(entree.get())))

# ----- : البرنامج الرئيسي -----

fenetre = Tk()
entree = Entry(fenetre)
entree.bind("<Return>", evaluer)
chaine = Label(fenetre)
entree.pack()
chaine.pack()

fenetre.mainloop()
```

في بداية السكريبت, بدأنا بإستدعاء الوحدتين tkinter و math, هذا الأخير ضروري في ألة الحاسبة من أجل توفير جميع الدالات الرياضية و العلمية المعتادة : sinus, cosinus, الجذر التربيعي إلخ ...

بعد ذلك نحن عرفنا الدالة **evaluer()**, و هو في الواقع أمر ينفذه البرنامج بعد أن يضغط المستخدم على زر الإدخال بعد أن يدخل التعليمة الرياضية في حقل الإدخال .

هذه الدالة تستخدم الأسلوب **configure()** لويدجت chaine[32], لتعديل سمة النص . السمة في السؤال يحصل إذا على قيمة جديدة, المحدد بما كتبناه على يمين علامة المساوات : موجود بها سلسلة نصية بنيت بشكل حيوي, بمساعدة دالتان مدمجتان في البايثون : **eval()** و **str()**, و مرتبطة بويدجت ل tkinter : الأسلوب **get()** .

يستخدم **eval()** لفحص تعبير البايثون الممرر إليه في سلسلة نصية , ناتج هذا الفحص في شكل "رجوع" (إرجاع قيمة) . على سبيل المثال :

32يمكن تطبيق الأسلوب **configure()** لأي ويدجت موجودة لتغيير خصائصها .

```
chaine = "(25 + 8)/3"     # سلسلة تحتوي على تعبير رياضي
res = eval(chaine)        # تقييم التعبير الموجود في السلسلة
print(res +5)             # => محتوي المتغير res رقمي
```

يستخدم **str()** لتحويل تعبير رقمي إلى سلسلة نصية . لقد قمنا بإستدعاء هذه الدالة لأن العائد السابق يقوم بإرجاع قيمة رقمية, و يجب علينا تحويلها إلى سلسلة نصية لنكون قادرين على دمجها مع رسالة **=** **Resultat** .

الأسلوب **get()** يرتبط مع الويدجات للصنف **Entry** . في برنامجنا الصغير (مثال), نحن إستخدمنا الويدجت من هذا النوع للسماح للمستخدم بإدخال أي عبارة رقمية بمساعدة لوحة مفاتيحه و يقوم الأسلوب **get()** بأخذ مدخلات المستخدم في الويدجت .

!!!!! نص البرنامج الرئيسي يحتوي على مرحلة التهيئة, التي تنتهي مع مستقبل الأحداث (**mainloop**) . و يوجد مثيل للنافذة **Tk()** , تحتوي على الويدجات **chaine** لتمثيل بداية من الصنف **Label()**, و الويدجت تدخل مثيل بداية من الصنف **Entry()** .

تحذير : الويدجت الأخيرة تقوم حقا بعملها, و هذا معناه نقل التعبير الذي أدخله المستخدم إلى البرنامج, و نحن ربطناه مع الحدث بمساعدة الأسلوب **bind()**[33] :

```
    entree.bind("<Return>",evaluer)
```

هذه التعليمة تعني : ربط الحدث - الضغط على زر الإدخال - مع الكائن - الإدخال -, و تعالجها الدالة **evaluer** .

تم وصف الحدث في سلسلة نصية معينة (في مثالنا, يقصد السلسلة "<Return>". و يوجد عدد كبير من هذه الأحداث(تحريك و نقرات الفأرة, ضغطات على لوحة المفاتيح, تحديد مواقع, و تغيير حجم النوافذ ...إلخ) . سوف تجدون قائمة سلاسل المحددة لكل هذه الأحداث في مراجع مكتبة tkinter .

لاحظ جيدا أنه لا يوجد أقواس بعد إسم الدالة **evaluer** . في الحقيقة : في هذه التعليمات, نحن لا نريد إستدعاء الدالة (هذا سيكون سابق لأوانه), ما نريده هو ربط نوع معين من الأحداث مع هذه الوظيفة, بطريقة لنستدعيها في وقت لاحقا, كلما يقع الحدث, إذا وضعنا داخل القوسين, البرامتر الذي سيتم تمريره لأسلوب **bind()** سيكون قيمة رجوع هذه الدالة و ليس مرجعها .

33كلمة bind معناها ربط .

سنغتنم هذه الفرصة لندرس عن تركيب التعليمات لتنفيذ أسلوب مرتبط بكائن :

كائن.الأسلوب(البرامترات)

نكتب أولا إسم الكائن الذي نريده ثم نقطة(التي تعمل بمثابة عامل), ثم إسم الأسلوب الذي نريد تنفيذه . و نضع في ما بين القوسين البرامترا التي تريد تمريرها .

مثال غرافيكي : كشف و تحديد مكان ضغطة زر الفأرة

في تعريف الدالة **evaluer** في المثال السابق, ربما لاحظت أننا قد مررنا برامتر الحدث(في ما بين القوسين) .

هذا البرامتر إلزامي [34]. عند تعريف دالة لمعالجة الأحداث مرتبطة بأي ويدجت بمساعدة الأسلوب **()bind**, و يجب عليك إستخدامه دائما كأنه البرامتر الأول . هذه البرامترات في الحقيقة كائن تم صنعها تلقائيا من قبل tkinter, و هو ينقل لمعالج الأحداث عدد من سمات الحدث :

* نوع الحدث : تحريك الفأرة, الضغط على أحد أزرارها, الضغط على زر من لوحة المفاتيح, وضع المؤشر في مكان محدد, فتح أو إغلاق نافذة, إلخ ...

* مجموعة من خصائص الحدث : لحظة وقعها, و خصائص الويدجت أو الويدجات إلخ ...

لن ندخل في تفاصيل أكثر من ذلك, إذا جربت السكريبت الذي بالأسفل, سوف تفهم بسرعة الفكرة .

```
# كشف و تحديد موقع نقرة الفأرة في النافذة

from tkinter import *

def pointeur(event):
    chaine.configure(text = "Clic détecté en X =" + str(event.x) +\
                            ", Y =" + str(event.y))

fen = Tk()
cadre = Frame(fen, width =200, height =150, bg="light yellow")
cadre.bind("<Button-1>", pointeur)
cadre.pack()
chaine = Label(fen)
chaine.pack()

fen.mainloop()
```

34البرامتر إلزامي, لكن إسم **event** هو مجرد تطبيق لإتفاقية, يمكنك إستخدام أي إسم, لكن هذا غير مستحسن .

السكريبت يعرض نافذة تحتوي على لوحة (**Frame**) صفراء مستطيلة الشكل, و تطلب من المستخدم النقر عليها .

الأسلوب **bind()** للويدجت من نوع لوحة و يرتبط الحدث "الضغط بالزر الأول للفأرة" بمعالج الحدث "المؤشر".

هذا معالج الأحداث يستطيع إستخدام السمات **x** و **y** للكائن **event** الذي أنشئ تلقائيا بواسطة tkinter , ثم لصنع سلسلة نصية التي تعرض موقع الفأرة في لحظة النقر .

تمرين

8.11. عدل البرنامج النصي في الأعلى لإظهار دائرة حمراء صغيرة في مكان الذي نقر عليه المستخدم !!!! (سوف تستبدل أول الويدجت اللوحة بويدجت **Canvas**)

أصناف الويدجت tkinter

> ملاحظة
>
> في هذا الكتاب, سوف نعرض لكم تدريجيا إستخدام عدد من الويدجات . و لكن ليس في نيتنا تقديم دليل مرجعي كامل ل ***tkinter*** . و نحن نتقيد هنا فقط بتفسير الويدجات التي تبدو الأكثر إثارة للإهتمام للتعلم الشخصي, و هذا معناه أن نسلط الضوء على المفاهيم البرمجية المهمة, مثل الصنف و الكائن . و يرجي الرجوع إلى (أنظر للصفحة :) إذا أردت المزيد من التفصيل .

هنالك 15 صنف أساسي للويدجت tkinter :

الوصف	الويدجت
زر كلاسيكي, يستخدم لتنفيذ أي أمر .	Button
مساحة لوضع مختلف العناصر الرسومية . هذا الويدجت تستطيع إستخدامه للرسم,	Canvas

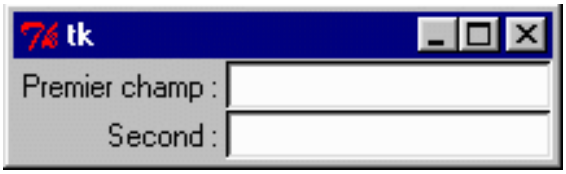
الويدجت	الوصف
	صنع و تعديل على رسوم, و أيضا لتطبيق ويدجات خاصة .
Checkbutton	خانة للإختيار إحدى الإختيارين(من تحديد المربع أو لا) عند الضغط على هذا الويدجت سيتغير الإختيار .
Entry	حقل للمدخلات, و الذي من خلال يستطيع المستخدم أن يدخل للبرنامج أي ن من لوحة المفاتيح .
Frame	سطح مستطيل الشكل في النافذة, أين نضع الويدجات الأخرى. هذا السطح يمكن تلوينه . و يمكن أيضا أن تضع له إطار(تزين حوافه).
Label	أي نص (أو حتى صورة) .
Listbox	قائمة إختيارات تقدم للمستخدم, عادة ما تقدم في نوع من العلب . و يمكننا ضبطه بطريقة بحيث يصبح مثل "أزرار راديو" أو خانات.
Menu	قائمة. قد تكون قائمة منسدلة يتم وضعها في شريط العنوان, أو في قائمة "منبثقة" - "pop up" و هي تظهر في أي مكان بعد الضغط بزر الفأرة .
Menubutton	قزر للقائمة, تستخدم بتشغيل قائمة المنسدلة .
Message	تعرض نصا . هذا الويدجت هو بديل ويدجت ملّصق(Label), يتكيف تلقائيا حسب النص المعروض إلى حجم معين أو إلى عرض\إرتفاع معين .
Radiobutton	يعّرف أنه (نقطة سوداء في دائرة صغيرة) واحد من القيّم لمتغير الذي قد يمتلك أكثر من قيمة . عندما تضغط على زر الراديو يمرر قيمة الزر إلى المتغير, و يمرر فارغ لجميع الأزرار الأخرى لنفس المتغير .
Scale	يسمح لك بتغيير قيمة متغير بطريقة مرئية عن طريق تحريك المؤشر على طول المسطرة .
Scrollbar	تستطيع إستخدامه مع العديد من الحاجيات : لوحة, نص, قائمة مربعات ...إلخ
Text	يستخدم لعرض نص منسق . كما يسمح للمستخدم تعدل(تحرير) النص المعروض . و يمكن إدراج صور أيضا .
Toplevel	نافذة عرض منفصلة, تظهر عند البداية .

هذه الأصناف لويدجات تحتوى كل واحد منهم عدد كبير من الأساليب . و يمكننا أيضا ربطها بالأحداث, كما رأينا في الصفحات السابقة . و سوف نتعلم كيفية وضع كل هذه الحاجيات في النوافذ بإستخدام ثلاثة أساليب مختلفة : الأسلوب **grid()** , و الأسلوب **pack()** و الأسلوب **place()** .

الفائدة من إستخدام هذه الأساليب هو أن نجعل هذه البرامج محمولة (و هذا معناه أن تعمل جيدا في جميع أنظمة التشغيل المختلفة مثل يونكس أو ماك أو ويندوز), و يمكن تغيير حجم نوافذها .

إستخدام الأسلوب grid للتحكم في أماكن الويدجات

حتى الأن, نحن نستخدم دائما لوضع الويدجات على النوافذ الأسلوب **pack()** . هذا الأسلوب تتميز بأنها بسيطة للغاية, لكن لكنها لا تعطينا الحرية الكامل في وضع الويدجات كما يحلو لنا . ماذا نفعل, على سبيل المثال, للحصول على نافذة مثل التي بالأسفل ؟



يمكننا أن نقوم بعدد من المحاوات لتجربة إستخدام الأسلوب **pack()** مع البرامترات نوع "**side=**", مثل التي قمنا بفعلها سابقا, لكن هذه الطريقة لا تفيدنا كثيرا . مثلا لو كتبت :

```
from tkinter import *

fen1 = Tk()
txt1 = Label(fen1, text = 'Premier champ :')
txt2 = Label(fen1, text = 'Second :')
entr1 = Entry(fen1)
entr2 = Entry(fen1)
txt1.pack(side =LEFT)
txt2.pack(side =LEFT)
entr1.pack(side =RIGHT)
entr2.pack(side =RIGHT)

fen1.mainloop()
```

ستكون النتيجة ليس ما كنا نريده !

لتفهم بشكل أفضل الأسلوب **pack()** , يمكنك أن تجرب العديد كم الخيارات, مثل **side=TOP** و **side=BOTTOM** لكل واحدة من الويدجات الأربعة . لكنك بالتأكيد لن تحصل على ما أردناه هنا . يمكنك فعل هذا من خلال تعريف 2 ويدجات منوع إطار **Frame()** إضافية, و ثم ستقوم بدمج الويدجات **()** **Label** و **Entry()** . و سيكون هذا معقد للغاية .

لقد حان الوقت لتعلم إستخدام أسلوب جديد لحل هذه المشكلة . يرجى منك الأن تحليل السكريبت بالأسفل : يحتوي (تقريبا) على الحل :

```
from tkinter import *

fen1 = Tk()
txt1 = Label(fen1, text = 'Premier champ :')
txt2 = Label(fen1, text = 'Second :')
entr1 = Entry(fen1)
entr2 = Entry(fen1)
txt1.grid(row =0)
txt2.grid(row =1)
entr1.grid(row =0, column =1)
entr2.grid(row =1, column =1)
fen1.mainloop()
```

في هذا السكريبت, لقد قمنا بإستبدال الأسلوب **pack()** بالأسلوب **grid()** . كما ترون, إن إستخدام الأسلوب **grid()** بسيط للغاية. هذذا الأسلوب يعتبر النافذة كأنها جدول (أو شبكة) . ثم ستكتفي أنت بالإشارة إلى الصف و العمود من الجدول الذي تريد وضع به الويدجت . يمكنك ترقيم الأعمدة و الصفوف كما تريد , إبتداء من أي رقم, مثلا 0 أو 1 أو 2 ...إلخ : يقوم tkinter بتجاخل الصفوف و الأعمدة الفارغة . إذا لم تضع أي رقم لسطر أو لعمود, ستكون القيمة الإفتراضية 0 .

يقوم tkinter تلقائيا بتحديد عدد الصفوف و الأعمدة اللازمة . و لكن ليس هذا فقط : فإذا حللت النافذة الصغيرة الذي أنتجها السكريبت الذي بالأعلى, ستجد أننا لم نصل لهدفنا بعد . السلسلتين في الجزء الأيسر من النافذة موجودتان في الوسط, إذا يجب علينا أن تكونا على اليمين . لتحقيق هذا, يكفي أن نضيف برامتر عند إستدعاء الأسلوب **grid()** لهذه الويدجات . الخيار **sticky** يمكن أن يأخذ واحد من هذه الأربعة قيم : **N** و **S** و **W** و **E** (الإتجاهات الأربعة باللغة الأنكليزية) . على أساس هذه القيم, ستكون محاذات الويدجات أعلى أو أسفل, أو على اليمين أو على اليسار . سنستبدل إذا السطرين الأولين لتعليمات **grid()** في السكريبت ب :

```
txt1.grid(row =0, sticky =E)
txt2.grid(row =1, sticky =E)
```

و أخير سنحصل على ما نريده.

حلل الأن النافذة التالية :

هذه النافذة تتكون من ثلاثة أعمدة : الأولى تتكون من 3 سلاسلة نصية, الثانية تتكون من 3 حقول للإدخال, و أما الثالثة فتتكون من صورة . أول عمودين يتكونان من ثلاثة صفوف, لكن الصورة التي تقع في العمود الثالث تنتشر على ثلاثة صفوف .

كود هذه النافذة :

```
from tkinter import *

fen1 = Tk()

# : صنع ويدجي "ملصق" و "مدخل"
txt1 = Label(fen1, text ='Premier champ :')
txt2 = Label(fen1, text ='Second :')
txt3 = Label(fen1, text ='Troisième :')
entr1 = Entry(fen1)
entr2 = Entry(fen1)
entr3 = Entry(fen1)

# : صنع ويدجت "لوحة" تحتوي على صورة نقطية
can1 = Canvas(fen1, width =160, height =160, bg ='white')
photo = PhotoImage(file ='martin_p.gif')
item = can1.create_image(80, 80, image =photo)

# : تنسيق بإستخدام الأسلوب "grid"
txt1.grid(row =1, sticky =E)
txt2.grid(row =2, sticky =E)
txt3.grid(row =3, sticky =E)
entr1.grid(row =1, column =2)
entr2.grid(row =2, column =2)
entr3.grid(row =3, column =2)
can1.grid(row =1, column =3, rowspan =3, padx =10, pady =5)

# : بدء التشغيل
fen1.mainloop()
```

لتشغيل هذا السكريبت, يجب عليك أول أن تغير إسم ملف الصورة **martin_p.gif** بإسم الصورة التي تريدها و يجب أن تكون بنفس مكان السكريبت . إنتبه : مكتبة tkinter القياسية لا تقبل سوى عدد قليل من أنواع الصور . من هذه الأنواع GIF[35].

يمكننا أن نلاحظ بعض الأشياء في السكريبت :1. التقنية المستخدمة لتضمين الصورة :

[35] الأنواع الأخرى من الرسوم ممكنة, لكنها تتطلب وحدات رسومية لمكتبة PIL (Python Imaging Library) و التي هي إمتداد للبايثون متاحى على : http://www.pythonware.com/products/pil/ . و هذه المكتبة تسمح لك بأداء العديد من المعالجات على الصور, و لكن دراسة هذه التقنيات خارج إطار دورتنا .

إن tkinter لا تسمح لك بتضمين الصور مباشرة في النافذة . يجب عليك أولا وضع لوحة (canevas), ثم نضع الصورة في اللوحة . نحن إخترنا اللوحة بلون أبيض, لكي نميزها عن النافذة . يمكنك إستبدال البرامتر **bg=white** ب **bg=gray** إذا أردت أن تصبح اللوحة غير مرئية . بما أن يوجد العديد من أنواع الصور, يجب عينا أن نعرف كائن الصورة على أنه صورة نقطية GIF, و ذلك عن طريق الصنف **PhotoImage()**.

2. سطر الذي وضعنا الصورة في اللوحة :

```
item = can1.create_image(80, 80, image =photo)
```

لإستخدام طريقة صحيحة, نحن ننصح هنا بإستخدام الأسلوب **create_image()** مرتبطة بالكائن can1 (و الذي هو كائن مثيل للصنف لوحة (Canvas) . أول برامترين يمررون (80,80) و هي إحداثيات x و y للوحة حيث يتم وضعها في المنتصف . إن أبعاد اللوحة هي 160 × 160, و إن إختيارنا سيؤدي إلى وضع الصورة في منتصف اللوحة .

3. طريقة ترقيم الصفوف و الأعمدة في أسلوب **grid()** :

يمكنك أن ترى أن ترقيم الأعمدة و الصفوف في أسلوب **grid()** يبدأ من الرقم 1(و ليس 0 كما في السكريبت السابق) . كما قلنا سابقا أن الترقيم حر (أي تبدأ بأي رقم تريده) .

يمكننا إختيار أي رقم مثلا : 5,10,15,20... لأن tkinter يقوم بتجاهل كل الصفوف و الأعمدة الفارغة . الترقيم من الرقم 1 سيزيد من سهولة قراءة الكود .

4. البرامترات المستخدمة مع **grid()** لوضح اللوحة (Canvas) :

```
can1.grid(row =1, column =3, rowspan =3, padx =10, pady =5)
```

أول برامترين تشير إلى أن اللوحة (Canvas) سوف يتم وضعها في الصف الأول للعامود الثالث . و البرامتر الثالث (**rowspan = 3**) يشير إلى أنه سيتم نشره على ثلاثة صفوف . و أما عن ×ر برامترين (**padx = 10, pady = 5**) تشير إلى أبعاد الفراغ حول الويدجت (الطول و العرض) .

5. بما أننا كتبنا الكود و إستفدنا من هذا السكريبت كمثال, سوف نقوم الأن بتبسيطه قليلا .

تركيب تعليمات لكتابة كود أكثر إيجازا

البايثون من لغات البرمجة عالية المستوى, غالبا يكون من الممكن (و مرغوب فيه) إعادة صياغة السكريبت لجعله أكثر إيجازا .

الكود سيصبح أكثر بساطة, و سيكون في الغالب أكثر قابلية للقراءة . على سبيل المثال تستطيع إستبدال السطرين التالين من السكريبت السابق :

```
txt1 = Label(fen1, text ='Premier champ :')
txt1.grid(row =1, sticky =E)
```

بسطر واحد :

```
Label(fen1, text ='Premier champ :').grid(row =1, sticky =E)
```

في هذا السطر الجديد, يمكنك أن ترى أننا إختصرنا كتابة المتغير txt1 و لقد وضعنا المتغير لكي نعيد إستخدامه في أماكن أخرى, و لكن هذا ليس ضروريا, ببساطة لقد قمنا بإستدعاء الصنف Label() الذي يؤدي إلى صنع مثيل لكائن من هذا الصنف . حتى لو لم نخزن مرجع هذا الكائن في متغير (tkinter يحفظها على أيت حال في التمثيل الداخلي للنافذة) . فإذا تم ذلك, يتم فقدان المرجع لبقية سكريبت البايثون, لكن يمكن أن يتم نقله إلى أسلوب مثل **grid()** في لحظة تمثيله, في عبارة مركبة واحدة . سوف نرى ذلك بأكثر تفاصيل .

حتى الأن, أنشأنا العديد من الكائنات(بواسطة تمثيل بداية من أي صنف), و التي عيناها في كل مرة إلى المتغيرات . على سبيل المثال, عندما نكتب :

```
txt1 = Label(fen1, text ='Premier champ :')
```

نكون قد صنعنا مثيل من صنف **Label()**, و لقد عيناها إلى المتغير **txt1** .

و يمكن بعد ذلك إستخدام المتغير **txt1** للإشارة (مرجع) لهذا المثيل, في مكان أخر لهذا السكريبت, لكننا في الحقيقة لا نستخدمه سوى لمرة واحدة عندما نطبق الأسلوب **grid()**, الويدجت هي ليست سوى ملصق(**Label**) بسيط للوصف . و صنع متغير جديد ليكون مرجعا لمرة واحدة فقط(و مباشرةً بعد إنشاءه) ليست فكرة جيدة, لأمه سيحجز بعض المساحة بدون داع .

عندما تكون في هذه الحالة, فمن الأفضل أن تستخدم تعليمات التركيب . على سبيل المثال, يفضل في معظم الأحيان إستبدال التعليمتين التاليتين :

```
somme = 45 + 72
print (somme)
```

بتعليمة واحدة مركبة :

```
print (45 + 72)
```

و بالتالي وفرنا متغير .

و بنفس الطريقة, عندما نضع ويدجات التي لا نرغب في إستخدامها في وقت لاحق, كما في الويدجات من صنف **Label()**, و التي يمكنك أن تطبق عليها أسلوب (**grid()** أو **pack()** أو **place()**)نقوم مباشرة بصنع الويدجت في تعليمة مركبة واحدة .

يطبق هذا فقط للويدجات التي لن نشير(كمرجع) إليها بعد صنعها . و يجب على الباقي أن يتم تعيينهم إلى متغير, حتى نتمكن من التفاعل معهم في أماكن مختلف في السكريبت .

و في هذه الحالة, نحن ملزمين أن نستخدم تعليمتين منفصلتين, واحدة لتمثيل الويدجت, و الأخرى لتطبيق عليها الأسلوب . فلا يمكنك, على سبيل المثال, كتابة هذه التعليمة المركبة :

```
entree = Entry(fen1).pack()          # خطأ برمجي !!!
```

و يجب في هذه الحالة, أن نقوم بتمثيل الويدجت الجديد و إسناد ذلك إلى المتغير **entree**, ثم سوف يتم تخطيط الصفحة بمساعدة الأسلوب **pack()** . في الحقيقة, هذه العبارة تولد ويدجت جديد من صنف **Entry()**, و بأسلوب **pack()** التي تضعها في النافذة, لكن القيمة التي تم تخزينها في المتغير **entree** ليست مرجع للويدجت ! بل هو قيمة رجوع للأسلوب **pack()** : إذا كنت ت**إستخدام الأسلوب grid للتحكم في أماكن الويدجات**تذكر إن الأساليب مثل الدالات, ترجع دائما قيمة للبرنامج الذي إستدعاها . و أنت لا تستطيع أن تفعل شيئا مع قيمة الرجوع : لذا فهو في هذه الحالة كائن فارغ (لاشيئ).

و إذا أردت أن تحصل على مرجع حقيقي . يجب عليك إستخدام هتين التعليمتين :

```
entree = Entry(fen1)          # تمثيل الويدجت
entree.pack()                 # تطبيق التخطيط
```

> عندما تستخدم الأسلوب *grid*() , يمكنك ببساطة تبسيط التعليمات البرمجية قليلا, بحذف الإضارة إلى العديد من أرقام الصفوف و الأعمدة . من اللحظة التي تستخدم فيها الأسلوب *grid*() لوضع

> الويدجات, سيقوم *tkinter* بوضع الصفوف و الأعمدة الضرورية .[36]. فإذا كان رقم صف أو عموج غير موجود , سيتم وضع الويدجت في أول مربع فارغ متاح .

السكريبت الذي بالأسفل بدون التبسيط الذي شرحناه :

```
from tkinter import *
fen1 = Tk()

# صنع ويدجات Label() و Entry() و Checkbutton() :
Label(fen1, text = 'Premier champ :').grid(sticky =E)
Label(fen1, text = 'Deuxième :').grid(sticky =E)
Label(fen1, text = 'Troisième :').grid(sticky =E)
entr1 = Entry(fen1)
entr2 = Entry(fen1)                # من المؤكد أن يتم إستخدام مرجع هذه الويدجات لاحقا
entr3 = Entry(fen1)
entr1.grid(row =0, column =1)      # لذلك يجب أن يتم تعيين كل واحدة في متغير مستقل
entr2.grid(row =1, column =1)
entr3.grid(row =2, column =1)
chek1 = Checkbutton(fen1, text ='Case à cocher, pour voir')
chek1.grid(columnspan =2)

# صنع ويدجت "لوحة" يحتوي على صورة نقطية :
can1 = Canvas(fen1, width =160, height =160, bg ='white')
photo = PhotoImage(file ='Martin_P.gif')
can1.create_image(80,80, image =photo)
can1.grid(row =0, column =2, rowspan =4, padx =10, pady =5)

# البداية:
fen1.mainloop()
```

تغيير (تحرير) خصائص كائن - الرسوم المتحركة

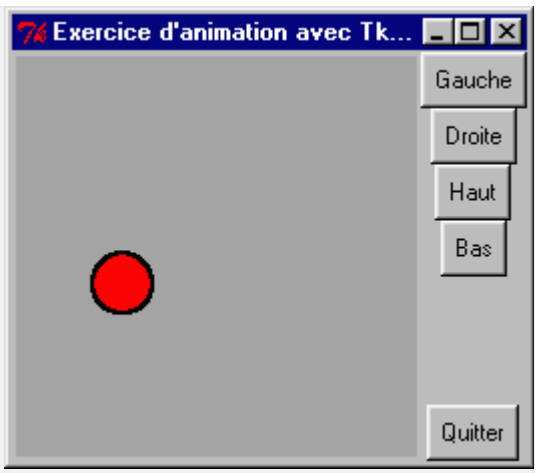
في هذه المرحلة من التعليمك, ربما تريد أن تظهر رسم صغير في اللوحة, ثم يقوم بالتحرك, على سبيل المثال بمساعدة الأزرار .

يجب عليك إذا كتابة, و تجربة و تحليل السكريبت الذي بالأسفل :

```
from tkinter import *

# الإجراء العام للحركة :
def avance(gd, hb):
    global x1, y1
```

36لا تستخدم عدة أساليب لتحديد أماكن متعددة في نفس النافذة ! الأساليب **grid()** و **pack()** و **place()** لا يمكن جمعهم .




```
    x1, y1 = x1 +gd, y1 +hb
    can1.coords(oval1, x1,y1, x1+30,y1+30)

# معالج الأحداث
def depl_gauche():
    avance(-10, 0)

def depl_droite():
    avance(10, 0)

def depl_haut():
    avance(0, -10)

def depl_bas():
    avance(0, 10)

#------ البرنامج الرئيسي -------

# المتغيرات التالية سيتم إستخدامهم كمتغيرات عامة
x1, y1 = 10, 10   # coordonnées initiales

# صنع الويدجت الرئيسي (السد)
fen1 = Tk()
fen1.title("Exercice d'animation avec tkinter")

# صنع ويدجات « العبيد »
can1 = Canvas(fen1,bg='dark grey',height=300,width=300)
oval1 = can1.create_oval(x1,y1,x1+30,y1+30,width=2,fill='red')
can1.pack(side=LEFT)
Button(fen1,text='Quitter',command=fen1.quit).pack(side=BOTTOM)
Button(fen1,text='Gauche',command=depl_gauche).pack()
Button(fen1,text='Droite',command=depl_droite).pack()
Button(fen1,text='Haut',command=depl_haut).pack()
Button(fen1,text='Bas',command=depl_bas).pack()

# بدء متلقي الأحداث (حلقة الأساسية)
fen1.mainloop()
```

جسم(محتوى) هذا البرنامج يحتوي على العديد من العناصر التي عرفها : لقد قمنا بإنشاء النافذة **fen1**, و لقد صنعنا في داخلها لوحة تحتوي على دائرة ملونة, بالإضافة إلى 5 أزرار للتحكم . لاحظ أننا لم نصنع مثيل للويدجات من نوع أزرار في متغيرات(لأننا لن نسير إليها لاحقا) : لذا يجب علينا تطبيق الأسلوب **pack()** مباشرة في اللحظة التي نصنع فيها هذه الكائنات .

الشيئ الجديد في هذا البرنامج هو الدالة **avance()** التي تم تعريفها في بداية السكريبت . في كل مرة يتم إستدعائها, تقوم هذه الدالة بإعادة تحديد إحداثيات كائن "الدائرة الملونة" التي تم وضعها في للوحة, مما يتسبب في تحريك هذا الكائن .

هذه الطريقة تتميز بها البرمجة الشيئية, التي تبدأ من صنع الكائنات ثم يتم العمل على هذه الكائنات من خلال تغيير خصائصها, من خلال الأساليب .

في البرمجة الحتمية "القديمة"(و هذا يعني بدون إستخدام الكائنات). يتم تحريك الأرقام عن طريق حذفها ثم إعادة رسمها في مكان أبعد قليلا . أما في البرمجة الشيئية, يتم التعامل مع هذه المهام تلقائيا من قبل الفئات التي يتم إشتقاق منها الكائنات, حتى لا يتم تضييع الوقط في إعادة برمجة.

التمارين

8.12 كتب برنامج الذي يظهر نافذة بها لوحة . في هذه اللوحة يجب أن تكون بها دائرتين (بلونين و حجمين مختلفين), و التي من المفترض أن يمثل كوكبين . و أزرار تسمح لنقلهم إلى volonté كليهما إلى جميع الإتجاهات . تحت اللوحة, يجب على البرنامج أن يظهر دائما أ)المسافة بين الكوكبين, ب) قوة الجاذبية التي يقوم بها كل واحد ضد الأخر(تستطيع أن تظهر في أعلى نافذة الكتل المختارة لكل واحد منهما, و مسافة النطاق). في هذا التمرين, يجب عليكم إستخدام قانون نيوتن للجاذبية (التمرين 6.16, ص 58, و دليل الفيزيائي العام).

8.13 إستوحي من برنامج الذي يكتشف نقرات الفأرة في اللوحة, عدل البرنامج المذكور أعلاه و ذلك للحد من عدد الأزرار : لوضع الكوكب, يتم ببساطة بإختيار زر, ثم يتم الضغط على اللوحة ليتم وضع الكوكب في المكان الذي يتم النقرة عليه .

8.14 إمتدادا للبرنامج أعلاه . ضع كوكب أخر, و أعرض القوة المؤثرة على الثلاثة (في الحقيقة: كل واحد و في جميع الأوقات قوة الجاذبية التي يبذلها من إثنين أخرين) .

8.15 نفس التمرين مع الشحنات الكهربائية(قانون كولوم) . أعطي هذه المرة لإختيار الشحنات .

8.16 أكتب برنامج صغير الذي يظر نافذة مع حقلين : الأول يشير إلى درجة الحراة المئوية, و الأخر درجة فهرنهايت . في كل مرة يتم تغيير أي واحدة من الدرجات الحرارة, يتم تصحيح الأخرى وفقا لذلك .

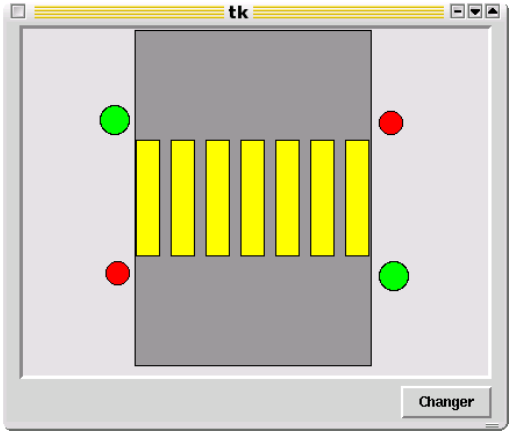
لتحويل الفهرنهايت إلى درجة مئوية, و العكس بالعكس, يتم بذلك بإستخدام المعادلة $T_F = T_C \times 1{,}80 + 32$. يمكنك أيضا مراجعة برنامج الة الحاسبة البسيطة (صفحة 90) .

8.17 أكتب برنامج الذي يظهر نافذة بها لوحة . في هذه اللوحة, ضع دائر صغيرة التي من المفترض أن تكون كرة . تحت اللوحة, ضع زر . في كل مرة يتم الضغط على الزر, تتقدم الكرة مسافة قصير إلى اليمين حتى تصل إلى نهاية اللوحة . فإذا لازلت تضغط على الزر, سوف ترجع الكرة إلى الطرف الأخر, و هكذا .

8.18 حسن البرنامج أعلاه لكل تتحرك الكرة بشكل دائري أو بيضوي في اللوحة (عند النقر مرارا و تكرارا) . ملاحظة : للحصول على النتيجة المطلوبة, سوف تقوم بالضرورة بتعريف متغير لتمثيل الزاوية, و إستخدام الدالتين sinus و cosinus لوضع الكرة لوفقا لهذه الزاوية .

8.19 عدل البرنامج الذي بالأعلى بطريقة تجعل الكرة, عندما تتحرك تترك ورائها أثر من المسار .

8.20 عدل البرنامج المذكور أعلاه بطريقة لتوجيه أرقام أخرى . إستشر أستاذك للحصول على إقتراحات (أرقام Lissajous) .



8.21 أكتب برنامج الذي يظهر نافذة مع لوحة و زر . في اللوحة, أرسم مستطيل رمادي غامق, و الذي يمثل الطريق, ثم قم برسم سلسلة مستطيلات صفراء التي تمثل ممر لعبور المشاة . أضف أربعة دوائر ملونة التي تشير إلى إشارة المرور للمشاة و السيارات . و مع كل ضغطة على الزر سوف يتغير لون الأضواء .

8.22 أكتب البرنامج الذي يظهر لوحة مرسوم عليها دارة كهربائية بسيطة(مولد + مبدل + مقاوم) . و يجب أن يكون في النافذة حقول إدخال لإدخال برامتر كل عنصر . (و هذا معناه تحديد قيم المقاومة و الفولتية) . و يجب أن يكون المبدل يعمل(بزر إعمل\توقف). بالإضافة لملصقات(لابل - Label) التي يجب أن تعرض دائما الفولتية و التيارات الناجمة عن الخيارات التي قام بها المستخدم .

رسوم متحركة تلقائية

وفي الختامو هذه أول مرة نتصل مع واجهة الرسومية tkinter, هذا هو أخر مثال لرسوم متحركة, و الذي يعمل بشكل مستقل عند الضغط على "أعمل" .

```
from tkinter import *

# تعريف متلقي الأحداث

def move():
    "déplacement de la balle"
    global x1, y1, dx, dy, flag
    x1, y1 = x1 +dx, y1 + dy
    if x1 >210:
        x1, dx, dy = 210, 0, 15
    if y1 >210:
        y1, dx, dy = 210, -15, 0
    if x1 <10:
        x1, dx, dy = 10, 0, -15
    if y1 <10:
        y1, dx, dy = 10, 15, 0
    can1.coords(oval1,x1,y1,x1+30,y1+30)
    if flag >0:
        fen1.after(50,move)          # => ضعه في الحلقة بعد 50 ميلي ثانية

def stop_it():
    "arrêt de l'animation"
    global flag
    flag =0

def start_it():
    "démarrage de l'animation"
    global flag
    if flag ==0:              # لكي لا يتم تشغيل سوى حلقة واحدة
       flag =1
       move()

#========== البرنامج الرئيسي ==========

# سيتم إستخدام هذه المتغيرات كمتغيرات عامة
x1, y1 = 10, 10               # الإحداثيات الأولية
dx, dy = 15, 0                # خطوة الإزاحة
flag =0                       # العداد

# صنع ويدجت الرئيسي الأصل
fen1 = Tk()
fen1.title("Exercice d'animation avec tkinter")
# صنع ويدجت الأطفال
can1 = Canvas(fen1,bg='dark grey',height=250, width=250)
```

```
can1.pack(side=LEFT, padx =5, pady =5)
oval1 = can1.create_oval(x1, y1, x1+30, y1+30, width=2, fill='red')
bou1 = Button(fen1,text='Quitter', width =8, command=fen1.quit)
bou1.pack(side=BOTTOM)
bou2 = Button(fen1, text='Démarrer', width =8, command=start_it)
bou2.pack()
bou3 = Button(fen1, text='Arrêter', width =8, command=stop_it)
bou3.pack()
# بدء متلقي الأحداث (الحلقة الأساسية)
fen1.mainloop()
```

الشيئ الجديد في هذا السكريبت يقع عند نهاية تعريف الدالة **move()** : هل لاحظت إستخدام الأسلوب **after()** . يمكن تطبيق هذا الأسلوب على أي ويدجت . هذا يتسبب في إستدعاء الدالة بعد فترة زمنية معينة . على سبيل المثال :

window.after(200,qqc)

يقوم الويدجت **window** بإستدعاء الدالة **qqc()** بعد توقف دام 200 ميلي ثانية .

في السكريبت الخاص بنا, الدالة التي تم إستدعاءها من قبل الأسلوب **after()** هي الدالة **move()** . نحن إستخدمنا هنا للمرة الأولى تقنية برمجة قوية جدا و تدعى (!!!!! إستدعاء الذاتي - récursivité) . لجعله بسيط, نقول إن الإستدعاء الذاتي هو ما يحدث عندما تستدعي الدالة نفسها . و بالطبع سوف نحصل على حلقة التي يمكن أن تستمر إلى لانهاية إذا لم تضع طريقة لوقفها .

دعونا نرى كيف يعمل في مثالنا .

يتم إستدعاء الدالة **move()** للمرة الأولى عندما يتم النقر على زر البدء (Démarrer). ستبدأ عملها (و هذا معناه وضع الكرة). ثم, من خلال الأسلوب **after()**, التي يتم إستدعائها بعد فاصل قصير . ثم تبدأ الدورة الثانية, السوف تستدعي نفسها مرة أخرى, و هكذا إلى أجل غير مسمى .

هذا على الأقل ما سيحدث إذا لم نتخذ الإحتياطات اللازمة لوضع تعليمة الإخراج في مكان ما . في هذه الحالة, هذا إختبار شرطي بسيط : في كل حلقة, نحن نفحص محتويات المتغير **flag** بمساعدة العبارة **if** . فإذا كان محتوى المتغير **flag 0**, سوف تتوقف الحلة و يتوقف تحريك الرسوم . لاحظ أننا حرصنا على تعريف المتغير **flag** على أنه متغير عام . و بالتالي يمكننا تغيير قيمته بسهولة بمساعدة دالات أخرى, مثل التي مرتبطة بأزرار بدء و إيقاف .

حصلنا على ألية بسيطة لتشغيل و إيقاف الرسوم المتحركة : عند الضغطة الأولى على زر البدء سيتم تعيين قيمة ليست لصفر للمتغير **flag** , ثم سيتم إستدعاء لأول مرة الدالة **move()** . التي ستعمل, و ستستدعي نفسها كل 50 ميلي ثانية, إلى أن يكون قيمة المتغير **flag** صفر . فإذا أنت مستمر بالضغط على زر البدء , لن يتم إستدعاء الدالة **move()**, لأن قيمة المتغير **flag** هي 1 . و لذلك سوف تبدأ حلقة مرات عديدة .

الزر توقف (Arrêter) يعين للمتغير **flag** القيمة صفر, و تتوقف الحلقة .

تمارين

8.23 في الدالة **start_it()**, إحذف التعليمة :**if flag == 0** (و السطرين التاليين الذين يبدأن بمسافة البادئة) . ماذا حدث ؟ (إضغط مرات عديدة على زر البدء . حاول أن تتكلم بأكبر قدر من الوضوح لتفسيركم لماذا حدث .

8.24 عدل البرنامج بحيث يتغير لون الكرة في كل دورة .

8.25 عدل البرنامج بحيث تجعل حركات الكرة منحرفة كما كرة البلياردو (متعرج) .

8.26 عدل البرنامج لتحصل على حركات أخرى . على سبيل المثال حركة دائرية(كما في التمرين صفحة 102) .

8.27 عدل البرنامج, أو أكتبه واحد مشابه له, لمحاكاة سقوط الكرة (بسبب الجاذبية) و إرتدادها . تنبيه : هذه المرة يجب أن تكون الحركة متسارعة (تزداد السرعة مع الوقت) .

8.28 بداية من السكريبتات المذكورة أعلاه, يمكنك الأن كتابة لعبة تعمل على النحو التالي : كرة تتحرك بشكل عشوائي على اللوحة, بسرعة بطيئة . يجب على اللاعب الضغط على الكرة بإستخدام الفأرة . فإذا نجح, يحصل على نقطة, لكن الكرة تزداد سرعتها, و هكذا . أوقف اللعبة بعد عدد معين من النفرات و قم بعرض النتيجة .

8.29 غير من السكريبت السابق : في كل مرة ينجح فيها اللاعب في القبض على الكرة يصبح حجمها أصغر (يمكن أيضا تغيير لونها).

8.30 أكتب البرنامج الذي به العديد من المرات مختلفة الألون, و التي تقفز في كل مكان, و على الجدران .

8.31 أصنع لعبة المثالية من خلال السكريبتات السابقة, من خلال دمج الخوارزميات أعلاه . يجب على اللاعب الضغط فقط على الكرات الحمراء . و إذا نقر بالخطأ على كرة من لون أخر يفقد بضعة نقاط .

8.32 أكتب البرنامج الذي يحاكي أثنين من كواكب التي تدور حول الشمس في مدارات دائرية مختلفة (أو إثنين من الألكترنات التي تدور حول نواة الئرة ...).

8.33 أكتب برنامج للعبة الثعبان : ثعبان (يتكون من خط قصير من المربعات) يتحرك على اللوحة في أربعة إتجاهات : يسار و يمين و أعلى و أسفل . يمكن للاعب تغيير إتجاه الثعبان من خلال الأسهم في لوحة المفاتيح . و في القماش يوجد أيضا "الفريسة" (دوائر صغيرة مرتية عشوائيا) . يجب على الثعبان أن يأكل الفريسة دون أن يصطدم مع حواف اللوحة . في كل مرة يأكل فيها فريسة يزداد اثعبان طولا, و يربح اللاعب نقطة , و تظهر الفريسة جديدة في مكان أخر . اللعبة تتوقف عندما يلمس الثعبان أحد الجدران أو عندما يصل إلى طول محدد .

8.32طور اللعبة السابقة بإضافة : تتوقف اللعبة إذا تداخل الثعبان .

9

التعامل مع الملفات

حتى الأن, جميع البرامج التي صنعناها لا تتعامل سوى مع كمية صغيرة جدا من البيانات . و في كل مرة نريد أن نتعامل مع هذه البيانات نضعها في جسم البرنامج نفسه(على سبيل المثال في قائمة) . هذه الطريقة غير كافية إلى حد بعيد عندما نريد التعامل مع كمية أكبر و مهمة من المعلومات .

فائدة الملفات

لنفترض على سبيل المثال أننا نريد كتابة برنامج صغير الذي يظهر على الشاشة أسئلة متعددة الخيارات, مع معالجة تلقائية لردود المستخدم . كيف يمكننا تخزين نص الأسئلة ؟

أبسط فكرة هي وضع كل نص في متغير, في بداية البرنامج, مع عبارات التعيين, مثلا :

```
a = "Quelle est la capitale du Guatemala ?"
b = "Qui à succédé à Henri IV ?"
c = "Combien font 26 x 43 ?"
        ... etc.
```

للأسف هذه الفكرة بسيطة جدا . و سيصبح بقية البرنامج معقدا جدا, هذا معناه أن التعليمات التي سيتم إستخدامها لإختيار بشكل عشوائي سؤال أو أكثر ليتم تقديمه للمستخدم . سوف تستخ\م على سبيل المثال مجموعة طويلة من عبارات **if....elif...elif** كما في المثال في الأسفل و هو بالتأكيد ليس الحل الجيد (و سيكون متعبا جدا في الكتابة, لا تنسى أيضا كتابة معالجة (إجابة) كل هذه الأسئلة!) :

```
if choix == 1:
    selection = a
elif choix == 2:
    selection = b
elif choix == 3:
```

```
    selection = c
        ... etc.
```

سيكون أفضل إذا إستخدمنا القوائم :

```
liste = ["Qui a vaincu Napoléon à Waterloo ?",
         "Comment traduit-on 'informatique' en anglais ?",
         "Quelle est la formule chimique du méthane ?", ... etc ...]
```

يمكن للمرء أن يستخرج أي عنصر من هذه القائمة بإستخدام المؤشر . على سبيل المثال :

```
print(liste[2])        ===>  "Quelle est la formule chimique du méthane ?"
```

> تذكير
>
> العداد يبدء من الصفر

في حين أن هذه الطريقة أفضل بكثير من الطريقة السابقة, لكن مازلنا نواجه العديد من المشاكل المزعجة :

* إن قابلية قراءة هذا البرنامج تتدهور بسرعة كبيرة عندما يصبح عدد الأسئلة كبير جدا . و طبعا, سوف نزيد من إحتمالية إدراج خطأ في تعريف هذه القائمة الطويلة . بعض الأخطاء قد يصعب جدا العثور عليها .

* إضافة أسئلة جديدة, أو تعديل على الأسئلة الموجودة , يجب علينا في كل مرة فتح كورد سورس البرنامج . و طبعا سيصعب إعادة صياغة تعليمات البرمجية في الكود سورس, لأنه يشمل العديد من الأسطر التي بها معطيات معقدة .

* تبادل البيانات مع برامج أخرى(ربما كتبت بلغات برمجة أخرى) هو بكل بساطة مستحيل, لأن هذه البيانات هي جزء من البرنامج نفسه .

ملاحظة أخيرة : حان الوقت لنتعلم فصل البيانات و البرامج التي تعالجها في ملفات مختلفة .

ليكون هذا ممكنا, سوف نقدم مجموعة متنوعة من الاليات لإنشاء الملفات و إرسال البيانات و إسترجاعها في وقت أخر .

لغات البرمجة تعرض تعليمات أكثر أو أقل تعقيد لأداء هذه المهام . عندما يتم التعامل مع مجموعات كبيرة من البيانات, يكون من المهم (ضروري) تنظيم العلاقة بين هذه البيانات , و علينا وضع أنظمة تدى بقواعد البيانات, يمكن أن تكون إدارتها معقدة للغاية . عندما تواجه مشاكل من هذا القبيل, يجب أن تفوض هذا إلى العديد من البرامج المتخصصة في هذا مثل : Oracle, IBM DB2, Sybase, Adabas, PostgreSQL, MySQL ....إلخ . البايثون يمكنها التواصل مع هذه الأنظمة, لكن سنترك هذا لوقت أخر(أنظر : إدارة قواعد البيانات, صفحة Error: Reference source not found).

طموحاتنا متواضعة جدا . البيانات التي لدينا ليست بمئات الألاف, نكتفي بألية بسيطة لحفظ البيانات في ملف متوسط الحجم, و من ثم إستخراجها عندما نحتاجها .

العمل مع الملفات

إستخدام الملف يشبه إلى حد كبير إستخدام كتاب . لإستخدام كتاب, يجب أن تجده أولا(بمساعدة إسمه), ثم يجب عليك فتحه . و عندما تنتهي من إستخدامه, تقوم بإغلاقه . و عندما يكون مفتوح, يمكنك قراءة المعلومات المختلفة, و يمكنك أيضا كتابة تعليقات توضيحية, لكن عموما لن تقوم بعمل الإثنين في وقت واحد . في جميع الحالات, يمكنك معرفة أين وصلت في داخل الكتاب, و ذلك بمساعدة أرقام الصفحات . تقرأ معظم الكتاب بإتباع نظام الصفحات, لكن يمكنك أيضا قراءة أي فقرة بشكل مضطرب .

كل ما قلناه عن الكتب ينطبق أيضا على ملفات الحاسوب .الملف يتكون من بيانات مخزنة على القرص الصلب, أو في قرص مرن, أو في إصبع usb أو في قرص مدمج(cd) . و التي يمكنك الوصول إليها من إسمها(و قد يشمل إسم الدليل). كنظرة تقريبية, قد تنظر إلى محتويات الملف كأنه سلسلة من الأحرف, مما يعني أنه يمكنك التعامل مع هذا المحتوى, أو أي جزء منه, بمساعدة دالات التي تتعامل مع سلاسل الأحرف[37].

إسم الملف - الدليل الحالي

[37] بعبارة أدق, يجب عليك أن تأخذ بعين الإعتبار أن محتوى الملف هو تسلسل من البايتات . معظم البايتات ممثلة بواسطة رموز, و العكس غير صحيح : سوف نتعلم في نهاية المطاق التمييز الواضح بين السلاسل من بايتات و السلاسل النصية .

لتبسيط التفسيرات التي تلي, سوف نقوم بتوضيح فقط الأسماء البسيطة للملفات التي سوف تعامل معها . إذا كنت تفعل هذا في تمارينك, الملفات سوف يتم صنعها و\أو يتم البحث عنها من قبل البايثون في الدليل الحالي . عادة ما يكون هذا الدليل في نفس مكان السكريبت, إلا إذا كنت تشغل السكريبت من خلال نافذة ال IDLE, في هذه الحالة, يتم تعيين الدليل الحالي عند تشغيل ال IDLE(في ويندوز, تعريف هذا الدلي هو جزء من خصائص أيقونة التشغيل).

إذا كنت تعمل مع IDLE, فإنه بالتأكيد تريد إجبار البايثون تغيير الدليل الحالي, بحيث يكون مثلما تريده . و للقيام بذلك, يجب عليك كتابة الأوامر التالية عند بداية الحصة . نحن نفترض أن الدليل الذي تريده هو "/**home/jules/exercices**" حتى لو كنت تعمل في نظام ويندوز(و هذه ليست قاعدة), يمكنك إستخدام نفس التعليمة(و لكن يجب عليك إستخدام \ بدل من / : لأن الأولى تعمل فقط على أنظمة UNIX) . البايثون سوف يقوم بالتحويلات اللازمة, هذا إذا كنت تعمل على Mac OS أو Linux أو Windows[38].

```
>>> from os import chdir
>>> chdir("/home/jules/exercices")
```

الأمر الأول يستدعي الدالة **chdir()** من وحدة **os** . وحدة **os** تحتوي عل جميع الدوالات التي تتعامل مع أنظمة التشغيل (os = operating system أي نظام تشغيل) بغض النظر عن نوعه .

الأمر الثاني يتسبب في تغيير الدليل (chdir =change directory أي تغيير الدليل)

> • يمكنك أيضا إدراج هذه الأوامر (التعليمات) في بداية البرنامج النصي, أو تحديد إسم المسار الكامل للملف الذي تريد معالجته, و لكن هذا قد يكون خطر على ثقل الكتابة في برامجك .
> • اختر أسماء ملفات قصيرة . تجنب قدر الإمكان الأحرف المعلمة و المسافات و العلامات المطبعية الخاصة . في بيئات عمل اليونكس (ماك, لينكس, *BSD* ...), ينصح في أغلب الأحيان بإستخدام الأحرف الصغيرة فقط .

شكلي الإستدعاء

[38] في حالة أنك تستخدم نظام تشغيل ويندوز, يمكنك تضمين في هذا المسار الرسالة التي تعين جهاز التخزين حيث يوجد الملف . على سبيل المثال D:/home/jules/exercices.

الأسطر التعليمات التي سوف نستخدمها هي فرصة لشرح اليات مثيرة للإهتمام . أنت تعرف أنه بالإضافة إلى الدوالات المدمجة في الوحدات الأساسية, البايثون يوقر لك كمية هائلة من الدوالات الخاصة, و التي تم تجميعها في وحدات . من الوحدات التي عرفتها الوحدة math و الوحدة tkinter .

لإستخدام الدالات من وحدة, يجب عليك إستدعائهم . لكن هذا يتم بطريقتين مختلفتين, كما سنرى بالأسفل, كل واحدة لديها مميزاتها و عيوبها .

هذا مثال على الطريقة الأولى :

```
>>> import os
>>> rep_cour = os.getcwd()
>>> print rep_cour
C:\Python22\essais
```

في السطر الأول من هذا المثال نحن نستدعي الوحدة **os**, التي تحتوي على وضائف كثيرة مثيرة للإهتمام التي تسمح لنا بالوصول إلى نظام التشغيل . أما السطر الثاني فهو يستخدم الدالة getcwd)_ من وحدة os[39]. كما ترون, الدالة **getcwd()** تقوم بإرجاع إسم الدليل الحالي (getcwd = get current working directory) . للمقارنة, هذا مثال على الطريقة الثانية :

```
>>> from os import getcwd
>>> rep_cour = getcwd()
>>> print(rep_cour)
C:\Python31\essais
```

في هذا هذا المثال الجديد, نحن إستدعينا من الوحدة os الدالة **getcwd()** فقط . بالإستدعاء بهذه الطريقة, سيتم دمج الدالة مع كود كما لو أننا كتبناه بأنفسنا . في الأسطر التي نستعملها, ليس من الضروري ذكر جزء الوحدة os .

و يمكننا إستدعاء العديد من الدالات من نفس الوحدة بنفس الطريقة :

39النقطة الفاصلة تعبر هنا عن علاقة الإنتماء . و هذا مثال على أسماء المؤهلة التي سيتم إستخدامها على نطاق واسع في ما تبقى من هذه الدورة . ربط الأسماء بمساعدة النقاط هي وسيلة لا بأس بها لعناصر التي تشكل جزءا من المجموعات, و التي هي ربما قد تكون من مجموعة أكبر, و إلخ . على سبيل المثال, تسمية systeme.machin.truc تشير إلى عنصر truc, و الذي هو جزء من المجموعة machin, و الذي هو جزء من المجموعة systeme . سوف نرى العديد من الأمثلة عن هذه التقنية, و خصوصا عندما ندرس أصناف الكائنات .

```
>>> from math import sqrt, pi, sin, cos
>>> print(pi)
3.14159265359
>>> print(sqrt(5))          # الجذر التربيعي ل 5
2.2360679775
>>> print(sin(pi/6))        # جيب زاوية 30 درجة
0.5
```

و يمكننا إستدعاء كل الدالات من وحدة , كما في :

```
from tkinter import *
```

لهذه الطريقة ميزة سهولة كتابة التعليمات البرمجية للدالات التي تم إستدعائها . و لديها أيضا عيب (خاصة في الشكل الأخير, عند إستدعاء جميع دالات الوحدة) تشوش مساحة الإسم الحالي . قد يكون أن بعض الدالات التي إستدعيتها لديها نفس إسم المتغير الذي عرفته أنت, أو نفس إسم دالة تم إستدعائها من وحدة أخرى . فإذا حدث هذا, واحد من الإسمين المتضاربين لن يتم الوصول إليه بشكل جيد .

في البرامج التي لديها بعض الأهمية, و التي تستدعي عدد كبير من وحدات من مختلف المصادر, سيكون من أفضل لها أن تستخدم الطريقة الأولى, و هذا معناه إستخدام أسماء مؤهلة بشكل كامل .

عموما, يوجد إستثناءات لهذه القاعدة في حالة معينة من الوحدة tkinter, لأنه يحتوي على دالات مطلوبة بشدة(عندما تقرر إستخدام هذه الوحدة).

كتابة متسلسلة في ملف

في البايثون, يتم توفير الوصول إلى الملفات عن طريق كائن الواجهة خاصة, التي تسمى كائن الملف . نحن نقوم بصنع هذا الملف بإستخدام الدالة المدمجة **open()**[40]. التي تقوم بإرجاع الكائن مع أساليب محددة, و التي تسمح لك بقراءة و كتابة في هذا الملف .

المثال التي يوضح كيفية فتح ملف , ثم كتابة سلسلتين نصيتين فيه, ثم إغلاقه . لاحظ أن إذا كان الملف غير موجود فسوف ستم إنشاءه تلقائيا . و من جهة أخرى, و إذا كان الملف موجود بالفعل و به بعض البيانات, الحروف التي سوف تسجلها ستكون بعد الموجودة . يمكنك أن تقوم بهذا التمرين مباشرة على سطر الأوامر :

40هذه الدالة, هي قيمة رجوه لكائن معين, و غالبا ما تسمى مصنع الدالة .

```
>>> obFichier = open('Monfichier','a')
>>> obFichier.write('Bonjour, fichier !')
>>> obFichier.write("Quel beau temps, aujourd'hui !")
>>> obFichier.close()
>>>
```

ملاحظات

- السطر الأول يقوم بإنشاء ملف الكائن **obFichier**, و التي ستشير (مرجع) إلى الملف الحقيقي(على القرص أو على القرص المرن) و هو سيكون إسمه (**Monfichier**) . تنبيه : لا تخلط بين إسم الملف مع إسم كائن الملف الذي يتيح الوصول إليه ! بعد هذا التمرين, يمكنك التحق من أن هذا الملف تم إنشائه في نظامك(في الدليل الحالي) إسم الملف هو **Monfichier** (و الذي تستطيع عرض محتواه مع أي محرر) .
- الدالة **open()** تنتظر برامترين[41], و الذين يجب أن يكونا سلسلتين نصيتين . البرامتر الأول سيكون إسم الملف الذي يجب فتحه, و الثاني هو إسم وضع الفتح . "**a**" يشير إلى فتح الملف بوضع إضافة (**append**), و هذا يعني أن البيانات التي سيتم حفظها سيتم إضافتها إلى نهاية الملف, إضافة إلى التي هي موجودة بالفعل . نستطيع أيضا إستخدام الوضع "**w**" (للكتابة - write), لكن عند إستخدام هذا الوضع, سيقوم البايثون بصنع ملف جديد(فارغ), و يكتب فيه البيانات, بداية من بداية الملف . فإذا وجد ملف بنفس الإسم, يتم مسحه و صنع ملف جديد .
- الأسلوب **write()** يكتب فعليا . و يجب أن تكون البيانات التي يجب كتابتها كبرامتر . هذه البيانات يتم حفظها في الملف واحد بعد الأخر(نحن نحدث هنا عن الوصول المتسلسل للملف). عند كل إستدعاء للدالة **write()** يتم إستمرار كتابة في الملف(مع الموجود بالفعل) .
- الأسلوب **close()** تغلق الملف . و هي متاحة لجميع الإستعمالات .

قراءة متسلسلة من الملف

سوف نقوم الأن بإعادة فتح الملف, لكن هذه المرة, من أجل قراءة المعلومات التي سجلناها في الخطوة السابقة :

---

[41] يمكن إضافة برامتر ثالث للإشارة إلى الترميز المتستخدم (أنظر إلى صفحة Error: Reference source not found).

```
>>> ofi = open('Monfichier', 'r')
>>> t = ofi.read()
>>> print(t)
Bonjour, fichier !Quel beau temps, aujourd'hui !
>>> ofi.close()
```

كما كان متوقعا, الأسلوب **read()** يقرأ البيانات في الملف و يحوله إلى متغير من نوع سلسلة نصية(string) . فإذا إستخدمنا هذا الأسلوب بدون برامترات, يتم نقل كامل محتويات الملف .

ملاحظات

- الملف الذي نريد إستدعائه لقرائته يدعى **Monfichier** . يجب علينا أن نكتب تعليمة فتح الملف ليشير إلى الملف . فإذا كان الملف غير موجود سوف تحصل على رسالة خطأ. على سبيل المثال :

```
>>> ofi = open('Monficier','r')
IOError: [Errno 2] No such file or directory: 'Monficier'
```

- على جهة أخرى, نحن غير ملزمين بإختيار إسم محدد لكائن الملف . نستطيع إختيار إسم أي متغير . و بالتالي في تعليمتنا الأولى نحن قمنا بصنع كائن ملف و سميناه **ofi**, و الضي سيكون إشارة للملف الأصلي **Monfichier**, و الذي سوف يتم فته للقراءة منه (البرامتر "**r**") .
- السلسلتين النصيتين التين وضعناهما قي الملف تم دمجهما في سطر واحد . هذا طبيعي, لأننا لم نستخدم أي رمز خاص عندما قمنا بحفظه . سوف نتعرف لاحقا على طريقة حفظ أسطر منفصلة
- الأسلوب **read()** يمكننا إستخدامه مع برامتر . و سوف يشير إلى عدد الحروف التي يجب أن تقرأ . بداية من الموقع الموجود بالفعل في الملف : الأسلوب **read()** يمكننا إستخدامه مع برامتر . و سوف يشير إلى عدد الحروف التي يجب أن تقرأ . بداية من الموقع الموجود بالفعل في الملف :

```
>>> ofi = open('Monfichier', 'r')
>>> t = ofi.read(7)
>>> print(t)
Bonjour
>>> t = ofi.read(15)
>>> print(t)
, fichier !Quel
```

فإذا لم يكن ما يكفي من الحروف في الملف, القراءة تتوقف في نهاية الملف :

•

```
>>> t = ofi.read(1000)
>>> print(t)
beau temps, aujourd'hui !
```

فإذا تم الوصول بالفعل إلى نهاية الملف, فإن read() تقوم بإرجاع سلسلة فارغة :

```
>>> t = ofi.read()
>>> print(t)
```

لا تنسى إغلاق الملف بعد إستعماله

```
>>> ofi.close()
```

> في كل ما سبق, لقد إفترضنا دون شرح أن السلاسل النصية يتم تبادلها بين مفسر البايثون و الملف . و هذا في الواقع غير صحيح, لأن السلاسل النصية يتم تحويلها إلى سلاسل بايتات ليتم تخزينها في ملفات . و بالإضافة إلى ذلك, هنالك معايير مختلفة للأسف لهذا الغرض . بالمعنى الدقيق, في البايثون ينبغي أن يكون واضحا معيار الترميز الذي يجب إستخدامه في ملفاتك : سنرى كيف يمكننا فعل هذا في الفصل القادم . في غضون ذلك, يمكنك الإعتماد على البايثون لأن البايثون يستخدم المعيار الإفتراضي لنظامك, و هذا سوف يجنبك المشاكل في التمارين الأولى . و مع ذلك, إذا كانت بعض المعلمات تظهر بشكل غريب, يرجى تجاهل هذا مؤقتا .

العبارة break للخروج من الحلقة

ليس علينا القول أننا بحاجة إلى حلقات في البرنامج عندما نتعامل مع ملف لا نعرف محتوياته . و الفكرة هي قراءة الملف جزء جزء حتى نصل إلى نهاية الملف .

الدالة التي بالأسفل تشرح هذه الفكرة . فهي تقوم بنسخ ملف بأكمله (بغض النظر عن حجمه) من خلال نقل 50 حرف في كل مرة :

```
def copieFichier(source, destination):
    "copie intégrale d'un fichier"
    fs = open(source, 'r')
    fd = open(destination, 'w')
    while 1:
        txt = fs.read(50)
        if txt == "":
```

```
            break
        fd.write(txt)
    fs.close()
    fd.close()
    return
```

فإذا أردت إختبار هذه الدالة, يجب عليك توفير دالتان : الأول إسم الملف الأصلي, و الثانية إسم الملف الذي تريد الإستنساخ إليه . على سبيل المثال :

```
copieFichier('Monfichier','Tonfichier')
```

ستلاحظ أننا عندما قمنا ببناء الحلقة, إستخدمنا طريقة مختلفة عن التي عرفناها سابقا . كما تعلم أن العبارة **while** يجب دائما أن تلحقا الشرط لتقييمه , و يتم تنفيذ الكتلة التي تلي في الحلقة, مادامت هذه الدالة صحيحة . و لكننا هنا قمنا بإستبدال الشرط التحقق برقم بسيط, و أنت تعرف[42] أن مفسر البايثون يعتبر أي قيمة غير الصفر قيمة صحيحة .

حلقة **while** بالأعلى سوف تعمل لأجل غير مسمى, لأن الشرط يبقى دائما صحيح . و مع ذلك, يمكننا أن نقطع هذه الحلقة بإستخدام عبارة **break**, و ربما يجب علينا أن نضع عدة عبارات توقف في عدة أماكن :

```
while <condition 1> :
    --- instructions diverses ---
    if <condition 2> :
        break
    --- instructions diverses ---
    if <condition 3>:
        break
    etc.
```

في دالتنا **copieFichier()**, من السهل أن يتم تنفيذ العبارة **break** بعد إنهاء قراءة الملف .

ملفات نصية

42 أنظر إلى صفحة Error: Reference source not found : تعبير حقيقي\مزيف

الملف النصي هو ملف يحتوي على أحرف "للطباعة - imprimables"[43] و فراغات التي تنظم الأسطر, هذه الأسطر يتم فصله عن بعض عن طريق رمز خاص غير مطبوع يسمى علامة نهاية السطر[44].

الملفات النصية هي الملفات التي نستطيع قراءتها وفهمها مع محرر نص بسيط، بدلا من الملفات الثنائية - على الأقل في جزء منه - الغير مفهومة للبشر . و يكون له معنى فقط عندما يتم فك شيفرته من قبل برامج خاصة . على سبيل المثال, الملفات التي تحتوي على صور و صوتيات و فيديوات ...إلخ . هم دائما تقريبا ملفات ثنائية . أعطينا مثال صغير على التعامل مع الملفات الثنائية , لكن في هذه الدورة, سوف نركز فقط على الملفات النصية .

من السهل جدا معالجة الملفات النصية مع البايثون . على سبيل المثال, هذه التعليمات كافية لصنع ملف نصي يتكون من أربعة أسطر :

```
>>> f = open("Fichiertexte", "w")
>>> f.write("Ceci est la ligne un\nVoici la ligne deux\n")
>>> f.write("Voici la ligne trois\nVoici la ligne quatre\n")
>>> f.close()
```

لاحظ أن في نهاية السطر **\n** حيث يتم إدراجها في السلاسل النصية, حتى نفصل بين أسطر النص السابق عندما نقوم بحفظه . من دون هذه العلامة, سوف يتم حفظ الحروف واحدة بعد الأخرى, كما في الأمثلة السابقة .

---

43 بالمعنى الدقيق, مثل ملف يحتوي على "بايتات قابلة للطباعة" التي قيمها يمكنها أن تمثل رموز طباعية في ترميز محدد جدا . سوف نناقش هذا بتفصيل أكثر في الفصل القادم . تحديدا, هو بيتات ذات قيمة رقمية ما بين 32 و 255 . و البايتات للقيمة أقل من 32 هي رموز للتحكم قديمة و عموما لا يمكن تمثيلها بواسطة أحرف .

44 إعتمادا على النظام التشغيل المستخدم, الترميز الموافق لنهاية السطر قد يكون مختلفا . و على سبيل المثال, في نظام تشغيل ويندوز هنالك تسلسل إثنين من البايتات (رمز الإرجاع و قفر السطر), في حين أن أنظمة التشغيل من نوع يونكس (مثل لينكس) يكفي أن تضع القفز السطر, أما في ماك فهو يستخدم رمز الرجوع . من حيث المبدأ, لا يوجد ما يدعو للقلق حول هذه الإختلافات . من خلال عمليات الكتابة, يستخدم البايثون إتفاقية الموجودة في نظامك . للقراءة, يقوم البايثون بتصحيحها حسب الإتفاقية (ما يعادلها) ..

عند قراءة الملف, أسطر الملف يمكن أن يتم إستراجها بشكل منفصل عن بعضهم . عن طريق الأسلوب **readline()**, على سبيل المثال, لن يقرأ هنا سوى سطر واحدة في كل مرة(بما في ذلك علامة نهاية السطر) .

```
>>> f = open('Fichiertexte','r')
>>> t = f.readline()
>>> print(t)
Ceci est la ligne un
>>> print(f.readline())
Voici la ligne deux
```

الأسلوب **readlines()** ينقل جميع الأسطر المتبقية في سلسلة نصية .

```
>>> t = f.readlines()
>>> print(t)
['Voici la ligne trois\n', 'Voici la ligne quatre\n']
>>> f.close()
```

ملاحظات

* في القائمة أعلاه تظهر في شكلها الخام, مع علامة إقتباس واحدة للسلاسل, وحروف خاصة في شكلها التقليدي . و يمكنك بالطبع إستعراض هذه القائمة(بمساعدة الحلقة **while**, على سبيل المثال) لإستخراج السلاسل الفردية .

* الأسلوب **readlines()** يجعل من الممكن قراءة ملف كامل بتعليمة واحد فقط . هذا ليس دائما ممكن, فمثلا لو كان الملف ليس كبير يمكن وضعه في متغير, و هذا معناه في ذاكرة الوصول العشوائي للحاسوب, لذا فمن الضروري أن يكون الحجم كافيا . فإذا كنت في حاجة لمعالجة ملفات ضخمة, و تريد أستخدام الأسلوب **readline()** في حلقة, كما في المثال التالي .

* لاحظ أن **readline()** هو أسلوب يرجع سلسلة نصية, في حين أن **readline()** ترجع قائمة . في نهاية الملف, **readlines()** تقوم بإرجاع قناة فارغة, في حين أن **realines()** تقوم بإرجاع سلسلة فارغة .

السكريبت التالي يوضح كيفية إنشاء دالة لتعامل مع الملفات النصية . في هذه الحالة, سوف يقوم بإستنساخ الملف النصي و سيحذف جميع الأسطر التي تبدأ برمز "**#**" :

```
def filtre(source,destination):
    "recopier un fichier en éliminant les lignes de remarques"
    fs = open(source, 'r')
```

```
    fd = open(destination, 'w')
    while 1:
        txt = fs.readline()
        if txt ==='':
            break
        if txt[0] != '#':
            fd.write(txt)
    fs.close()
    fd.close()
    return
```

لإستدعاء هذه الدالة, يجب عليك إستخدام برامترين : إسم الملف الأصلي, إسم الملف الذي سيتلقى النسخة التي تمت تصفيتها. على سبيل المثال :

```
filtre('test.txt', 'test_f.txt')
```

تسجيل و عرض مختلف المتغيرات

البرامتر المستخدم مع الأسلوب **write()** في الملف النصي يجب أن يكون سلسلة نصية . مع الذي تعلمناه حتى الأن, نحن لا نستطيع حفظ مع أنواع أخرى من البيانات لذا لنسطيع حفظها في ملف يجب علينا أن نحولها إلى سلسلة نصية (string). يمكننا أن نفعل ذلك بمساعدة الدالة المدمجة **()str** :

```
>>> x = 52
>>> f.write(str(x))
```

سوف نرى لاحقا أنه يوجد طرق أخرى لتحويل قيم رقمية إلى سلاسل نصية(أنظر : تنسيق السلاسل النصية, صفحة Error: Reference source not found). لكن السؤال ليس هنا . إذا قمنا بحفظ قيم رقمية بتحويلها أولا إلى سلاسل نصية, سوف نستطيع إذا أن نعيد حويلها إلى قيم رقمية عندما نقوم بقراءة الملف . على سبيل المثال :

```
>>> a = 5
>>> b = 2.83
>>> c = 67
>>> f = open('Monfichier', 'w')
>>> f.write(str(a))
>>> f.write(str(b))
>>> f.write(str(c))
>>> f.close()
>>> f = open('Monfichier', 'r')
>>> print(f.read())
52.8367
>>> f.close()
```

لقد قمنا بحفظ ثلاثة قيم رقمية . لكن كيف يمكننا تميز بين السلاسل النصية الناتجة, عندما نقوم بتشغيل الملف ؟ هذا مستحيل ! لا شيئ يشير إلى أنه يوجد 3 قيمة بدل من وحدة أو 2 أو 4 ....إلخ هنالك عدة حلول لهذه المشكلة . أفضل واحدة هل إستدعاء وحدة بايثون متخصصة : الوحدة **pickle**[45].
هذا هو شرح كيفية إستخدامها :

```
>>> import pickle
>>> a, b, c = 27, 12.96, [5, 4.83, "René"]
>>> f =open('donnees_test', 'wb')
>>> pickle.dump(a, f)
>>> pickle.dump(b, f)
>>> pickle.dump(c, f)
>>> f.close()
>>> f = open('donnes_test', 'rb')
>>> j = pickle.load(f)
>>> k = pickle.load(f)
>>> l = pickle.load(f)
>>> print(j, type(j))
27 <class 'int'>
>>> print(k, type(k))
12.96 <class 'float'>
>>> print(l, type(l))
[5, 4.83, 'René'] <class 'list'>
>>> f.close()
```

كما ترى في هذا المثال القصيرة, الوحدة **pickly** تسمح لك بحفظ البيانات مع المحافظة على نوعها . يتم تخزين محتويات المتغيرات الثلاثة **a** و **b** و **c** في ملف **donnees_test**, و من ثم ننشئها مرة أخرى, مع نوعها في المتغيرات **j** و **k** و **l** . يمكنك إذا تخزين القيم من نوع "الصحيحة" و "سلاسل نصية" و "قوائم" و الأنواع الأخرى التي سوف ندرسها في وت لاحق .
تنبيه, الملفات التي يتم معالجتها بمساعدة الدالة الوحدة **picle** ليس بملفات نصية, لكنها ملفات ثنائية[46].
لهذا السبب, يجب أن تكون مفتوحة بمساعدة الدالة **open()** مثلا . و يجب علك إستخدام البرامتر '**wb**' لفتح ملف نصي و الكتابة فيه(كما في السطر الثالث من مثالنا), و البرامتر '**rb**' لفتح الملف الثنائي و القراءة منه(مثل السطر الثامن من مثالنا).

45في اللغة الإنكليزية, المصطلح pickle معناه "حافظ" . و لقد أطلق هذا الإسم على هذه الوحدة لأنها تستخدم لتخزين البيانات مع الحفاظ على نوعها .

46في الإصدارات السابقة للبايثون, الوحدة pickle يتم إستخدامها مع الملفات النصية (و لكن السلاسل النصية يتم معالجتها داخليا مع الإتفاقيات مختلفة) . بيانات الملفات يتم صنعها مع صدارات مختلفة للبايثون غير متوافقة مباشرة . المحولات موجودة .

الدالة **dump()** من الوحدة **pickle** تحتاج برامترين : الأول هو المتغير الذي سوف يتم حفظه, و الثاني هم ملف الكائن الذي تعمل فيه . الدالة **pickle.load()** تنفذ عملها, و هذل معناه عودة كل متغير مع نوعه .

التعامل مع الإستثناءات: التعليمات **try - except - else**

الإستثناءات هي عمليات التي تعمل عندما يكشف المترجم أو المفسر خطأ أثناء تنفيذ البرنامج . عموما, يتم توقف تنفيذ البرنامج, و يتم إظهار رسالة خطأ أو أكثر . على سبيل المثال :

```
>>> print(55/0)
ZeroDivisionError: int division or modulo by zero
```

> يتم عرض معلومات أخرى, تشير على وجه الخصوص في البرنامج النصي على الخطأ الذي تم كشفه, و لكنها لا تتكاثر هنا .

رسالة الخطأ نفسها تحتوي على جزئين يفصل بينهما بنقطتين : الجزء الأول هو نوع الخطأ و الجزء الثاني هو معلومات هذا الخطأ .

في الكثير من الحالات, من الممكن التنبؤ مسبقا ببعض الأخطاء التي قد تحدث في أي لحظة معينة في البرنامج, و سوف نضم تعليمات محددة و التي يتم تفعيلها إذا حدثت هذه الأخطاء . في اللغات عالية المستوى مثل البايثون, من الممكن أيضا ربط ألية رصد مجموعة من التعليمات, و بالتالي تبسيط معلجة الأخطاء التي قد تحدث في أي من هذه التعليمات .

و تسمى ألية من هذا النوع عامة "اليات التعامل مع الإستثناءات" . البايثون تستخدم التعليمات **try** - **except** - **else** , التي تمكنك من إلتقاط الخطا و تشغيل جزء من سكريبت مححد لهذا الخطأ . هي تعمل على النحو الأتي .

يتم تنفيذ كتلة البيانات مباشرة بعد التعليمة **try** . فإذا حدث خطأ أثناء تنفيذ أي من هذه التعليمات, سيقوم البايثون بإلغاء التعليمة المخالفة و يتم تنفيذ بدل عنها كتلة التعليمات **except** . فإذا لم تقع أيت أخطاء يتم تنفيذ التعليمات التي تلي التعليمة **try**, و إذا الكتلة التي تلي التعليمة **else** يتم تنفيذهل (إذا كانت هذه التعليمة موجودة). في جميع الحالت, يتم إستمرار عمل البرنامج مع التعليمات الأخرى .

على سبيل المثال, أنظر في السكريبت الذي يطلب من المستخدم إدخال إسم الملف الذي يريد قرائته . فإذا كان الملف غير موجود نحن لا نريد إيقاف البرنامج و إظهار الخطأ . ما نريده هو عرض التحذير و جعل المستخدم يدخل إسم أخر .

```
filename = input("Veuillez entrer un nom de fichier : ")
try:
    f = open(filename, "r")
except:
    print("Le fichier", filename, "est introuvable")
```

إذا كنا نرى أن مثل هذه التجارب أن تعمل في أكثر من مكان في البرنامج, يمكننا إدراجها في وظيفة :

```
def existe(fname):
    try:
        f = open(fname,'r')
        f.close()
        return 1
    except:
        return 0

filename = input("Veuillez entrer le nom du fichier : ")
if existe(filename):
    print("Ce fichier existe bel et bien.")
else:
    print("Le fichier", filename, "est introuvable.")
```

و من الممكن أيضا إستخدام التعليمة **try** من مجموعة كتل **except**, كل واحدة منها تعمل عند نوع معين من الأخطاء, لكننا لن نطور ملاحق هنا . إذا أردت يرجى الرجوع إلى مراجع لغة البايثون .

تمارين

9.1 أكتب سكريبت الذي يسمح بصناعة و قراءة ملف نصي . يجب على برنامج أن يطلب من المستخدم إدخال إسم الملف . ثم تظهر إختيار, إما بحفظ أسطر جدبد أو إظهار محتوى الملف . يجب أن يكون المستخدم أن يدخل الأسطر على التوالي من النص بإستخدام زر الإدخال, لفصل كل سطر عن الأخر . لإكمال المدخلات يجب على المستخدم إدخال سطر فارغ(و هذا معناه الضغط على زر الإدخال فقط) . و يجب أن تظهر أسطر الملف مفصولة عن بعضها البعض بشكل طبيعي(رمز الخاص بنهاية السطر يجب أن لا يظهر) .

9.2 نفترض أنك لديك ملف نصي يحتوي على جمل مختلفة الأطوال . أكتب برنامج الذي يعرض لك أطول جملة .

9.3 أكتب السكريبت الذي يولد تلقائيا ملف نصي يحتوى لى جداول الضرب من 2 إلى 30 (كل جدول يجب أن يتكون من 20) .

9.4 أكتب السكريبت الذي يقوم بنسخ ملف نصي و يقوم بمضاعفة المسافات بين الكلمات ثلاثة مرات .

9.5 لديك تحت تصرفك ملف نصي في كل سطر يحتوي على قيمة عددية من نوع "حقيقي". على سبيل المثال :

```
14.896
7894.6
123.278
etc.
```

أكتب سكريبت الذي يقوم بنسح هذه القيم في ملف أخر و يقوم يتقريبها إلى أقرب عدد صحيح(التقريب يجب أن يكون صحيحا)

9.6 أكتب سكريبت الذي يقوم بمقارنة محتويات ملفين و يظهر أو إختلاف يصادفه .

9.7 بداية نفترض أنك لديك ملف نصي يحتوي على جمل مختلفة الأطوال . أكتب برنامج الذي يعرض لك أطول جملة .من الملفين الموجودين مسبقا **A** و **B**, أصنع ملف **C** الذي يحتوي بالتناول على عنصر من **A** و عنصر من **B** إلى أن يصل إلى نهاية واحدة من الملفين الأصليين . قم بإكمال **C** من خلال لعناصر المتبقية في الملف الأخر .

9.8 أكتب برنامج الذي يقوم بتشفير ملف نصي يحتوي على أسماء, و ألقاب و عناوين و رموز بريدية من أشخاص مختلفين(مثلا أعضاء في نادي) .

9.9 أكتب برنامج الذي يقوم بنسخ الملف المستخدم في التمرين السابق, ثم يقوم بإضافة تاريخ ولادة و جنس كل الأشخاص(سيقوم الحاسوب بإظهار الأسطر واحدة واحدة و يطلب من المستخدم إدخال البيانات لإضافية).

9.10 أفترض أنك قمت بحل جميع التمارين السابقة و أصبح لديك الأن ملف يحوي على معلومات العديد من الناس . أكتب السكريبت الذي يقوم بإستخراج من هذا الملف الأسطر التي تحتوي على رموز البريدية .

9.11 عدل السكريبت في التمرين السابق, بطريقة تجعله يجد الأسطر التي تحتوي على أسماء الأشخاص التي تبدأ أسمائهم **F** و **M** بطريقة أبجدية .

9.12 أكتب الدالات التي تؤدي نفس العمل وحدة **pickle** (أنر للصفحة 116) . هذه الدالات يجب أن تسمح بحفظ المتغيرات في ملف, و ترافقها معلومات حول نوعها بتدقيق .

10

المزيد من هياكل البيانات

حتى الأن, لقد قمنا بعمليات بسيطة . سوف نتحرك الأن بالسرعة القصوى . الهياكل البيانات التي قد إستخدمتها حتى الأن لديها بعض الميزات التي لا تعرفها, و حان الوقت أيضا لأعرض عليكم هياكل غيرها أكثر تعقيدا .

النقطة على سلاسل النصية

لقد درسنا بالفعل السلاسل النصية في الفصل الخامس . و على عكس البيانات الرقمية, و التي هي كيانات فردية, سلاسل النصية من نوع بيانات المركبة . و نعني بذلك أنها متكونة من كيانات أصغر و هي : الحروف . تبعا للظروف, يجب علينا أن نتعامل مع هذه البيانات المركبة, أحيانا كائنا واحد, و في بعض الأحيان تسلسل عناصر . في الحالة الأخيرة, ربما نريد الوصول إلى كل عنصر من هذه العناصر على حدة .

في الحقيقة, السلاسل النصية هي جزء من فئة من كائنات البايثون تسمى المتسلسلات, و تنتمي إلى هذه الفئة القوائم والأنفاق (tuples) . يمكننا القيام على المتسلسلات العمليات نفسها, ربما قد تكون تعرف بعضها, سوف نقوم بشرح عدد قليل منها في الفقرات القادمة .

Indiçage و الإستخراج و الطول

تذكير صغير بالفصل الخامس : السلاسل هم تسلسل من الحروف . كل واحدة منهم تأخذ مكان خاص في التسلسل . في البايثون, عناصر التسلسل دائما مفهرسة(أو مرقمة) بنفس الطريقة, هذا معناه تبدأ من الصفر . لإستخراج حرف من السلسلة, يكفي أن تضع إسم المتغير الذي يحتوي على السلسلة , ثم تضع مؤشره بين قوسين(قوسي نصف مربع أي : []) :

```
>>> nom = 'Cédric'
>>> print(nom[1], nom[3], nom[5])
é r c
```

غالبا يكون تحديد مكان حرف في نهاية السلسلة مفيدا . للقيام بذلك, يجب إستخدام مؤشرات سالبة . مثلا -1 للحرف الأخير و -2 للحرف قبل الأخير ...إلخ ::

```
>>> print (nom[-1], nom[-2], nom[-4], nom[-6])
cid
>>>
```

فإذا أردت تحديد عدد أحرف في السلسلة, نستخدم الدالة المدمجة **len()** :

```
>>> print(len(nom))
6
```

إستخراج أجزاء سلسلة

في الكثير من الأحيان, عندما تعمل مع السلاسل, قد ترغب بإستخراج سلسلة صغيرة من سلسلة طويلة . البايثون يوفر لك تقنية بسيطة تسمى التقطيع ("التشريح") . و تتكون هذه التقنية من قوسين (نصف مربع : []) و في داخلها مؤشرات بداية و نهاية الشريحة التي تريد إستخراجها :

```
>>> ch = "Juliette"
>>> print(ch[0:3])
Jul
```

في شريحة **[n,m]**, يتم تضمين بداية من **n**, و لا يتم تضمين **m** . فإذا أردت تخزين هذه الألية بسهولة, يجب أن تمثل أدلة تشير للمواقع في ما بين الحروف, كما في الرسم بالأسفل :

و نظرا لهذا النموذج, فإنه ليس صعبا أن تعرف أن إستخراج **ch[3:7]** تساوي "**iett**".

```
ch = "Juliette"
     ↑↑↑↑↑↑↑↑↑
     0 1 2 3 4 5 6 7 8
```

مؤشرات الشريحة لديها قيم إفتراضية : يعتبر أن أول مؤشر غير معرف مثل الصفر, في حين أن المؤشر الثاني يعتبر أن طول السلسلة الكاملة :

```
>>> print(ch[:3])        # أول ثلاثة أحرف
Jul
>>> print(ch[3:])        # ما بعد ثلاثة الأحرف
iette
```

و ينبغي على الحروف المعلمة أن لا تواجه مشاكل :

```
>>> ch = 'Adélaïde'
>>> print(ch[:3], ch[4:8])
Adé aïde
```

التسلسل, التكرار

و يمكن للسلاسل أن ترتبط بواسطة المعامل "+" و أن تتكرر بواسطة المعامل "*" :

```
>>> n = 'abc' + 'def'        # جمع سلسلة
>>> m = 'zut ! ' * 4         # تكرار
>>> print(n, m)
abcdef zut ! zut ! zut ! zut !
```

نلاحظ أن المعاملين "+" و "*" يستطيعون أيضا أن يستخدموا للجمع و ضرب عند تطبيقها على برامترات رقمية . الحقيقة أن نفس المعاملات يمكن أن تعمل بشكل مختلف تبعا للسياق الذي تعمل فيه و هي تستخدم الية مثيرة جدا تسمى حمولة الزائدة للمشغل . في لغات البرمجة الأخرى, الحمولة الزائدة للمعاملات ليست ممكنة دائما : و يجب علينا إستخدام رموز مختلفة للإضافة و التكرار, على سبيل المثال .

تمارين

10.1 أعرف بنفسك ماذا يحدث, عند تقطيع السلسلة, عندما يكون واحد أو أكثر من المؤشرات الشريحة خاطئة, و صف ذلك بأفضل طريقة ممكنة . (إذا كان المؤشر الثاني هو الأصغر من المؤشر الأول, على سبيل المثال' أو إذا كان المؤشر الثاني أكبر من حجم السلسلة) .

10.2 إقطع أجزاء سلسلة كبيرة على أجزاء كل واحدة بها 5 أحرف . ثم قم بجمع القطع في ترتيب عكسي . السلسلة يجب أن تحتوي على الأحرف المعلمة .

10.3 حاول كتابة دالة صغير إسمها **trouve()** التي تفعل بالضبط عكس الذي يفعله عامل المؤشر(هذا معناه ما بين قوسين []). بدل من البدء بمؤشر ليرجع لك حرف المطلوب, هذه الدالة تقوم بإرجاع مؤشر الكلمة التي تم إدخالها .

و بعبارة أخرى, أكتب دالة التي تأخذ برامترين : الأولى إسم السلسلة التي يجب معالجتها و الثانية الحرف الذي يجب إيجاده . يجب على الدالة أن تقوم بإرجاع أول مؤشر للحرف في السلسلة . Ainsi par على سبيل المثال :

```
print(trouve("Juliette & Roméo", "&"))
```

يجب أن يطبع : **9**

إنتبه : يجب أن تفكر في جميع الحالات المحتملة . و هذا يشمل أن تقوم الدالة بإرجاع قيمة معينة(على سبيل المثال -1) إذا كان الحرف الذي يبحث عنه غير موجود في السلسلة . و يجب على السلسلة أن تقبل الأحرف المعلمة .

10.4 حسن الدالة في التمرين السابق بإضافة برامتر ثالث : و هو مؤشر من أين يبدأ البحث في السلسلة . على سبيل المثال :

```
print(trouve ("César & Cléopâtre", "r", 5))
```

يجب أن يطبع : 15 (و ليس 4)

10.5 أكتب الدالة **compteCar()** التي تحسب عدد مرات تكرار الحرف في السلسلة في السلسلة :

```
print(compteCar("ananas au jus","a"))
```

يجب أن تطبع : 4

```
print(compteCar("Gédéon est déjà là","é"))
```

يجب أن تطبع : 3

دورة من التسلسل : العبارة for ... in ...

كثيرا ما يحدث أنه يجب علينا أن نتعامل مع السلسلة النصية بأكملها بحرف, من الحرف الأول إلى الحرف الأخير, للقيام بداية من أي واحدة أي معامل . ندعو هذه العملية ب "دورة" . سنقتصر فقط

على أدوات البايثون التي نعرفها, نحن نستطيع أن نقوم بترميز هذه الدورة بمساعدة الحلقة, تتمحور حول العبارة while :

```
>>> nom ="Joséphine"
>>> index =0
>>> while index < len(nom):
...   print(nom[iindex] + ' *', end =' ')
...   index = index +1
...
J * o * s * é * p * h * i * n * e *
```

هذه الحلقة تقطع السلسلة **nom** لتستخرج كل حروفها واحدة واحدة, ثم تطبع معها علامة نجمة . أنظر جيدا إلى الشرط المستخدم مع العبارة while وهو : **index < len(nom)** , و هذا معناه أن الحلقة ستتوقف عندما تصل للمؤشر 9(السلسلة تتكون من 10 حروف) . و في هذه الحالة سوف نتعامل مع كل حرف في السلسلة, لأن الفهرسة تبدأ من 0 و تنتهي ب 9 .

دورة التسلسل هي عملية متكررة جدا في البرمجة . لسهولة الكتابة, البايثون توفر لك هيكل حلقة أكثر ملائمة من الحلقة while, و هو الأسلوب **for.....in** :

مع هذه التعليمات, يصبح البرنامج في الأعلى هكذا :

```
>>> nom ="Cléopâtre"
>>> for car in nom:
...     print(car + ' *', end =' ')
...
C * l * é * o * p * â * t * r * e *
```

كما ترون, هذا هيكل الحلقة هو الأكثر ملائمة . فهو يريحكم من تعريف و زيادة متغير معين (عداد) لإدارة مؤشر الحرف الذي تريدون معالجته في كل تكرار(البايثون هو الذي يفعل ذلك) الهيكل for ..... in لا يظهر سوى الضروري, أي أن المتغير **car** سيحتوي على كل الحروف على التوالي, من البداية إلى النهاية .

العبارة for تتيح كتابة الحلقات, في تكرار الذي يعالج فيه على التوالي كل عناصر لسلسلة المقدمة . في المثال أعلاه. كان التسلسل هو سلسلة نصية . المثال التالي يوضح أنه يمكن لنا أن نطبق نفس العملية على القوائم (و سيكون نفس الشيئ مع ال tuples في وقت لاحق) :

```
liste = ['chien', 'chat', 'crocodile', 'éléphant']
for animal in liste:
    print('longueur de la chaîne', animal, '=', len(animal))
```

عند تشغيل هذا السكريبت يظهر لنا :

```
longueur de la chaîne chien = 5
longueur de la chaîne chat = 4
longueur de la chaîne crocodile = 9
longueur de la chaîne éléphant = 8
```

العبارة for .... in ... : هو مثال أخر من العبارة المجمعة . لا تنسه النقطتين في نهاية السطر, و مسافة البادئة للكتلة التي تليها .

الإسم بعد الكلمة المحجوز **in** هو الذي تجب معالجته . الإسم بعد الكلمة المحجوز for هو إسم الذي إخترته لمتغير الذي سيحتوي على جميع عناصر التسلسل على التوالي . هذا المتغير سيتم تعريف تلقائيا(هذا معناه أنه ليس من الضروري أن تحدده سلفا, و سوف يتكيف تلقائيا لعنصر التسلسل التي تتم معالجته(تذكر أنه في حالة وجود قائمة, ليس بالضرورة أن تكون جميع عناصره من نفس النوع) على سبيل المثال :

```
divers = ['lézard', 3, 17.25, [5, 'Jean']]
for e in divers:
    print(e, type(e))
```

عند تشغيل هذا السكريبت فسوف يعطينا :

```
lézard <class 'str'>
3 <class 'int'>
17.25 <class 'float'>
[5, 'Jean'] <class 'list'>
```

على الرغم من أنعناصر القائمة مختلفة الأنواع (سلسلة نصية, عدد حقيقي, عدد صحيح, قائمة), يمكننا أن نضعهم في المتغير **e** , دون ظهور أخطاء(و هذا أصبح ممكن مع إختيار النوع التلقائي للمتغيرات في البايثون).

تمارين

10.6 في قصة أمريكية, تسمى 8 فراخ بط على التوالي : Jack, Kack, Lack, Mack, Nack, Oack, Pack و Qack . أكتب سكريبت صغير الذي يولد هذه الأسماء من السلسلتين التاليتين :

prefixes = 'JKLMNOP' و **suffixe = 'ack'**

إذا إستخدمت العبارة **for ... in .....** يجب على سكريبتك أن يتكون من سطرين فقط .

10.7 أكتب في سكريبت دالة التي تبحث عن عدد الكلمات التي تحتويها الجملة المقدمة .

10.8 أكتب سكريبت الذي يبحث عن عدد الحروف e, é, è, ê, ë التي تحتويها الجملة المقدمة .

إنتماء عنصر لتسلسل : إستخدام التعليمة in وحدها

التعليمة **in** يمكن إستخدامهابشكل مستقل عن for, للتحقق ما إذا كان العنصر المعين هو جزء من سلسلة أو لا . يمكنك على سبيل المثال إستخدام التعليمة **in** للتحقق من أن حرف أبجدي هو جزء من مجموعة معينة أو لا :

```
car = "e"
voyelles = "aeiouyAEIOUYàâéèêëùîï"
if car in voyelles:
    print(car, "est une voyelle")
```

بنفس الطريقة, يمكنك التحقق من إنتماء عنصر لقائمة :

```
n = 5
premiers = [1, 2, 3, 5, 7, 11, 13, 17]
if n in premiers:
    print(n, "fait partie de notre liste de nombres premiers")
```

> هذه التعليمة قوية جدا لأنها حلقة حقيقة للتسلسل . كتمرين, أكتب التعليمات الـتي من شأنها أن تفعل ذلك بإستخدام الحلقة التقليدية بإستخدام العبارة *while* .

تمارين

10.9 أكتب الدالة **estUnChiffre()** التي تقوم بإرجاع "صحيح", إذا كان البرامتر الممرر عبارة عن رقم, و إلا سوف يقوم بإرجاع "خطأ" . أختبر جميع أحرف سلسلة الدورة بمساعدة الحلقة for .

10.10 أكتب الدالة **estUneMaj()** التي تقوم بإرجاع "صحيح" إذا كان البرامتر الممرر هو حرف كبير . حاول النظر إلى الأحرف الكبيرة المعلمة !

10.11 أكتب الدالة **chaineListe()** التي تقوم بتحويل جملة إلى سلسلة من الكلمات .

10.12 إستخدم الدالات في التي تم تعريفهم في التمارين السابقة لكتابة سكريبت الذي يقوم بإستخراج من نص جميع الكلمات التي تبدأ بحرف كبير .

10.13 إستخدم الدالات التي تم تعريفها في التمارين السابقة لكتابة دالة التي تقوم بإرجاع رقم الحرف الكبير الموجود في جملة المقدمة بواسطة البرامتر .

السلاسل الغير قابلة للتعديل

لا يمكنك تحرير محتويات سلسلة موجودة . و بعبارة أخرى, لا يمكنك إستخدام المعامل [] على الجانب الأيسر من عبارة التكليف . على سبيل المثال, حاول تشغيل البرنامج الصغير بالأسفل (الذي يبحث بالتخمين لإستبدال حرف في سلسلة) :

```
salut = 'bonjour à tous'
salut[0] = 'B'
print(salut)
```

النتيجة المتوقعة من المبرمج الذي كتب هذه التعليمات هي "Bonjour a tous" (مع B كبيرة) . لكن على عكس التوقعات, هذا السكريبت يظهر خطأ "TypeError: 'str' object does not support item assignment" . و هذا الخطأ سببه السطر الثاني من البرنامج . عندما تحاول إستبدال حرف بحرف أخر من السلسلة, و لكن هذا غير مسموح به .

لكن هذا السكريبت يعمل جيدا :

```
salut = 'bonjour à tous'
salut = 'B' + salut[1:]
print salut
```

في هذا المثال, في الحقيقة, نحن لم نغير سلسلة **salut** . نحن صنعنا متغير جديد, بنفس الإسم, في السطر الثاني من السكريبت .

السلاسل متشابهة

كل عوامل المقارنة التي تحدثنا عنها التي تتحكم في تدفق التعليمات(هذا معنا التعليمات : if ... else ... elif) تعمل أيضا مع السلاسل النصية . و هذا قد يكون مفيد لفرز الكلمات حسب الترتيب الأبجدي :

```
while True:
    mot = input("Entrez un mot quelconque : (<enter> pour terminer)")
    if mot ==="":
        break
    if mot < "limonade":
        place = "précède"
    elif mot > "limonade":
        place = "suit"
    else:
        place = "se confond avec"
    print("Le mot", mot, place, "le mot 'limonade' dans l'ordre alphabétique")
```

هذه المقارنة ممكنة, و ذلك لأن في كل معايير الترميز, الرموز الرقمية تمثل الحروف التي تم تعينها في ترتيب أبجد, على الأقل بالنسبة للحروف الغير معلمة . في نظام الترميز ASCII, على سبيل المثال A=65, B=66, C=67 ... إلخ .

أرجو منك أن تفهم أنه لا يعمل سوى للكلمات التي بالصغير أو بالكبير, التي لا تحتوي على أي حروف معلمة . في الحقيقة, إذا كنت تعرف إن الحروف الكبيرة و الصغيرة تستخدم مجموعة من الرموز المتميزة . أما بالنسبة إلى الحروف المعلمة, فقد رأيت أنها يتم ترميزها في خارج مجموعات تتألف من ASCII . إن بناء خوارزمية التي تقوم بفرز حسب الحروف الأبجدية و التي ترى في كل مرة حالة الحروف و لهجاتهم, ليست بالمهمة السهلة !

المعيار يونيكود

من المفيد في هذا المستوى الإهتمام بقيم المعرفات الرقمية المرتبطة بكل حرف، في البايثون 3 تكون مجموع الحروف (من صنف سلسلة حروف) هي سلاسل تخضع لنفس الترميز الموحد[47], و هذا يعني بأن المحددات الرقمية لحروفها تكون أحادية المعنى(فلا يمكن أن يوجد إلا حرف مطبعي بالنسبة لكل رمز) و كونية(فالمحددات المختارة تشمل مجموع الحروف المستعملة في مختلف لغات العالم).

ففي بداية ظهور تكنولوجيا المعلوميات،في وقت كانت فيه قدرات تخزين أجهزة الحواسيب جد محدودة، بحيث لم نتخيل بأن هذه سوف تستعمل في يوم ما لمعالجة نصوص أخرى غير تلك المتعلقة بتقنية الاتصالات، و خاصة باللغة الإنجليزية. لذلك كان يبدو من المعقول أن نضع لهاته الحواسيب لغة تتكون من مجموعة حروف محدودة، و بهذا الشكل يمكننا أن نمثل كل واحدة من هاته الحروف بعدد قليل من البتات، و بهذا شغل أقل حيز ممكن في وحدات التخزين المكلفة في ذلك الوقت . *فنظام الحروف ASCII*[48] الذي اختير في ذلك الوقت كان يعتقد بأن 128 حرف كافية له (مع علمنا بعدد التركيبات الممكنة لمجموعات تتكون من 7 بتات[49]). وبتوسيعها بعد ذلك إلى 256 حرف ، أصبح من الممكن جعلها مناسبة لمتطلبات معالجة النصوص

---

[47] في الإصدارات السابقة للبايثون, السلاسل النصية من نوع string هي في الواقع تسلسل من البايتات (و التي تمثل أحرف, لكن مع عدد من القيود المزعجة نوعا ما) . و لكن كان هنالك بالفعل نوع ثاني من السلاسل, و هو النوع Unicode لمعالجة السلاسل النصية بالمعنى الذي نفهمه نحن .

[48] ASCII = *American Standard Code for Information Interchange*

[49] En fait, on utilisait déjà les octets à l'époque, mais l'un des bits de l'octet devait être réservé comme bit de contrôle pour les systèmes de rattrapage d'erreur. L'amélioration

المكتوبة في لغات أخرى غير الإنجليزية، لكن و كثمن لذلك تشتتت المعايير (فعلى سبيل المثال ،معيار (ISO-8859-1 (latin-1 يقوم بترميز جميع الحروف المعلمة في اللغة الفرنسية أو الألمانية (من بين أخرى)، لكنها لا تستطيع ترميز أي حرف إغريقي عبري أو سريالي. فالنسبة لهاته اللغات، ينبغي على التوالي استعمال المعايير ISO-8859-5 ,ISO-8859-8 ,ISO-8859-7، والتي هي بالطبع غير متوافقة فيما بينها هذا إضافة إلى أنه وجب استعمال معايير أخرى مختلفة للغات مثل العربية أو التشيكية أو الغنهارية...

وتكمن أهمية هاته المعايير القديمة في بساطتها، فهي تسمح بالطبع لمطوري تطبيقات الحواسيب إعتبار أن كل حرف مطبعي يمثل واحد بايت، وكنتيجة لذلك فإن مجموعة من الحروف لا تمثل سوى متوالية من بايتات.حيث أنها هي الطريقة التي إشتغل بها النموذج القديم للمعطيات من نوع سلسلة نصية في بايثون (في النسخ السابقة للنسخة 3.0).

ولكن، وكما أشرنا إلى ذلك بشكل موجز في الفصل 5، فلا يمكن أبدا لتطبيقات الحواسيب الحديثة أن تكتفي فقط بهاته المعايير الضيقة. فينبغي الآن أن نكون قادرين على ترميز في نفس النص جميع حروف أبجدية أي لغة. لذلك تم إنشاء تنظيم دولي يدعى كونسورتيوم يونيكود، الذي تمكن من تطوير معيار كوني تحت إسم يونيكود، هذا المعيار الجديد يهدف إلى إعطاء لكل حرف من كل نظام لغوي مكتوب اسما و محدداً رقمياً، وذلك بطريقة موحدة، كيفما كان الحاسوب أو البرنامج المستعمل.

لكن هنا تطرح مشكلة، فبسعيه نحو الكونية، ينبغي على معيار يونيكود أن يعطي محدداً رقمياً مختلفاً للعديد من عشرات الالاف من الحروف. فبالطبع لا يمكن لكل هاته المحددات أن يتم ترميزها تحت أوكت واحدة. ولعله سيكون من المغري أن نعلن على أنه في المستقبل، سيكون بالإمكان ترميزه كل حرف باستعمال اثنان بايت (هذا الذي يعطينا 65536 إمكانية) أو باستعمال ثلاث (16 777 216 إمكانية) أو أربع (أكثر من أربع مليارات إمكانية). فكل واحدة من هاته الاختيارات الصعبة، ومع ذلك تنتج الكثير من السلبيات. أول هاته السلبيات وهي مشتركة بين الجميع، هي اننا و باستعمالنا لهاته الاختيارات نفقد التوافق مع العديد من الوثائق الحاسوبية الموجودة مسبقا، (وخصوصا البرامج)، التي استعمل في ترميزها المعايير القديمة، و التي هي مبنية على أساس نموذج (كل حرف يساوي 1 بايت)؛ثاني هاته السلبيات تكمن في عدم القدرة على تلبية مطلبين متناقضين: فإذا قبلنا باستعمال 2 بايت فنحن عند اذن نجازف بعدم إيجاد إمكانيات من أجل تعريف

ultérieure de ces systèmes permit de libérer ce huitième bit pour y stocker de l'information utile : cela autorisa l'extension du jeu ASCII à 256 caractères (normes ISO-8859, etc.).

أحرف نادرة، أو سمات أحرف ستكون مطلوبة بالتأكيد في المستقبل؛ ومن ناحية أخرى، فإذا افترضنا إستعمال ثلاث، أربع أو أكثر من ذلك، فنحن عندئذ نتجه نحو إهدار وبشكل وحشي للموارد، حيث أن أغلب النصوص المستعملة لا تحتاج إلا لعدد محدود من الأحرف وبذلك فالعدد الأكبر من هاته الأوكت لن يحتوي سوى على أصفار.

ولكي لا نجد أنفسنا محاصرون في قيود من هذا النوع، فإن معيار يونيكود لا يقوم بوضع أية قواعد تخص عدد البايتات أو البتات المحفوظة لاستعمالها في الترميز. حيث أن هذا المعيار يقوم بتحديد قيمة المعرف الرقمي المقترن مع كل حرف. حسب الحاجة، حيث أن كل نظام كمبيوتر هو حر في استعمال نظام الترميز الداخلي الذي يناسبه في ترميز هذا المعرف حسب الحاجة، كمثال على ذلك فيمكن ترميزه على شكل عدد صحيح عادي. وذلك كمعظم لغات البرمجة الحديثة، كلغة بايثون التي قد تم تجهيزها ببيانات من نوع حروف (أو سلسلة محارف)، والذي تتوافق تماما مع معيار يونيكود. فالتمثيل الداخلي لهاته الرموز الرقمية المقابلة ليس بذي أهمية للمبرمج.
سنرى لاحقا في هذا الفصل أنه من الممكن وضع في سلسلة من هذا النوع أي تركيبة أحرف من الأبجديات المختلفة (قد تكون ASCII قياسية، أحرف معلمة، رموز رياضية أو الأحرف اليونانية، السيريلية أو العربية، وما إلى ذلك.)، والتي يمكن تمثيلها داخليا بواسطة رمز رقمي فريد من نوعه.

تسلسل الأوكت : نوع البايت

في هاته المرحلة من التفسيرات الخاصة بنا، فنحن بحاجة ماسة لتوضيح شيء آخر...

لقد راينا سابقاً أن معيار يونيكود لا يقوم بتثبيت أي شيء آخر غير القيم الرقمية، بالنسبة لكل المعرفات القياسية التي مناط بها و بشكل لا لبس فيه مهمة وصف حروف الهجاء المكونة للغات العالم بأسره (أكثر من 240،000 لغة في نوفمبر تشرين الثاني 2005). ولكن ورغم ذلك فمعيار يونيكود لا يحدد بأي شكل من الأشكال كيف سيتم ترميز هاته القيم بشكل ملموس كمجموعة أوكتيات أو بايتات.

بالنسبة للعمل الداخلي للتطبيقات الحاسوبية، فهذا ليس بذي أهمية، فمصممي لغات البرمجة، مترجميها أو مفسريها سوف يتمكنون من الاختيار وبكامل حرية تمثيل هاته الحروف باستعمال 8، 16، 24، 32، 64 بايت، أو حتى (على الرغم من أننا لا نرى الجدوى من ذلك) تمثيلها باستعمال أعداد

حقيقية ذات فاصلة العائمة: هذا يبقى اختيارهم وهو لا يعنينا. لذلك ليس علينا القلق بشأن الشكل الفعلي للأحرف داخل سلسلة المحارف في بايثون.

ولكن وعلى عكس ذلك لوحدات الإدخال و الإخراج. فبالنسبة لنا كمطورين فهو واجب علينا أن نبين وبشكل دقيق ما هو نوع المعطيات التي تنتظرها برامجنا، فهل هاته البيانات سيتم إدخالها باستعمال لوحة المفاتيح أو سيتم استيراد من أي مصدر كيف ما كان.إظافة إلى ذلك وجب علينا اختيار شكل البيانات التي سيتم تصديرها إلي جهاز محيط، أن يكون سواء طابعة، قرص صلب، أو شاشة.

فبالنسبة لوحدات الإدخال والإخراج الخاصة بالحروف، وجب علينا دائما أن نأخذ في الحسبان، أن الأمر يتعلق فعليا بمتوالية من الأوكتيات، وأنه يجب استخدام آليات مختلفة لتحويل سلاسل الأوكتيات هاته إلى سلاسل حروف والعكس بالعكس.

بايثون يتيح اليوم نوعا جديدا من البيانات يدعى (البايت)، وقد تم تطويره على وجه التحديد من أجل التعامل مع متواليات (أو سلاسل) الأوكت. فالبيانات من نوع بايت تشبه كثيرا البيانات من نوع سلسلة المحارف، مع فارق بسيط هو أنها تعتبر متواليات أوكتيات، وليس تسلسل لأحرف. ولكن من المؤكد أن الأحرف يمكن أن يتم ترميزها على شكل أوكت، والأوكت فك تشفيرها لتغدو حروفا، ولكن ليس بشكل لا لبس فيه: حيث أن هناك معايير عدة لترميز وفك التشفير، لنفس السلسلة التي يمكن تحويلها إلى عدة سلاسل من البايتات المختلفة.

فعلى سبيل المثال[50], سوف نقوم بتمرين بسيط حول الكتابة والقراءة في ملف نصي باستعمال سطر الأوامر، مستغلين بعضا من الإمكانيات التي توفرها الدالة **open()**، والتي لم نصادفها لحد الآن، حيث سنسعى للقيام بهذا التمرين مع سلسلة تحتوي على عدد قليل من الحروف المعلمة، ورموز أخرى غير.

```
>>> chaine = "Amélie et Eugène\n"
>>> of =open("test.txt", "w")
>>> of.write(chaine)
17
>>> of.close()
```

[50] على سبيل المثال, نفترض أن التميز الإفتراضي القياسي على نظام تشغيلك هو Utf-8 . فإذا كنت تستخدم نظام تشغيل قديم يستخدم معايير مثل CP437 و CP850 و CP1252 و ISO8859-1 (Latin-1), فإن النتائج قد تختلف قليلا فيما يتعلق بالأعداد و قيم البايتات, و لكن يجب أن لا تكون لديك صعوبة في تفسر ما تحصل عليه .

مع هذه الأسطر القليلة, لقد قمنا بحفظ سلسلة نصية في هيئة أسطر نصية في ملف, بالطريقة المعتادة . دعونا نقوم بقراءة هذا الملف, لكن سوف نقوم بفتحه في الوضع البيناري, و سوف نقوم بتمرير البرامتر "**rb**" إلى الدالة **open()** . في هذا الوضع, يتم نقل البايتات بوضع خام, دون تحويل من أي نوع . و القراءة عن طريق الدالة **read()** التي ستعطينا سلسلة نصية كما في الفصل السابق, لكن في سلسلة من البايتات, و المتغير الذي يتلقى تلقائيا نوع المتغير كنوع بايت :

```
>>> of =open("test.txt", "rb")          # "rb" => وضع القراءة (r) بيناري (b)
>>> octets =of.read()
>>> of.close()
>>> type(octets)
<class 'bytes'>
```

بذلك, نحن لن نسترد السلسلة الأصلية, و لكن في ترجمته بالبايت . حاول عرض هذه المعطيات بمساعدة الدالة **print()** :

```
>>> print(octets)
b'Am\xc3\xa9lie et Eug\xc3\xa8ne\n'
```

ماذا تعني هذه النتيجة ؟ عندما نطلب من البايثون عرض معطيات من نوع بايت بمساعدة الدالة **print()**, البايثون يوفر لنا في الواقع التمثيل, بين علامتى إقتباس للإشارة إلى أنها سلسلة, و لكن هذه سبقت بحرف b صغيرة الخاصة التي تعيين سلسلة من نوع بايت, مع هذه المصطلاحات :

* و تتمثل قيم البايت الرقمية بين 32 و 127 من حروف ال ASCII .

* يتم تمثيل بعض القيم الرقمية التي تقل عن 32 بطرق تقليدية, مثل رمز (أو حرف) نهاية السطر .

* أما ابايتات المتبقية فتتمثل قيمتها بأعداد الست العشرية, و سبقها x\ .

في مثالنا, جميع الحروف الغير معلمة للسلسلة يستخدم لترميز كل واحد منها بمساعدة بايت واحد الموافق لجدول ASCII : و نحن نعرف ذلك . أما للحروف المعلمة, (التي لا توجد في الكود ASCII), يتم ترميزها ببايتين : مثلا xc3\ و xa9\ للحرف é و xc3\ و xa8\ ل è . هذا الشكل المعين للترميز هو المعيار UTF-8, الذ سوف نشرحه بتفاصيل أكثر في الصفحات القادمة .

التمثيل الذي تم الحصول بواسطة **print()** يساعدنا على التعرف على سلسلتنا الأولية, لكنها لا تبين لنا جيدا بما فيه الكفاية بالبايتات . لذا دعونا نجرب شيئا أخر . هل تعرف ان للمرء أن يفحص محتوى التسلسل, عنصر بعد عنصر, بمساعدة دورة الحلقة . لنرى ما يحدث هنا :

```
>>> for oct in octets:
...     print(oct, end =' ')
...
65 109 195 169 108 105 101 32 101 116 32 69 117 103 195 168 110 101 10
```

هذه المرة, نحن نرى بوضوح البايتات : نحن قمنا بإرجاع جميع القيم الرقمية, بالترقيم العشري .

الحروف المعلمة التي تم ترميزها ببايتين في المعيار UTF-8, الدالة **len()** لا ترجع لنا نفس القيمة للسلسلة النصية, و لمعادلتها يجي علينا ترميزها ب UTF-8 في سلسلة بايتات .

```
>>> len(chaine)
17
>>> len(octets)
19
```

عمليات إستخراج العناصر, و التقطيع, إلخ ... تعمل بطريقة مماثلة مع البيانات من موع بايت و نوع string, على الرغم من أن النتائج مختلفة, بطبيعة الحال :

```
>>> print(chaine[2], chaine[12], "---", chaine[2:12])
é g --- élie et Eu
>>> print(octets[2], octets[12], "---", octets[2:12])
195 117 --- b'\xc3\xa9lie et E'
```

إنتبه, لا يمكن حفظ سلسلة من البايتات في ملف نصي . على سبيل المثال :

```
>>> of =open("test.txt", "w")
>>> of.write(octets)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not bytes
```

لحفظ سلسلة من البايتات, يجب عليك دائما فتح الملف في الوضع البيناري(الثنائي), و إستخدام البرامتر "wb" بدل من "w" في التعليمة **open()** .

ملاحظة أخيرة : نحن نستطيع أن نعرف متغير من نوع بايت و أن نعطيه قيمة حرفية, بإستخدام هيكل جملة التالية :

var = b'chaîne_de_caractères_strictement_ASCII '

الترميز UTF-8

كل ما سبق يشير إلى أن السلسلة النصية الأولية في مثالنا يتم تحويل تلقائيا, عند حفظها في ملف, السلسلة من البايتات وفق لمعايير ال UTF-8 . التسلسل من البايتات التي ناقشناها حتى الأن تتوافق مع شكل معين من أشكال الترميز الرقمية, لسلسلة النصية "Amélie et Eugène " .

قد يبدوا لك هذا للوهلة الأولى معقدا بعض الشيئ, و انت تقول أن لسوء الحظ الترميز المثالي لا وجود له . إعتمادا على الذي سنقوم به, قد يكون من الأفضل ترميز النص بعدة طرق مختلفة . و لهذا السبب قد تم تعريفه, بالتوازي مع معايير Unicode, عدة معيايير ترميز : UTF-8 و UTF-16 و UTF-32 و بضعة خيارات أخرى . كل هذه المعايير تستخدم نفس المعرفات الرقمية لترميز الحروف, لكنهم يختلفون في طريقة حفظ هذه المعرفات في شكل بايتات . لا تقلق : على الأرجح أنك لن تتعامل سوى مع الأولى (UTF-8) . الأخرين لا يقلقون سوى من المتخصصين في المجالات الأخرى .

معيار الترميز القياسي UTF-8 هو المعيار المفضل لمعظم النصوص الحالية, و ذلك لأن :

* من ناحية إنه يتصمن توافق تام مع ترميز النصوص ASCII (كما هو الحال مع الكودات المصدرية للبرامج), بالإضافة إى أنه يتوافق جزئيا مع نصوص تم ترميزها مع مشتقاته, مثل Latin-1 .

* من ناحية أخرى, هذا المعيار الجديد هو واحد من الأكثر كفائة في إستخدام موارد الحاسوب, على الأقل النصوص المكتوبة بلغات غربية .

في هذا المعيار, يتم ترميز حروف ASCII القياسية في بايت واحد . أما بالنسبة للبقية فيتم ترميزها عادة في بايتين, و في بعض الأحيان 3 أو 4 بايت للحروف الأكثر ندرة .
و على سبيل المقارنة, نذكر أن المعيار الأكثر إستخداما من قبل الفرنكوفنيين قبل UTF-8 هو المعيار Latin-1 (و هو لا يزال واسع الإنشار, خاصة في بيئات عمل ويندوز تحت إسم CP1252[51]). هذا المعيار يسمح بترميز حرف بايت واحد لمجموعة معينة من الأحرف المعلمة, الموافق للغات الرئيسية لأوروبا الغربية (فرنسية, الألمانية, البرتغالية, إلخ) .

[51] في نافذة موجه الأوامر في دوس في ويندوز إكس بي, الترميز الإفتراضي هو CP850 .

المعيارين UTF-16 و UTF-32 يتم الترميز فيهما ببايتين بالنسبة للأولى و 4 بايتات بالنسبة للثانية . لا تستخدم هذه المعايير سوى للإستخدامات الخاصة جدا, مثل معالجة السلاسل النصية الداخلية بواسطة مترجم أو مفسر .

التحويل (ترميز\فك ترميز) السلاسل

مع إصدارات الباثون التي سبقت الإصدار 3, مثل العديد من لغات البرمجة, فإنه في كثير من الأحيان يتم تحويل ترميز السلاسل النصية من معيار ترميز إلى أخر . لأن الإتفاقيات و الاليات معتمد عليها الأن, و سوف لا يكون لديك الكثير لتقلق على البرامجك الخاصة للتعامل مع المعطيات الحديثة .

عندما حدث ذلك يجب علينا تحويل الملفات التي تم ترميزها وفقا معايير قديمة أو\و خارجية : المبرمج الذي يستحق هذا الإسم يجب أن يكون قادر على أداء هذه التحويلات . لحسن الحظ, البايثون توفر لك الأدوات اللازمة, في شكل أساليب للكائنات المعنية .

تحويل سلسلة بايت إلى سلسلة نصية (string)

على سبيل المثال, أنظر إلى تسلسل البايتات التي تم الحصول عليها في نهاية التمرين الصغير السابق . فإذا كنت تعلم فإن هذا التسلسل يتوافق مع النص وفقا لمعايير الترميز UTF-8, نحن نستطيع فك ترميز سلسلة نصية بمساعدة الأسلوب **decode()**, مع البرامتر "Utf-8" (أو بواسط "utf-8" أو " Utf8" أو "utf8") :

```
>>> ch_car = octets.decode("utf8")
>>> ch_car
'Amélie et Eugène\n'
>>> type(ch_car)
<class 'str'>
```

هذا المسار للسلسلة المتحصل عليها يوفر لنا العديد من الحروف, و هذه المرة :

```
>>> for c in ch_car:
...     print(c, end =' ')
...
A m é l i e   e t   E u g è n e
```

تحويل سلسلة نصية (string) إلى سلسلة بايت

لتحويل سلسلة نصية إلى سلسلة بايت, يتم ترميزها وفقا للمعايير معينة, نستخدم الأسلوب **()encode**, و الذي يعمل بشكل مماثل للأسلوب **()decode** المذكور أعلاه . على سبيل المثال, قم بتحويل نفس السلسلة النصية, إلى Utf-8 و Latin-1 للمقارنة بينهما .

```
>>> chaine = "Bonne fête de Noël"
>>> octets_u = chaine.encode("Utf-8")
>>> octets_l = chaine.encode("Latin-1")
>>> octets_u
b'Bonne f\xc3\xaate de No\xc3\xabl'
>>> octets_l
b'Bonne f\xeate de No\xebl'
```

في سلاسل بايت التي تم الحصول عليها, فمن الواضح أن الحروف المعلمة ê و ë تم ترميزها بمساعدة بايتين في حالة Utf-8, و بمساعدة بايت واحد في حالة Latin-1 .

التحويلات التلقائية عند معالجة الملفات

يجب عليك الأن إعادة النظر إلى ما يحدث عندما تريد تخزين سلسلة نصية في ملف نصي . في الحقيقة حتى الأن, نحن لم نلفت الإنتباه إلى مشكلة ترميز معايير هذه السلاسل, لأن الدالة **()open** للبايثون لديها لحسن الحظ إعدادات إفتراضية مناسبة لحالات محددة حديثة . عند فتح ملف للكتابة عليه, على سبيل المثال, نحن نختار "w" أو "a" كبرامتر ثاني ل**()open**, البايثون يقوم تلقائيا بترميز السلاسل بمعايير الإفتراضية لنظام التشغيل الخاص بك (في أمثلتنا, يقوم بترميزها وفق معايير Utf-8), و يتم تنفذ عمليات التحويل العكسي من خلال عمليات القراءة[52]. إذا, يمكننا إتباع منهج دراسة الملفات, في الفصل السابق, دون أن يعيقكم الشرح المفصل للغاية ..

في التمارين في الصفحات السابقة, نحن نواصل الإستغلال دون أن نقول أن هذه الإمكانيات توفرها البايثون . لكن أنظر الأن كيفية حفظ النصوص من خلال تطبيق ترميز نصوص مختلفة عن تلك التي تقدم إفتراضيا, لن يكون ذلك سوى لضمان أن الترميز الذي نريده (يجب علينا المضي قدما إذا كنا نريد أن تعمل سكريبتاتنا على أنظمة تشغيل مختلفة) .

---

[52] في الإصدارات السابقة للبايثون, يجب دائما على السلاسل النصية أن يتم تحويلها إلى سلسلة من البايتات قبل حفظها . و النوع السابق string هو ما يعادل نوع bytes الحالي, و يتم تحويل أيت ملفات تلقائية عند عمليات قراءة\كتابة الملفات .

التقنية بسيطة . تشير فقط الترميز إلى **open()** بمساعدة برامترات ضافية : **encoding="**المعيار الذي أخترته", على سبيل المثال, يمكننا أن نكرر هذه التمارين من الصفحات السابقة و لكن يجب علينا هذه المرة إستخدام الترميز Latin-1 ::

```
>>> chaine ="Amélie et Eugène\n"
>>> of =open("test.txt", "w", encoding ="Latin-1")
>>> of.write(chaine)
17
>>> of.close()
>>> of =open("test.txt", "rb")
>>> octets =of.read()
>>> of.close()
>>> print(octets)
b'Am\xe9lie et Eug\xe8ne\n'
```

...إلخ.

> يمكنك تنفيذ إختبارات مختلفة على هذه السلسلة من البيتات, إذا كنت ترغب في ذلك .

نفس الشيئ عندما نفتح ملف للقراءة . إفتراضيا, يقوم البايثون بفتح الملف وفق للمعيار الإفتراض لنظام التشغيل . لكن هذا غير واضح من أنه مؤكد . على سبيل المثال دعونا نعيد فتح الملف (بدون مبالات) **test.txt** الذي قمنا بصنعه في الخطوات السابقة :

```
>>> of =open("test.txt", "r")
>>> ch_lue =of.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib/python3.1/codecs.py", line 300, in decode
    (result, consumed) = self._buffer_decode(data, self.errors, final)
UnicodeDecodeError: 'utf8' codec can't decode bytes in position 2-4:
invalid data
```

رسالة الخطأ واضحة : تم إفتراض أن الملف تم ترميزه ب Utf-8, و البايثون لا يمكنه فك ترميزه[53]. سوف يعمل إذا قمنا بما يلي :

```
>>> of =open("test.txt", "r", encoding ="Latin-1")
>>> ch_lue =of.read()
>>> of.close()
>>> ch_lue
```

[53] في المعلوماتية, نسمي codec (ترميز\فك ترميز) أي تحويل شكل . سوف ترى على سبيل المثال العديد من codecs (ترميزات) في عالم المعلوماتية (ترميز الصوت, الفيديو) . و البايثون لديه العديد من برامج تحويل السلاسل النصية لتحويل السلاسل وفقا لمعايير مختلفة .

```
'Amélie et Eugène\n'
```

في السكريبتات المتقدمة, فإنه من المحتمل دائما تحديد الترميز المفترض للملفات معالجته, حتى لو كان الطلب من المستخدم هذه المعلومات, أو فكر في تجارب متقدمة كثيرا أو قليلا لتحديد تلقائيا نوع الترميز .

حالة السكريبتات البايثون

سكريبتات البايثون هي في حد ذاتها ملفات نصية, بالطبع . بناءً على تكوين برنامج تحرير الخاص بك, أو على نظام تشغيل الخاص بك, يمكن لهذه النصوص أن يتم ترميزها بمعاير مختلفة . بحيث يمكن للبايثون تفسيره بشكل صحيح, و ينصح لك دائما أن تشمل شبه التعليق هذا (يجب أن يكون في السطر الأول أو الثاني) :

```
# -*- coding:Latin-1 -*-
```

أو :

```
# -*- coding:Utf-8 -*-
```

... هذا يدل على الترميز المستخدم فعليا, بطبيعة الحال !
و بالتالي مفسر البايثون يعرف كيف يفك السلاسل النصية التي إستخدمتها في السكريبت . لاحظ أنه يمكنك حذف هذا الشبه تعليق إذا كنت متأكد أنه يتم ترميز النصوص الخاص بك بترميز UTF-8, لأنه هو الأن المعيار الإفتراضي لنصوص البايثون[54].

الوصول إلى حروف أخرى غير الموجودة على لوحة المفاتيح

دعونا نرى أي جزء يمكنك أن تستخرجه في الحقيقة جميع الحروف لديها معرف رقمي عالمي يونيكود . للوصول إلى هذه المعرفات, يوفر لك البايثون عددا من الدالات المعرفة مسبقا .

الدالة ord(ch) تقبل أي حرف كبرامتر . و ترجع القيمة الرقمية للحرف . مثلا ord("A") تقوم بإرجاع العدد 65, و ord("Ā") تقوم بإرجاع العدد 296 .

---

[54] في الإصدارات السابقة للبايثون, الترميز الإفتراضي هو ASCII .

و الدالة chr(num) تقوم بالعكس تماما, تعطيها الحرف المطبعي و هي تقوم بإرجاع معرف اليونيكود الذي يساوي الرقم . و لتعمل هذه, يتطلب هذا الإستيفاء بشرطين :

* قيمة num يجب أن تكون لحرف موجود مسبقا (معرفات اليونيكود ليست مستمرة : بعض الرموز لا تتطابق مع أي حرف)

يجب على حاسوبك أن يعرف وصف الغرافيكي للحرف, أو, بطريقة أخر, يعرف رسم هذا الحرف, و هذا يسمى "glyphe" - "حرف رسومي" . أنظمة التشغيل الحديثة تحتوى على مكتبات كبيرة للحروف الرسومية, و التي ينبغي لها أن تعرض الالاف على الشاشة .

على سبيل المثال char(65) تقوم بإرجاع الحرف A و chr(1046) يرجع الحرف السيريلي Ж .

يمكنك إستخدام هذه الدالات المعرفة مسبقا للهو بإستكشاف لعبة إظهار الأحرف على الشاشة الحاسوب . يمكنك على سبيل المثال أن تقوم بإرجاع الحروف الصغيرة للأبجدية اليونانية, مع العلم أن الرموز المخصصة لها تتراوح ما بين 945 إلى 969 . أنظر للسكربت بالأسفل :

```
s = ""                      # سلسلة فارغة
i = 945                     # أول كود
while i <= 969:             # اخر كود
    s += chr(i)
    i = i + 1
print("Alphabet grec (minuscule) : ", s)
```

يجب عليه أن يظهر النتيجة التالية :

```
Alphabet grec (minuscule) : αβγδεζηθικλμνξοπρςστυφχψω
```

تمارين

10.14 أكتب سكريبت صغير الذي يجب عليه أن يظهر جدول رموز ASCII . يجب على البرنامج أن يقوم بإظهار جميع الحروف و الرموز . بداية من هذا حدد العلاقة الرقمية البسيطة بين كل حرف كبير و بين كل حرف صغير يقابله .

10.15 عدل السكريبت السابق لإستكشاف الرموز ما بين 128 و 256, حيث ستجد حروف معلمة (من بين أمور كثيرة) . هل العلاقة العددية التي وجدتها في التمرين السابق ستبقى نفسها للحروف المعلمة الفرنسية ؟

10.16 بداية من هذه العلاقة, أكتب دالة التي تحول جميع الحروف الصغيرة إلى حروف كبيرة, و العكس بالعكس (مقدمة في الجملة كبرامتر).

10.17أكتب سكريبت الذي يقوم بنسخ ملف نصي بإستبدال جميع الفراغات(المساحات الفارغة) بهذه المجموعة من الرموز -*- . الملف الذي ستنسخه يجب أن يكون ترميزه بمعيار Latin-1, و الملف الهدف سيكون ترميزه Utf-8 . إسم الملفين سيتم طلبهم في بداية السكريبت .

10.18 أكتب الدالة voyelle(car), التي تقوم بإرجاع "صحيح" إذا كان الحرف في البرامتر فوايال .

10.19 أكتب الدالة compteVoyelles(phrase), التي تقوم بإرجاع عدد الحروف الفوايال في الجملة المقدمة .

10.20 إكتشف نطاق (مدى) حروف اليونيكود الموجودة في حاسوبك, بمساعدة حلقة برمجية مشابهة للتي قمنا بإستخدامها لإظهار الأبجدية اليونانية . و إعثر على رموز المقابلة للحروف السيريلية, و أكتب سكريبت الذي يظهرها بالكبير و الصغير .

السلاسل هي كائنات

في الفصول السابقة, لقد تعرفت إلى العديد من الكائنات . و أنت تعرف أنه يمكننا أن نعمل على كائن بمساعدة الأساليب (هذا معناه الدالات مرتبطة بهذا الكائن) .

في البايثون, سلاسل النصية هي كائنات . لذلك يمكننا القيام بمعالجة السلاسل النصية بإستخدام الأساليب الملائمة . و في ما يلي بعضها, لقد إخترنا الأكثر إستخداما[55] :

- **split()** : تحويل سلسلة إلى قائمة . يمكننا إختيار الحرف الذي يفصلها (يتم وضعه كبرامتر), فإذا لم نضع سيكون الفراغ(مساحة فراغة) هي الإفتراضية :

```
>>> c2 ="Votez pour moi"
>>> a = c2.split()
>>> print(a)
['Votez', 'pour', 'moi']
>>> c4 ="Cet exemple, parmi d'autres, peut encore servir"
```

55هذه سوى أمثلة قليلة . و أغلب هذه الأساليب يمكن أن يتم إستخدمها مع برامترات مختلفة التي لم نحدد جميعها هنا (على سبيل المثال, بعض البرامترات لا تسمح بمعالجة سوى جزء من السلسلة) . يمكنك الحصول على قائمة كاملة من جميع الأساليب المرتبطة بكائن لإستخدام الدالة المدمجة dir() . يرجى الرجوع إلى أي كتاب مرجعي (أو وثائق على الإنترنات للبايثون) إذا كنت تريد أن تعرف أكثر من ذلك .

```
>>> c4.split(",")
['Cet exemple', " parmi d'autres", ' peut encore servir']
```

- **join(liste)** : جمع قائمة نصية إلى واحدة (هذا الأسلوب يعمل عكس الأسلوب السابق) . تنبيه : السلسلة التي نطبق عليها هذا الأسلوب ستكون بمثابة فاصلة(واحدة أو أكثر) ; البرامتر الممرر سيكون قائمة نصية التي نريد جمعها :

```
>>> b =["Bête", "à", "manger", "du", "foin"]
> >> print(" ".join(b))
Bête à manger du foin
>>> print("---".join(b))
Bête---à---manger---du---foin
```

- **find(sch)** : البحث عن مكان كلمة sch في سلسلة :

```
>>> ch1 = "Cette leçon vaut bien un fromage, sans doute ?"
>>> ch2 = "fromage"
>>> print(ch1.find(ch2))
25
```

- **count(sch)** : عدد تكرار الكلمة sch في سلسلة :

```
>>> ch1 = "Le héron au long bec emmanché d'un long cou"
>>> ch2 = 'long'
>>> print(ch1.count(ch2))
2
```

- **lower()** : تحويل سلسلة إلى حروف صغيرة :

```
>>> ch = "CÉLIMÈNE est un prénom ancien"
>>> print(ch.lower())
célimène est un prénom ancien
```

- **upper()** : تحول سلسلة إلى حروف كبيرة :

```
>>> ch = "Maître Jean-Noël Hébèrt"
>>> print(ch.upper())
MAÎTRE JEAN-NOËL HÉBÈRT
```

- **title()** : تحويل أول حرف في كل كلمة إلى حرف كبير (مثل العناوين الإنكليزية):

```
>>> ch ="albert René élise véronique"
>>> print(ch.title())
Albert René Élise Véronique
```

- **capitalize()** : بديل الأسلوب السابق . يحول فقط أول حرف في السلسلة إلى حرف كبير :

```
>>> b3 = "quel beau temps, aujourd’hui !"
>>> print(b3.capitalize())
"Quel beau temps, aujourd’hui !"
```

- **swapcase()** : يحول جميع الحروف الكبيرة إلى صغير و العكس بالعكس :

```
>>> ch = "Le Lièvre Et La Tortue"
>>> print(ch.swapcase())
lE lIÈVRE eT lA tORTUE
```

- **strip()** : حذف الفراغات في بداية و نهاية السلسلة :

```
>>> ch = "   Monty Python   "
>>> ch.strip()
'Monty Python'
```

- **replace(c1, c2) :** إستبدال جميع الحروف c1 بحروف c2 في السلسلة :

```
>>> ch8 = "Si ce n'est toi c'est donc ton frère"
>>> print(ch8.replace(" ","*"))
Si*ce*n'est*toi*c'est*donc*ton*frère
```

- **index(car)** : إيجاد مؤشر أول ظهور للحرف car في السلسلة :

```
>>> ch9 ="Portez ce vieux whisky au juge blond qui fume"
>>> print(ch9.index("w"))
16
```

في معظم هذه الأساليب, يكون من الممكن تحديد جزء ما يجب معالجته, و هذا ون بإضافة برامترات إضافية . على سبيل المثال :

```
>>> print (ch9.index("e"))        # البحث من البداية السلسلة
4                                 # e و العثور على أول حرف
>>> print (ch9.index("e",5))      # إبدأ فقط من المؤشر 5
8                                 # الثانية e و جد
>>> print (ch9.index("e",15))     # إبدأ البحث بداية من المؤشر 15
29                                # و جد رابع 4
```

... إلخ

أرجو منك أن تفهم أنه لا يمكن وصف كل الأساليب المتاحة, و البرامترات الخاصة بها, في إطار هذه الدورة . فإذا أردت المزيد من المعلومات, فيجب عليك قراءة وثائق البايثون (Library reference), أو مراجع جيدة .

دالات مدمجة

لجميع الأغراض العملية, تذكر أيضا أنه يمكننا أيضا أن نطبق على السلاسل عدد كبير من الدالات المدمجة في اللغة :

- **len(ch)** إرجاع طول السلسلة ch, أو بعبارة أخر, عدد أحرفها .
- **float(ch)** تحويل السلسلة ch إلى رقم حقيقي (float) (و بطبيعة الحال فإن هذا لا يعمل إلا إذا كانت السلسلة رقما, حقيقي أو صحيح)

```
>>> a = float("12.36")        # إنتبه : ليس الفاصلة العشرية
>>> print a + 5
17.36
```

- **int(ch)** يحويل السلسلة ch إلى عدد صحيح (مع نفس القيود)

```
>>> a = int("184")
>>> print a + 20
204
```

- **str(obj)** يحول أو (يمثل) كائن obj إلى سلسلة نصية . و obj يمكن أن يكون معطيات من أي نوع :

```
>>>> a, b = 17, ["Émile", 7.65]
>>> ch =str(a) +" est un entier et " +str(b) +" est une liste."
>>> print(ch)
17 est un entier et ['Émile', 7.65] est une liste.
```

تنسيق السلاسل النصية

لإكمال هذه النظرة النظرة العامة على المميزات المرتبطة بالسلاسل النصية, يبدوا من الحكمة أن أقدم لك تقنية معالجة أكثر قوة, تسمى "تنسيق السلاسل" . هذا مفيد جدا في جميع الحالات التي تحتاج بناء سلسلة معقدة من عدة قطع, مثل قيم المتغيرات المختلفة .

على سبيل المثال, أكتب برنامج الذي يعالج لون و درجة حرارة محلول مائي, في كيمياء, يتم تخزين اللون في سلسلة نصية تدعى **coul**, و درجة الحرارة في متغير من نوع حقيقي يدعى **temp** . و يجب على برنامجك أن يقوم ببناء سلسلة نصية من هذه البيانات, على سبيل المثال, جملة مثل هذه : "المحلول سيكون أحمر و درجة الحرارته 12.7 درجة مئوية " .

يمكنك بناء هذه السلسلة بجمع القطع مع بعضها البعض بمساعدة المعامل التسلسل (الرمز +), و لكن يجب عليك أيضا إستخدام الدالة المدمجة **str()** لتحويل سلسلة نصية و القيمة الرقمية موجودة داخل متغير من نوع float (قم بالتمارين) .

يوفر لك البايثون إمكانيات أخرى . يمكنك صنع سلسلة ("patron" - الزعيم) التي تحتوي على أغلب النص الذي لم يتغير, مع علامات المواقع المحددة (حقول) حيث تظهر عندما تريد محتويات المتغيرات . قم الأن بتطبيق الأسلوب **format()** على هذه السلسلة, و سوف توفرها على شكل

برامترات و الكائنات المختلفة تحول إلى حروف و يتم إضافتها بدل العلامات . مثال أفضل من كل هذا :

```
>>> coul ="verte"
>>> temp =1.347 + 15.9
>>> ch ="La couleur est {} et la température vaut {} °C"
>>> print(ch.format(coul, temp))
La couleur est verte et la température vaut 17.247 °C
```

- تستخدم العلامات المتكونة من الأقواس, التي تحتوي أو قد لا تحتوي على معلومات التنسيق :
- إذا كانت العلامات فارغة (في أبسط الحالات), سوف يتحصل الأسلوب **format()** على برامترات التي ستكون بمثابة العلامات في السلسلة . و سوف يقوم البايثون إذا بتطبيق الدالة **str()** على كل من هذه البرامترات, و سوف يضفها إذا في السلسلة في مكان العلامات, في نفس الترتيب . البرامترات يمكن أن تكون أي كائن أو تعبير بايثون :

```
>>> pi =3.1416
>>> r =4.7
>>> ch ="L'aire d'un disque de rayon {} est égale à {}."
>>> print(ch.format(r, pi * r**2))
L'aire d'un disque de rayon 4.7 est égale à 69.397944.
```

- يمكن للعلامات أن تحتوي على أرقام متسلسلة (العد يبدأ من الرقم 0) لوصف بدقة البرامترات التي ستمرر ل format() لتحل مكانها . هذه التقنية هي قيمة خاصة إذا كانت نفس البرامتر يجب عليه إستبدال مجموعة من العلامات المحددة :

```
>>> phrase ="Le{0} chien{0} aboie{1} et le{0} chat{0} miaule{1}."
>>> print(phrase.format("", ""))
Le chien aboie et le chat miaule.
>>> print(phrase.format("s", "nt"))
Les chiens aboient et les chats miaulent.
```

- يمكن للعلامات أن تحتوي على معلومات تنسيق (بالإشتراك أو ليس مع تسلسل الأرقام) . على سبيل المثال, يمكنك تحديد بدقة النتيجة النهائية أو إجبار إستخدام الرموز العلمية أو تحديد عدد الأحرف, ... إلخ :

```
>>> ch ="L'aire d'un disque de rayon {} est égale à {:8.2f}."
>>> print(ch.format(r, pi * r**2))
L'aire d'un disque de rayon 4.7 est égale à    69.40.
>>> ch ="L'aire d'un disque de rayon {0} est égale à {1:6.2e}."
```

```
>>> print(ch.format(r, pi * r**2))
L'aire d'un disque de rayon 4.7 est égale à 6.94e+01.
```

في الإختبار الأول, النتيجة تم تنسيقها بطريق لتحمل 8 أحرف, رقمين منهم بعد الفاصل . في التجربة الثانية, تم عرض النتيجة في شكل علمي (**e+01** معناها **x 10^01**) . ملاحظة : يتم تنفيذ التقريبات المحتملة بشكل صحيح .

الوصف الكامل لجميع إمكانيات التنسيق يجب أن تكون في العديد من الصفحات, و هذا أكبر من حجم نطاق الكتاب . فإذا كنت في حاجة لمعرفة المزيد حول التنسيق, فيجب عليك الإطلاع على وثائق لغة البايثون, أو الكتب الأكثر تخصصا . ملاحظة بسيطة : هذا التنسيق يسمح بإظهار بسهولة النتيجة الرقمية بالترقيم الثنائي (بيناري) أو الثماني أو الست العشري :

```
>>> n =789
>>> txt ="Le nombre {0:d} (décimal) vaut {0:x} en hexadécimal et {0:b} en binaire."
>>> print(txt.format(n))
Le nombre 789 (décimal) vaut 315 en hexadécimal et 1100010101 en binaire.
```

سلاسل التنسيق " القديمة "

إصدارات البايثون قبل الإصدار 3 كانت تستخدم تقنيات تنسق مختلفة قليلا و أقل تطورا, لكن لا تزال صالحة للإستخدام . و أنصحك بأن تعتمد الطريقة التي ذكرناها في الفقرات السابقة . و سوف نشرح هنا بإختصار الطريقة القديمة, لأمك قد تواجهها في سكريبتات الكثير من المبرمجين (و حتى في بعض أمثلتنا !). و تتكون في التنسيق السلسلة بجمع عنصرين بمساعدة المعامل % . على يسار هذا المعامل, السلسلة "الرئيسة" التي تحتوي على علامات تبدأ دائما ب %, و في اليمين (بين قوسين) أين يضع البايثون الكائن في السلسلة, بدلا من العلامات .

على سبيل المثال :

```
>>> coul ="verte"
>>> temp = 1.347 + 15.9
>>> print ("La couleur est %s et la température vaut %s °C" % (coul, temp))
La couleur est verte et la température vaut 17.247 °C
```

العلامة **% s** تلعب نفس دور {} في الطريقة الجديدة . و تقبل أي كائن (سلسلة, عدد صحيح, عدد حقيقي, قائمة ...) . و يمكنك إستخدام علامات أخرى أكثر تقدما, مثل 8.2f % أو 6.2e % التي هي

مثل {:8.2f} و {:6.2e} في الطريقة الجديدة . و هذا في أبسط الحالات, لكن إقتنع أن إمكانيات الصيغة الجديدة هي واسعة .

تمارين

10.21 أكتب سكريبت الذي يقوم بنسخ ملف نصي تم ترميزه ب Latin-1 إلى Utf-8, و يجب أن تكون كل كلمة تبدأ بحرف كبير . سوف يقوم البرنامج بطلب أسماء الملفات من المستخدم . المعاملات القراءة و الكتابة للملفات في وضع الوضع الملف النصي العادي .

10.22 بديل التمرين السابق : فعّل عمليات قراءة و كتابة الملفات في وضع الثنائي, و العمليات ترميز\فك ترميز سلاسل البايتات . بالإضافة, يجب عليك التعامل مع الأسطر بطريقة لإستبدال جميع الفراغات بمجموعة من 3 رموز **-*-** .

10.23 أكتب سكريبت الذي يقوم بحساب عدد الكلمات الموجودة في ملف نصي .

10.24 أكتب سكريبت الذي يقوم بنسخ ملف نصي مع دمج (مع السابقة) الأسطر التي لا تبدأ بحرف كبير .

10.25 لديك ملف يحتوي على قيم رقمية . إعتبر أن هذه القيم هي أقطار من سلسلة من الكرات . أكتب سكريبت الذي يقوم بإستخدام معطيات هذا الملف لصنع ملف أخر, منظمت في أسطر من النصوص بوضوح الخصائص الأخرى لهذه الكرات (مساحة الجزء و مساحة الخارجية و الحجم), في جمل مثل هذه :

```
Diam.  46.20 cm Section  1676.39 cm² Surf.  6705.54 cm² Vol.  51632.67 cm³
Diam. 120.00 cm Section 11309.73 cm² Surf. 45238.93 cm² Vol. 904778.68 cm³
Diam.   0.03 cm Section     0.00 cm² Surf.     0.00 cm² Vol.      0.00 cm³
Diam.  13.90 cm Section   151.75 cm² Surf.   606.99 cm² Vol.   1406.19 cm³
Diam.  88.80 cm Section  6193.21 cm² Surf. 24772.84 cm² Vol. 366638.04 cm³
.etc
```

10.26 لديك نحت تصرفك ملف نصي يحتوي على أسطر تمثل قيم رقمية من نوع حقيقي, دون أن تعرض (و تم ترميزها كسلاسل نصية) . أكتب سكريبت الذي يقوم بنسخ هذه القيم في ملف أخر, و تقريبهم بحيث لا تحتوي بعد الفاصل أكثر من رقم واحد, هذا الرقم لا يمكن أن بكون سوى 0 أو 5 (التقريب يجب أن يكون صحيحا) .

النقطة في القوائم

لقد التقينا القوائم بالفعل عدة مرات, منذ تقديمه بإختصار في الفصل 5 . القوائم هي مجموعات مرتبة من الكائنات . مثل السلاسل النصية, القوائم هي مجموعة من جزء من نوع عام الذي سميه في البايثون التسلسل . مثل الحروف في السلسلة, الكائنات تم وضعها في القائمة من خلال الفهرس (رقم الذي يشير إلى مكان الكائن في التسلسل) .

تعريف قائمة - الوصول إلى عناصرها

أنت تعرف بالفعل أنه يتم تحديد القائمة بمساعدة الأقواس المعقوفة (نصف مربع) :

```
>>> nombres = [5, 38, 10, 25]
>>> mots = ["jambon", "fromage", "confiture", "chocolat"]
>>> stuff = [5000, "Brigitte", 3.1416, ["Albert", "René", 1947]]
```

في المثال الأخير أعلاه, قمنا بتجميع عدد صحيح و سلسلة و عدد حقيقي و حتى قائمة, لتذكيركم بأنه يمكن وضع معطيات من أي نوع في القائمة, بما في ذلك القوائم و القواميس و tuples (سوف نناقشها في وقت لاحق) .

للوصول إلى عناصر قائمة, سوف نستخدم نفس الطرق (رقم المؤشر و التقطيع إلى قطع) للوصول إلى الأحرف في سلسلة :

```
>>> print(nombres[2])
10
>>> print(nombres[1:3])
[38, 10]
>>> print(nombres[2:3])
[10]
>>> print(nombres[2:])
[10, 25]
>>> print(nombres[:2])
[5, 38]
>>> print(nombres[-1])
25
>>> print(nombres[-2])
10
```

و ينبغي أن تكون الأمثلة المذكورة أعلاه أن تلفت إنتباهكم إلى أن قطعة (شريحة) من القائمة هي دائما قائمة (حتى لو كانت القطعة (الشريحة) تحتوي على عنصر واحد, كما في المثال الثالث) إذا

يمكن للعنصر الواحد أن يحتوي على معطيات من أي نوع . و سوف نقوم بإستكشاف هذه المميزة طوال هذه الأمثلة القادمة .

القوائم يمكن تغييرها

على عكس السلاسل النصية, القوائم هي تسلسل قابل للتغيير . و هذا سيسمح لنا لاحقا ببناء قوائم كبيرة الحجم, قطعةً قطعة, بطريقة ديناميكية (و هذا معناه بمساعدة أي خوارزمية) . على سبيل المثال :

```
>>> nombres[0] = 17
>>> nombres
[17, 38, 10, 25]
```

في المثال أعلاه, قمنا بإستبدال العنصر الأول من القائمة **nombres**, بإستخدام المعامل [] على يسار علامة المساوات .

فإذا كنت تريد الوصول إلى عنصر في قائمة داخل قائمة أخرى. يكفي أن تشير ببساطة إلى مؤشير بين قوسين (نصف مربع) :

```
>>> stuff[3][1] = "Isabelle"
>>> stuff
[5000, 'Brigitte', 3.1415999999999999, ['Albert', 'Isabelle', 1947]]
```

كما هو الحال بالنسبة لجميع التسلسلات, يجب علينا أن لا ننسى أن الترقيم يبدأ من الصفر . و بالتالي, في المثال أعلاه قمنا بإستبدال العنصر رقم 1 في قائمة, والذي هو العنصر 3 في قائمة أخرى : سلسلة **stuff** .

القوائم هي كائنات

في البايثون, القوائم هي كائنات في حد ذاتها, و يمكنك إذا تطبيق عليها عدد من الأساليب الفعالة بشكل خاص . و هذه بعضها :

```
>>> nombres = [17, 38, 10, 25, 72]
>>> nombres.sort()                    # قم بفرز القائمة
>>> nombres
[10, 17, 25, 38, 72]

>>> nombres.append(12)                # أضف عنصر إلى النهاية
>>> nombres
```

```
[10, 17, 25, 38, 72, 12]

>>> nombres.reverse()                    # أعكس ترتيب العناصر
>>> nombres
[12, 72, 38, 25, 17, 10]

>>> nombres.index(17)                    # جد مؤشر عنصر
4

>>> nombres.remove(38)                   # إحذف عنصر
>>> nombres
[12, 72, 25, 17, 10]
```

و بالإضافة إلى هذه الأساليب, يوجد أيضا التعليمة المدمجة **del** الذي تسمح لك بحذف عنصر أو أكثر من خلال مؤشره (أو مؤشراتهم) :

```
>>> del nombres[2]
>>> nombres
[12, 72, 17, 10]
>>> del nombres[1:3]
>>> nombres
[12, 10]
```

لاحظ الفرق بين الأسلوب **remove()** و التعليمة **del** : التعليمة **del** تعمل مع مؤشر أو شريحة المؤشر, في حين أن **remove()** تبحث عن القيمة (فإذا كان يوجد العديد من العناصر بنفس القيمة يتم مسح الأولى فقط) .

تمارين

10.27 أكتب سكريبت الذي يقوم بإنشاء قائمة من المربعات و المكعبات عددها 20 إلى 40 .

10.28 أكتب السكريبت الذي يقوم بصنع تلقائيا قائمة من المنحنيات ذات زوايا من 0° إلى 90°, في خطوات من 5° . تنبيه : الدالة **sin()** للوحدة math تعتبر أن الزوايا بالراديان (360° = 2 π راديان) .

10.29 أكتب سكريبت الذي يسمح بعرض أول 15 نتيجة لجدول الضرب على, 2, 3, 4, 5, 7, 13, 17, 19 (هذه الأقام سوف يتم وضعها في بداية القائمة) في شكل جدول مشابه للجدول التالي :

2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
3	6	9	12	15	18	21	24	27	30	33	36	39	42	45
5	10	15	20	25	30	35	40	45	50	55	60	65	70	75

إلخ ...

10.30 أنظر للقائمة التالية : ['Jean-Michel', 'Marc', 'Vanessa', 'Anne', 'Maximilien', 'Alexandre-Benoît', 'Louise'] أكتب سكريبت الذي يقوم بعرض على إسم من هذه الأسماء مع عدد الحروف التي تتكون منها .

10.31 لديك قائمة من أي الأعداد صحيحة, بعضها مكرر في أماكن مختلفة في القائمة . أكتب سكريبت الذي يقوم بنسخ هذه القائمة في قائمة أخرى مع حذف التكرار (سيتم فرز القائمة النهائية) .

10.32 أكتب السكريبت الذي يبحث عن الكلمة الأطول في الجملة المقدمة (يجب على المستخدم أن يدخل الجملة حسب إختياره) .

10.33 أكتب سكريبت الذي يقوم بعرض قائمة بكل أيام السنة من مخيلتك, و التي تبدأ بيوم الخميس . السكريبت الخاص بك يستخدم 3 قوائم : قائمة بأيام الأسبوع, قائمة بأسماء الأشهر, و قائمة بعدد الأيام لكل شهر(تجهال السنة الكبيسة) . على سبيل المثال :

jeudi 1 janvier **vendredi 2 janvier** **samedi 3 janvier** **dimanche 4 janvier**

و هكذا إلى يوم 31 ديسمبر (كانون الأول)

10.34 لديك ملف نصي الذي يحتوي على أسماء التلاميذ . أكتب سكريبت الذي يقوم بنسخة مرتبة من هذا الملف .

10.35 أكتب دالة التي تقوم بفرز قائمة . هذه الدالة لا يجب عليها أن تستخدم الأسلوم المدمج **sort()** الخاص بالبايثون : لذا يجب عليك أن تقوم بكتابة بنفسك خوارزمية الفرز .

تقنيات تقطيع متقدم للتعديل على قائمة

كما لاحظنا للتو, يمكنك إضافة أو حذف عناصر في قائمة بإستخدام التعليمة (**del**) و الأسلوب (**()append**) المدمج . فإذا كان لايزال لديك فهم أساسي "للتقطيع إلى شرائح", يمكنك إذا الحصول على نفس النتائج بمساعدة معامل واحد [] . إستخدام هذا المعامل هو أكثر عرضة للتلف من التعليمات أو الأساليب المخصصة, لكنه يسمح بمزيد من المرونة :

إدخال عنصر أو أكثر في أي مكان في القائمة

```
>>> mots = ['jambon', 'fromage', 'confiture', 'chocolat']
>>> mots[2:2] =["miel"]
>>> mots
```

```
['jambon', 'fromage', 'miel', 'confiture', 'chocolat']

>>> mots[5:5] =['saucisson', 'ketchup']
>>> mots
['jambon', 'fromage', 'miel', 'confiture', 'chocolat', 'saucisson', 'ketchup']
```

لإستخدام هذه التقنية, يجب عليك أن تعرف هذه المميزات :

* إذا إستخدمت المعامل [] على يسار علامة المساوات لإدراج أو حذف عنصر أو عناصر في قائمة, يجب عليك أن تشير إلى "الشريحة" في القائمة المستهدفة (و هذا معناه مؤشرين الذين جمعتهما بإستخدام الرمز :), و ليس عنصر واحد في هذه القائمة .

* يجب على العنصر الذي على يمين علامة المساوات أن يكون قائمة . فإذا لم تدرج سوى عنصر واحد, يجب عليك إذا تقديمه بين معقفين لتحويل أولا إلى سلسلة بعنصر واحد . لاحظ أن العنصر **mots[1]** ليس قائمة (هو سلسلة "fromage"), إذا العنصر **mots[1:3]** في واحدة .

سوف تفهم بشكل أفضل من خلال تحليل ما يلي :

إزالة \ إستبدال عناصر

```
>>> mots[2:5] = []                # [] يدل على قائمة فارغة
>>> mots
['jambon','fromage','saucisson', 'ketchup']

>>> mots[1:3] = ['salade']
>>> mots
['jambon', 'salade', 'ketchup']

>>> mots[1:] = ['mayonnaise', 'poulet', 'tomate']
>>> mots
['jambon', 'mayonnaise', 'poulet', 'tomate']
```

* في السطر الأول من مالنا, قمنا بإستبدال الشريحة [2:5] بقائمة فارغة, و الذي يتوافق مع الذي حذفناه .

* في السطر الرابع, قمنا بإستبدال شريحة بعنصر واحد . لاحظ مرة أخرى أن هذا العنصر هو في حد ذاته "يعرض" على شكل قائمة .

* في السطر السابع, قمنا بإستبدال الشريحة بها عنصريبن بأخرى بها

إنشاء قائمة من الأرقام بمساعدة الدالة range()

إذا كان يجب عليك التعامل مع سلاسل من الأرقام, يمكنك إنشاءها بسهولة بمساعدة هذه الدالة المدمجة . فهي تقوم بإرجاع سلسلة من الأعداد الصحيحة[56] التي يمكنك إستخدامها مباشرة, أو تحويلها إلى سلسلة عن طريق الدالة **list()**, أو تحويلها إلى tuple بمساعدة الدالة **tuple()** (سوف نقوم بشرح ال tuples في وقت لاحق) :

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

الدالة **range()** تقوم بتوليد إفتراضيا سلسلة من الأعداد الصحيحة بشكل متزايد, و مختلفة بعدد واحد . فإذا إستدعيت **range()** مع برامتر واحد, القائمة ستحتوي على عدد الأرقام القيم المساوية للبرامتر المقدم, لكنها تبدأ من الرقم صفر (و هذا معناه أن **range(n)** تسوف تقوم بتوليد الأرقام من 0 إلى (n-1) . لاحظ أن البرامتر المقدم لن يكون في السلسلة .

يمكننا إستخدام **range()** مع برامترين أو 4 برامترات مفصولة بفواصل, لتوليد متواليات من أعداد أكثر تحديدا :

```
>>> list(range(5,13))
[5, 6, 7, 8, 9, 10, 11, 12]
>>> list(range(3,16,3))
[3, 6, 9, 12, 15]
```

فإذا كان لديك صعوبة في إستيعاب المثال أعلاه, أفترض أن **range()** تنتظر منك دئاما 3 برامترات, و التي يمكن أن نسميها FROM و TO و STEP . و FROM هي القيمة الأولى لتوليدها, TO هي الأخيرة (أو بالأحرى الأخيرة مع واحد), و STEP هي الخطوة للقفز من قيمة لأخرى . فإذا لم نضعها فإن البرامترين FROM و STEP ستكون قيمتهم إفتراضية هي 0 و 1 .

و يسمح بالبرامترات السلبية :

```
>>> list(range(10, -10, -3))
[10, 7, 4, 1, -2, -5, -8]
```

تكرار القائمة بمساعدة for و range() و len()

[56] إن **range()** تعطي في الواقع الوصول إلى المكرر (كائن بايثون مولد لمتسلسلات), و لكن شرحها خارج إطار الذي وضعناه لهذه الكتاب . يرجى الرجوع إلى قائمة مراجع, صفحة Error: Reference source not found, أو وثائق البايثون على الإنترنات إذا كنت تريد التوضيح .

العبارة for هي عبارة المثالية لتكرار قائمة :

```
>>> prov = ['La','raison','du','plus','fort','est','toujours','la','meilleure']
>>> for mot in prov:
...     print(mot, end =' ')
...
La raison du plus fort est toujours la meilleure
```

فإذا كنت تريد تكرار مجموعة من الأعداد الصحيحة, الدالة **range()** ستكون مناسبة :

```
>>> for n in range(10, 18, 3):
...     print(n, n**2, n**3)
...
10 100 1000
13 169 2197
16 256 4096
```

من المريح الجمع بين الدالات **range()** و **len()** للحصول تلقائيا على كافة مؤشرات لتسلسل (قائمة أو سلسلة) . على سبيل المثال :

```
fable = ['Maître','Corbeau','sur','un','arbre','perché']
for index in range(len(fable)):
    print(index, fable[index])
```

تشغيل هذا البرنامج سيظهر لنا :

```
0 Maître
1 Corbeau
2 sur
3 un
4 arbre
5 perché
```

نتيجة هامة من الطباعة الديناميكية

كما لاحظنا سابقا (في الصفحة 125), نوع المتغير المستخدم مع العبارة for سيتم إعادة تعريفه بإستمرار في الدورات : حتى لو كانت عناصر القائمة بأنواع نختلفة, يمكننا تكرار هذه القائمة بمساعدة for بدون أي خطأ, لأن نوع المتغير في الدورات (الحلقات) سوف يتم تعديله تلقائيا إلى نوع العنصر الذي يتم قراءته . على سبيل المثال :

```
>>> divers = [3, 17.25, [5, 'Jean'], 'Linux is not Windoze']
>>> for item in divers:
...     print(item, type(item))
...
3 <class 'int'>
17.25 <class 'float'>
```

```
[5, 'Jean'] <class 'list'>
Linux is not Windoze <class 'str'>
```

في المثال أعلاه, إستخدمنا الدالة المدمجة **type()** لإظهار أن المتغير item يتغير مع كل تكرار (دورة) (و قد أصبح هذا ممكننا من خلال الطباعة الديناميكية للمتغيرات في البايثون).

العمليات على القوائم

يمكننا تطبيق على القوائم المعاملات + (جمع) و * (ضرب) :

```
>>> fruits = ['orange','citron']
>>> legumes = ['poireau','oignon','tomate']
>>> fruits + legumes
['orange', 'citron', 'poireau', 'oignon', 'tomate']
>>> fruits * 3
['orange', 'citron', 'orange', 'citron', 'orange', 'citron']
```

المعامل * مفيد جدا لإنشاء قائمة من n عناصر متطابقة :

```
>>> sept_zeros = [0]*7
>>> sept_zeros
[0, 0, 0, 0, 0, 0, 0]
```

على سبيل المثال, أنت تريد صنع قائمة **B** الذي تحتوي على نفس العدد من العناصر في قائمة A .
يمكنك إنجاز هذا بطرق مختلفة, و لكن أبسطها هي : **B = [0]*len(A)**

إختبار العضوية

يمكنك بسهولة تحديد ما إذا كان العنصر هو جزء من قائمة أو لا بمساعدة العبارة in (هذه العبارة يمكن إستخدامها مع المتسلسلات) :

```
>>> v = 'tomate'
>>> if v in legumes:
...     print('OK')
...
OK
```

نسخ لائحة

نفترض أن لديك قائمة تريد نسخها في متغير جديد يسمى **phrase** . فإن أول فكرة تتبادر إلى ذهنك هي أن تكتب مهمة بسيطة مثل :

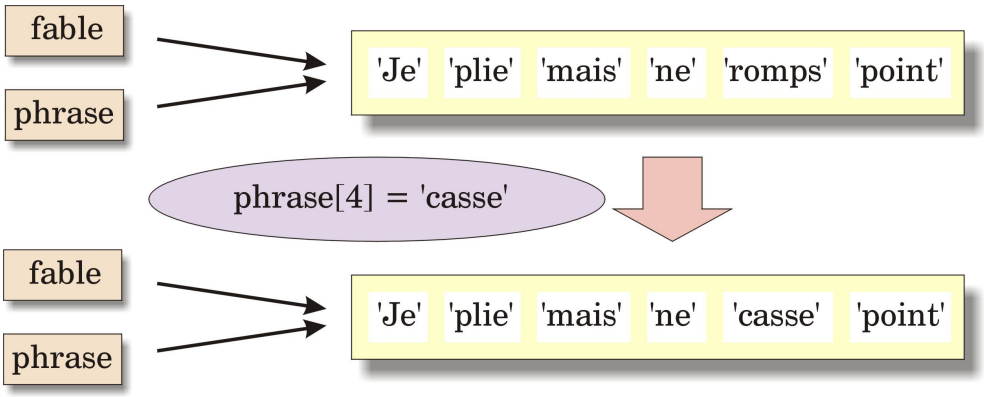
```
>>> phrase = fable
```

من خلال القيام بذلك, إعلم أنك لم تقم بصنع نسخة أصلية . بعد هذه التعليمة, لا توجد سوى قائمة واحدة في ذاكرة الحاسوب . أنت لم تقم سوى بصنع مرجع بسيط لهذه القائمة . حاول على سبيل المثال :

```
>>> fable = ['Je','plie','mais','ne','romps','point']
>>> phrase = fable
>>> fable[4] ='casse'
>>> phrase
['Je', 'plie', 'mais', 'ne', 'casse', 'point']
```

إذا كان المتغير **phrase** يحتوي على نسهة أصلية من القائمة, سوف تكون هذه النسخة مستقلة عن النص الأصلي, و ينبغي ألا يتم تعديلها بتعليمة مثل التي بالسطر الثالث, التي تطبق على المتغير **fable** . يمكنك تجربة المزيد من التغييرات الأخرى, سواء على محتويات **fable**, أو على محتويات **phrase**, في جميع الأحوال, سوف تجد أن التعديلات عدلت أيضا على الأخرى, و العكس بالعكس .

في الواقع, **fable** و **phrase** تم تعيين كلاهما في كائن واحد في الذاكرة . لوصف هذه الحالة, يقول علماء الحاسوب أن **phrase** هو إسم مستعار ل **fable** .



سوف نرى لاحقا, إستخدام الإسم المستعار . أما الأن, سوف نتعلم تقنية لإجراء نسخة أصلية(فعلية) لقائمة . مع المفاهيم المذكورة أعلاه, يجب أن تكون قادر على إيجاد واحدة بنفسك .

ملاحظة بسيطة حول تركيب الجملة

البايثون يسمح لك "بتوسيع" تعليمة طويلة على عدة أسطر, فإذا إستمررت بترميز شيئا ما محدد بواسطة زوج من الأقواس, أو المعقفين, أو قوسين(نصف مربع) . يمكنك معالجة العبارات بين قوسين, أو تعريف قوائم طويلة, أو tuples كبيرة أو قواميس كبيرة(أنظر أدناه) . مستوى مسافة البادئة غير مهم : المفسر يكشف عن نهاية العبارة حيث يتم إغلاق زوج المركب .

هذه الميزة تسمح لك بتحسين إمكانية قراءة برامجك . على سبيل المثال :

```
couleurs = ['noir', 'brun', 'rouge',
            'orange', 'jaune', 'vert',
            'bleu', 'violet', 'gris', 'blanc']
```

تمارين

10.36 أنظر في القوائم التالية :

```
t1 = [31,28,31,30,31,30,31,31,30,31,30,31]
t2 = ['Janvier','Février','Mars','Avril','Mai','Juin',
      'Juillet','Août','Septembre','Octobre','Novembre','Décembre']
```

أكتب برنامج صغير الذي يقوم بإدراج في القائمة الثانية جميع عناصر القائمة الأولى, بحيث يتم فرز كل إسم شهر مع عدد أيامه : `['Janvier',31,'Février',28,'Mars',31, etc.]`.

10.37 إصنع قائمة **A** تحتوي على بضعة عناصر . قم بعمل نسخة منها في متغير جديد **B** . إقتراح : قم أول بصنع القائمة **B** بنفس طول القائمة **A** لكنها لا تحتوي سوى على أصفار . ثم قم بإستبدال كل الأصفار بعناصر من القائمة **A** .

10.38نفس السؤال, لكن إقتراح أخر : قم بصنع أولا القائمة **B** فارغة . ثم قم بملئها بالعناصر القائمة **A** واحدة تلو الأخرى .

10.39 نفس السؤال, لكن إقتراح أخر : لصنع القائمة **B**, قم بقص من القائمة **A** شريحة تحتوي على كل العناصر (بمساعدة المعامل **[:]**) .

10.40 الرقم الأولي هو الرقم الذي يقبل القسمة على نفسه و على واحد . أكتب برنامج الذي يقوم بعرض جميع الأرقام الأولية ما بين 1 و 1000, بإستخدام أسلوب غربال إراتونسثين :

* أصنع قائمة من 1000 عنصر, قم بتهيئة كل عنصر على القيمة 1 .

* إستعرض هذه القائمة بداية من العنصر الثاني : إذا كان العنصر حلل لديه القيمة 1, ضع 0 لجميع العناصر الباقية, و التي هي مؤشرات مضعفات العدد الصحيح للمؤشر الذي حصلت عليه .

عندما تستعرض جميع القائمة, مؤشرات العناصر التي بقية 1 سيكونوا الأرقام الأولية لتي نبحث عنها . في الحقيقة : بداية من المؤشر 2, ستلغي جميع العناصر الزوجية : 4, 6, 8, 10 ...إلخ . مع المؤشر 3, سوف تقوم بإلغاء جميع العناصر المؤشر 6, 9, 12, 15 ...إلخ, و هكذ . و العناصر التي بقيت 1 هي أرقام أولية .

الأرقام العشوائية - المدرج الإحصائي

معظم البرامج تقوم بعمل الشيئ نفسه في كل مرة تقوم بتشغيلها . وتسمي هذه البرامج بالحتمية (المحددة) . الختمية هي شيئ جيد : بالطبع نحن نريد نفس السلسلة من العمليات الحسابية تطبق على نفس المعطيات تؤدي دائما إلى نفس النتيجة . لبعض التطبيقات, إن الحاسوب صعب تكهنه . على سبيل المثال الألعاب هو مثال واضح, هنالك العديد من الأخرين .

على عكس المظاهر, فإنه ليس من السهل كتابة خوارزمية التي هي حقا غير حتمية (و هذا معناه أن تنتج نتيجة لا يمكن التنبؤ بها) . و مع ذلك, هنالك تقنيات رياضية لمحاكات أكثر أو أقل نتيجة الصدفة . لقد كتبت كتب بالكامل لشرح كيفية إنتاج عشوائي "بنوعية جيدة" . و نحن بالطبع لن نضع سؤال كهذا .

في وحدة random, يقدم لك البايثون مجموعة متنوع من الدالات لتوليد أرقام عشوائية التي تتبع توزيعات رياضية مختلفة . نحن لن نجرب هنا سوى بعضها . إطلع على وثائق البايثون على الشبكة لمعرفة المزيد . يمكنك إستدعاء جميع الدالات في الوحدة ب :

```
>>> from random import *
```

الدالة random لوحدة random تسمح لك بإنشاء أعداد حقيقية عشوائية ذات قيمة مابين 0 و 1 . البرامتر هو حجم القائمة المطلوبة :

```
>>> def list_aleat(n):
...     s = [0]*n
...     for i in range(n):
...         s[i] =random()
...     return s
```

```
...
>>> list_aleat(3)
[0.37584811062278767, 0.03459750519478866, 0.714564337038124]
>>> list_aleat(3)
[0.8151025790264931, 0.3772866844634689, 0.8207328556071652]
```

يمكنك أن ترى أننا أخذنا جزء أولا بناء قائمة من الأصفار بطول n , ثم قمنا بإستبدال الأصفار بأرقام عشوائية .

تمارين

10.41 أعد كتابة الدالة **list_aleat()** في الأعلى, بإستخدام الأسلوب **append()** لصنع قائمة جزءا جزءا بداية من قائمة فارغة (بدل من إستبدال الأصفار من قائمة موجودة سابقا كما فعلنا قبل قليل ..

10.42 أكتب الدالة **imprime_liste()** التي تسمح بإظهار سطرا سطرا جمع عناصر الموجودة في قائمة بأي حجم . إسم القائمة سيكون في البرامتر , إستخدم هذه الدالة لطباعة قائمة من الأرقام العشوائية التي تم صنعها بواسطة الدالة **list_aleat()** . على سبيل المثال التعليمة **((imprime_liste(liste_aleat(8** سوف تقوم بعرض عمود من 8 أرقام حقيقية عشوائية .

هل الأرقام التي تم تولدها هي فعلا عشوائية ؟ هذا الشيئ صعب قوله . نحن لم نفعل سوى لعدد قليل من القيم, لا يمكننا التحقق من هذا . و من جانب أخر, إذا إستخدمنا الدالة random() مرات عديدة, نتوقع أن يكون نصف القيم المنتجة هي أكبر من 0.5 (و النصف الأخر أقل).

نركز على هذا المنطق . القيم التي يتم الحصول عليها هي دائما في نطاق 0 - 1 . مشاركة هذا الفاصل الزمني في 4 أجزاء متساوية : من 0 إلى 0.25 و من 0.25 إلى 0.5 و من 0.5 إلى 0.75 و من 0.75 إلى 1 فإذا وضعنا عدد كبير من القيم العشوائية, نحن نتوقع أنه سيكون هنالك الكثير من الإنخفاض في الكسور الأربعة . و يمكننا تعميم هذا المنطق إلى رقم أي كسر, طالما أنهم متساوون .

تمرين

10.43 أكتب برنامج الذي يتحقق من عمل مولد الأرقام العشوائية في البايثون بإستخدام النظرية المذكورة أعلاه . البرنامج سيقوم بالتالي :

* أطلب من المستخدم عدد القيم العشوائية التي سيتم إنشائها بمساعدة الدالة **()random** . و سيكون من المثير للإهتمام أن يوفر البرنامج عدد إفتراضي (1000 على سبيل المثال) .

* أطلب من المستخدم كم يريد لمشاركة في مجموعة قيم الكسور الممكنة (و هذا معناه ما بين 0-1) . هنا أيضا, يجب أن تقدم عدد الكسور الإفتراضي (5 على سبيل المثال) . يمكنك أن تحدد للمستخدم ما بين 2 و 1\10 عدد القيم العشوائية .

* قم ببناء قائمة من N عدادات (N ستكون عدد من الكسور المطلوبة) . و سيتم تهيئة كل واحدة منهم إلى الصفر .

* إسحب عشوائيا جميع القيم المطلوبة, بمساعدة الدالة **random()**, و قم بتخزين هذه القيم داخل قائمة .

* قم بتدوير قائمة القيم التي تم سحبها عشوائيا(حلقة), و قم بإجراء إختبار لكل واحد منها لتحديد ما هي جزء من فترة الفاصلة 0-1 هي عليها . يزداد العداد واحد واحد .
* عند الإنتهاء, إعرض حالة كل عداد .

مثال على نتائج التي يتم عرضها من برنامج من هذا النوع :

```
Nombre de valeurs à tirer au hasard (défaut = 1000) : 100
Nombre de fractions dans l'intervalle 0-1 (entre 2 et 10, défaut =5) : 5
Tirage au sort des 100 valeurs ...
Comptage des valeurs dans chacune des 5 fractions ...
11 30 25 14 20
Nombre de valeurs à tirer au hasard (défaut = 1000) : 10000
Nombre de fractions dans l'intervalle 0-1 (entre 2 et 1000, défaut =5) : 5
Tirage au sort des 10000 valeurs ...
Comptage des valeurs dans chacune des 5 fractions ...
1970 1972 2061 1935 2062
```

أسلوب جبد لهذه المشكلة من خلال تصور دالات بسيطة لكتابتها لحل جزء أو أخر من المسكلة, ثم إستخدامها لأشياء أكبر .

على سبيل المثال, تستطيع أولا أن تحاول تعريف الدالة **numeroFractio()** التي تحدد أي جزء ما بين 0-1 (قيمةالمستمدة) . هذه الدالة تأخذ برامترين (القيمة المستمدة, عدد الكسور التي يتم إختيارها من قبل المستخدم) و يقوم بإرجاع مؤشر العداد لزيادته (هذا معناه رقم الكسر) . قد يكون

هنالك منطق رياضي بسيط الذي يسمح لك بتحديد المؤشر الكسر من هذان البرامتران . بما في ذلك الدالة المدمجة **int()**, التي تسمح لك بتحويل عدد حقيقي إلى عدد صحيح بإزالة الجزء العشري .

فإذا لم تجدها, فكرة أخرى مثير للإهتمام, إبدأ ببناء قائكة تحتوي على القيم "المحاور" التي تحدد الكسور المحددة (على سبيل المثال 0 – 0,25 – 0,5 – 0,75 - 1 في حالة 4 كسور) . معرفة هذه القيم قد يسهل كتابة الدالة **numeroFraction()** التي نريد أن نطورها .

إذا كان لديك المزيد من الوقت, يمكنك أيضا صنع نسخة رسومية من البرنامج, و التي سوف تعرض لك النتائج على رسم بياني (الرسم البياني "بالعصا")

سحب عشوائيا الأعداد الصحيحة

عندما تطور مشاريعك الشخصية, سوف تحتاج إلى الكثير من الأحيام إلى دالة تسمح لك يسحب عشوائيا عدد صحيح في حدود معينة . على سبيل المثال, فإذا أردت كتابة برنامج للعبة أوراق اللعب التي يتم سحبها عشوائيا (اللعبة العادية 52 ورقة), سوف تستخدم بكل تأكيد دالة التي يمكنها سحب عشوائيا عدد صحيح ما بين 1 و 52 .

يمكنك القيام بهذا عن طريق الدالة **randrange()** للوحدة random . هذه الدالة تستخدم مع 1 أو 2 أو 3 برامترات .

بإستخدام برامتر واحد, تقوم بإرجاع عدد صحيح ما بين 0 و القيمة البرامتر ناقص 1 . على سبيل المثال, **randrange(6)** سوف تقوم بإرجاع عدد ما بين 0 و 5 .

بإستخدام برامترين, العدد الذي سيتم إرجاعه سيكون ما بين البرامتر الأول و البرامتر الثاني ناقص واحد . على سبيل المثال, **randrange(2, 8)** تقوم بإرجاع عدد ما بين 2 و 7 .

و إذا أضفنا برامتر ثالث, فإنه يشير إلى أن العدد الذي يجب سحبه يجب أن يكون من مجموعة من الأعداد محددة من الأعداد الصحيحية, و يتم فصلها عن بعض بفاصل معين, الذ تم تعريفه بالبرامتر الثالث . على سبيل المثال, **randrange(3,13,3)** سوف تقوم بإرجاع عدد من 3, 6, 9, 12 :

```
>>> from random import randrange
>>> for i in range(15):
...     print(randrange(3, 13, 3), end =' ')
```

```
...
12 6 12 3 3 12 12 12 9 3 9 3 9 3 12
```

تمارين

1.3 أكتب سكريبت الذي يقوم بسحب عشوائيا أوراق اللعب . إسم ورقة التي يتم سحبها يجب أن يكون قد عرض بشكل صحيح, "بوضوح" . سيقوم البرنامج بعرض على سبيل المثال :

```
Frappez <Enter> pour tirer une carte :
Dix de Trèfle
Frappez <Enter> pour tirer une carte :
As de Carreau
Frappez <Enter> pour tirer une carte :
Huit de Pique
Frappez <Enter> pour tirer une carte :
```

...إلخ

الأنفاق (tuples)

لقد درسنا حتى الأن نوعين من المعطيات المركبة : السلاسل, و التي هي مركبات حروف, و القوائم, و التي هي مركبات عناصر من أي نوع . و يبج أن تتذكر فرق أخر ما بين السلاسل و القوائم : غير ممكن تغيير الأحرف في سلسلة, إذا يمكنك تعديل(تحرييير) العناصر في سلسلة . و بعبارة أخرة, القوائم هي متسلسلات قابلة للتعديل, و السلاسل النصية هي متسلسلان غير قابلة للتعديل, على سبيل المثال :

```
>>> liste =['jambon','fromage','miel','confiture','chocolat']
>>> liste[1:3] =['salade']
>>> print(liste)
['jambon', 'salade', 'confiture', 'chocolat']

>>> chaine ='Roméo préfère Juliette'
>>> chaine[14:] ='Brigitte'

        ***** ==> Erreur: object doesn't support slice assignment  *****
```

لقد حاولنا تغيير نهاية السلسلة النصية, لكننا لم ننجح . السبيل الوحيد لتحقيق أهدافنا هو بصنع سلسلة جديدة, و نقوم بنسخ ما نريد تغييره :

```
>>> chaine = chaine[:14] +'Brigitte'
>>> print(chaine)
Roméo préfère Brigitte
```

توفر لك البايثون نوع من البيانات يدعى النفق (tuple)[57], و هو يشبه كثيرا القوائم, لكنه مثل السلاسل, لا يمكن تعديله .

من المنظور اللغوي, النفق (tuple) هو مجموعة من العناصر مفصولة بفواصل :

```
>>> tup = 'a', 'b', 'c', 'd', 'e'
>>> print(tup)
('a', 'b', 'c', 'd', 'e')
```

على الرغم من أنه غير ضروري, لكن من المستحسن تحديد النفق (tuple) بزوج من الأقواس, مثل الدالة **print()** في البايثون . هذا ببساطة لزيادة إمكانية قراءة الكود, لكن هذا مهم .

```
>>> tup = ('a', 'b', 'c', 'd', 'e')
```

العمليات على الأنفاق (tuples)

لا يمكن أن تعمل العمليات على الأنفاق (tuple) بناء جملة الأنفاق (tuples) ليس مماثل للقوائم, لأن الأنفاق (tuples) غير قابلة للتغيير :

```
>>> print(tup[2:4])
('c', 'd')
>>> tup[1:3] = ('x', 'y')                ==> ***** erreur ! *****
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> tup = ('André',) + tup[1:]
>>> print(tup)
('André', 'b', 'c', 'd', 'e')
```

لاحظ أن يجب عليك دائما على الأقل فاصلة واحدة لتعريف النفق (tuple) (المثال الأخير فوق يستخدم نفق (tule)يحتوي على عنصر واحد : "**André**") .

يمكنك تحديد طول النفق (tuple) بمساعدة len(), التدوير بمساعدة الحلقة for, إستخدم التعليمة in لمعرفة ما إذا كان العنصر المقدم هو جزء, إلخ ..., تماما مثلما كنت تفعل مع القائمة . عمليات

57هذا المصطلح ليس كلمة إنكليزية عادية : هو لفظ حاسوبي جديد .

الجمع و الضرب تعمل إذا ؟ لكن لأن الأنفاق (tuples) غير قابلة للتغيير (التحرير), لا يمكنط إستخدام مع الأنفاق (tuples) التعليمة del و لا الأسلوب **remove()** :

```
>>> tu1, tu2 = ("a","b"), ("c","d","e")
>>> tu3 = tu1*4 + tu2
>>> tu3
('a', 'b', 'a', 'b', 'a', 'b', 'a', 'b', 'c', 'd', 'e')
>>> for e in tu3:
...     print(e, end=":")
...
a:b:a:b:a:b:a:b:c:d:e:
>>> del tu3[2]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
```

سوف تفهم فائدة إستخدام الأنفاق (tuples) تدريجيا . نلاحظ ببساطة أننا نفضله عن قوائم لان البيانات التي يتم تمريرها لن تتم تعديلها بالخطأ في البرنامج . و بالإضافة إلى ذلك, الأنفاق (tuples) أقل "جشع" على موارد النظام (أي أنه تأخذ مساحة أقل في ذاكرة, و يمكن معالجتها بسرعة من قبل المفسر) .

القواميس

أنواع البيانات المركبة التي ذكرناها حتى الأن هي : السلاسل و القوائم و الأنفاق (tuples) و هم جميعا من المتسلسلات, و هذا معناه تسلسل مرتب من العناصر . في التسلسل, من السهل الوصول إلى أي عنصر من خلال مؤشره (عدد صحيح), لكن شرط معرفة موقعها .

القواميس التي نكتشفها الأن هي نوع أخر من المركبات . و هي تبدو مثل القوائم إلى حد كبير (يمكن تعديلها مثل القوائم), لكنها ليست من المتسلسلات . العناصر التي نقوم بحفظها لن نكون في ترتيب ثابت . و من جهة أخرى, يمكننا الوصول إلى أي وحدة منها (عنصر)بمساعدة مؤشر خاص يسمى المفتاح, و الذي قد يكون من الحروف أو الأرقام أو حتى مركب بشروط معينة .

كما هو الحال في القائمة. العناصر التي يتم تخزينها في القاموس يمكن أن تكون من أي نوع . هذا يعني أنها يمكن أن تكون من القيم الرقمية أو سلاسل أو قوائم أو أنفاق (tuples) أو قواميس أو حتى دالات أو أصناف أو المثيل (سوف نراه لاحقا)[58].

إنشاء قاموس

على سبيل المثال, سوف ننشئ قاموس للغة, لترجمة المصطلحات الحاسوبية من اللغة الإنكليزية إلى اللغة الفرنسية .

و بما أن القواميس قابلة للتعديل, يمكننا البدء بإنشاء قاموس فارغ, ثم نقوم بملئه تدريجيا . من المنظور لغوي (تركيب الجملة), إعلم أن القاموس يتم وضع عناصره في زوج من الأقواس(معقف) . و للإشارة إلى القاموس الفارغ ب {} :

```
>>> dico = {}
>>> dico['computer'] = 'ordinateur'
>>> dico['mouse'] ='souris'
>>> dico['keyboard'] ='clavier'

>>> print(dico)
{'computer': 'ordinateur', 'keyboard': 'clavier', 'mouse': 'souris'}
```

و كما ترى في السطر الأخير أعلاه, يظهر القاموس في البايثون على شكل سلسلة من العناصر مفصولة بفواصل, يتم إحاطة كل هذا بقوسين (معقفين) . كل عنصر من هذه العناصر هو في حد ذاته يشكل زوج من الكائنات : المؤشر و القيمة مفصولة بنقطتين .

في القاموس, يسمى المؤشر بالمفتاح, و العناصر تدعى إذا بزوج من مفتاح القيمة . القاموس في مثالنا, المفاتيح و القيم سلاسل نصية .

يرجى ملاحظة أن ترتيب العناصر الموجود في السطر الأخير لا طابق الترتيب الذي قدمناه .

```
>>> print(dico['mouse'])
souris
```

يرجى ملاحظة أن ترتيب أي عنصر لا يطابق الذي قدمناه سابقا . و هذه ليست لها أي أهمية : و هذا معناه أننا لن نستخرج قيمة أحد عناصر القاموس بمساعدة رقم ترتيبه, بدلا من ذلك نستخدم المفاتيح :

58القوائم و الأنفاق قد تحتوي أيضا على قواميس, و دالات, و أصناف و مثيلات, و نحن لن نذكرها حتى الأن, حتى لا تقيد العرض التقديمي .

نلاحظة أيضا, على عكس القوائم, فإنه ليس من الضروري إستدعا أسلوب معين (مثل **append()**) لإضافة عنصر جديد إلى القاموس : فقط قم بصنع زوج جديد من مفتاح-قيمة .

العمليات على القواميس

أنت تعرف كيفية إضافة عنصر إلى قاموس . لإزالة عنصر, نستخدم التعليمة المدمجة del . إصنع على سبيل المثال قاموس أخر, في هذه المرة يحتوي على مخزون من الفاكهة المؤشرات (أو المفاتيح) سيكونوا أسماء فواكهة, و القيم ستكون كتل هذه الفواكهة المدرجة في المخزون (القيم ستكون هذه المرة قيم رقمية) .

```
>>> invent = {'pommes': 430, 'bananes': 312, 'oranges' : 274, 'poires' : 137}
>>> print(invent)
{'oranges': 274, 'pommes': 430, 'bananes': 312, 'poires': 137}
```

فإذا أراد سيدك تصفية جميع التفاح و لا بيعها, يمكننا إزالة من القاموس :

```
>>> del invent['pommes']
>>> print(invent)
{'oranges': 274, 'bananes': 312, 'poires': 137}
```

الدالة المدمجة **len()** متكاملة مع القواميس : تقوم بإرجاع عدد العناصر :

```
>>> print(len(invent))
3
```

إختبار الإنتماء

مثل ما يحدث للسلاسل و القوائم و الأنفاق (tuples), التعليمة in تعمل مع القواميس . فهي تسمح لم إذا كان القاموس يحتوي على مفتاح محدد[59] :

```
>>> if "pommes" in invent:
...     print("Nous avons des pommes")
... else:
...     print("Pas de pommes. Sorry")
```

[59] في الإصدارات السابقة للبايثون, كان من الضروري إستدعاء أسلوب معين (الأسلوب has_key()) لأداء هذا الإختبار .

```
...
Pas de pommes. Sorry
```

القواميس هي كائنات

يمكننا أن نطبق على القوائم العديد من الأساليب الخاصة :

الأسلوب **keys()** تقوم بإرجاع تسلسل المفاتيح المستخدمة في القاموس . هذا التسلسل يمكن إستخدامه في العبارات أو تحويلها إلى قائمة أو إلى أنفاق (tuples) إذا لزم الأمر, عن طريق دالات المدمجة مثل **list()** أو **tuples()** .

```
>>> print(dico.keys())
dict_keys(['computer', 'mouse', 'keyboard'])
>>> for k in dico.keys():
...     print("clé :", k, " --- valeur :", dico[k])
...
clé : computer  --- valeur : ordinateur
clé : mouse  --- valeur : souris
clé : keyboard  --- valeur : clavier
>>> list(dico.keys())
['computer', 'mouse', 'keyboard']
>>> tuple(dico.keys())
('computer', 'mouse', 'keyboard')
```

و بشكل مماثل, فإن الأسلوب **values()** يقوم بإرجاع سلسلة من القيم المخزنة في القواميس :

```
>>> print(invent.values())
dict_values([274, 312, 137])
```

أما بالنسبة للأسلوب **items()**, فهو يقوم بإستخراج من القاموس سلسلة معادلة للأنفاق (tuples) . و هذا الأسلوب سيكون مفيدا جدا فيما بعد, عندما نريد تكرار قاموس بمساعدة حلقة :

```
>>> invent.items()
dict_items([('poires', 137), ('bananes', 312), ('oranges', 274)])
>>> tuple(invent.items())
(('poires', 137), ('bananes', 312), ('oranges', 274))
```

الأسلوب **copy()** تسمح لك بصنع نسخة طبق الأصل من قاموس . يجب أن تعرف إذا قمت متغير جديد و ربطه بالقاموس سيكون هذا فقط مرجع للكائن نفسه, و ليس كائن جديد . سبق و أن تحدثنا على هذا الأمر في القوائم (أنظر إلى الصفحة 240) . على سبيل المثال, العبارة بالأسفل لا تعرف قاموس جديد (عكس المظاهر) :

```
>>> stock = invent
>>> stock
{'oranges': 274, 'bananes': 312, 'poires': 137}
```

فإذا قمنا بتغيير **invent**, سيتم تغيير **stock** أيضا, و العكس بالعكس (هذان الإسمان يشيران إلى نفس كائن القاموس في ذاكرة الحاسوب) :

```
>>> del invent['bananes']
>>> stock
{'oranges': 274, 'poires': 137}
```

لصنع نسخة حقيقية (مستقلة) لقاموس موجود مسبقا, يجب عليك إستخدام الأسلوب **copy()** :

```
>>> magasin = stock.copy()
>>> magasin['prunes'] = 561
>>> magasin
{'oranges': 274, 'prunes': 561, 'poires': 137}
>>> stock
{'oranges': 274, 'poires': 137}
>>> invent
{'oranges': 274, 'poires': 137}
```

تدوير قاموس

يمكنك إستخدام حلقة التكرار for لمعالجة جميع عناصر في قاموس, لكن إنتبه :

* خلال التكرار, المفاتيح المستخدمة في القاموس هي التي سيتم تعينها تبعا لمتغير العمل, و ليس القيم .

* الترتيب الذي سيتم إستخراج العناصر به لا يمكن التنبؤ به (لأن القاموس ليس متسلسل) .

على سبيل المثال :

```
>>> invent ={"oranges":274, "poires":137, "bananes":312}
>>> for clef in invent:
...     print(clef)
...
poires
bananes
oranges
```

إذا كنت ترغب في أن تقوم بمعالجة على القيم, يكفي أن تقوم بإسترداد كل واحد من المفاتيح المطابقة :

```
>>> for clef in invent:
...     print(clef, invent[clef])
...
poires 137
bananes 312
oranges 274
```

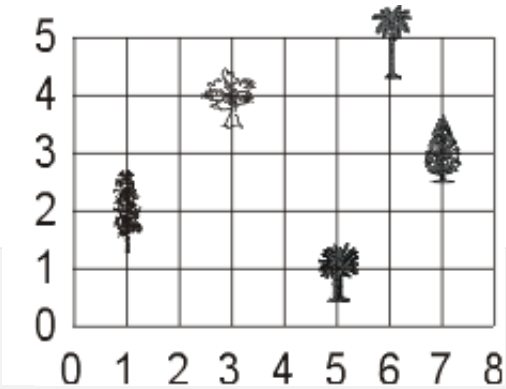
هذا النهج ليس مثاليا, سواء من حيث الأداء أو حتى من وجهة نظر الوضوح . فمن المستحسن بدلا مـن ذلـك إستخدام طريقة items() الذي تم وصفه في القسم السابق :

```
for clef, valeur in invent.items():
    print(clef, valeur)
...
poires 137
bananes 312
oranges 274
```

في هذا المثال, الأسلوب **items()** يتم تطبيقه على قاموس **invent** الذي يقوم بإرجاع سلسلة من الأنفاق (tuples) (المفتاح, القيمة) . يتم تنفيذ هذه الدورة على هذه القائمة بمساعدة حلقة التي تقوم بفحص كل عنصر من عناصر النفق (tuples) .

المفاتيح ليس بالضرورة أن تكون سلاسل نصية

حتى الأن وصفنا أن مفاتيح القواميس هي من نوع سلسلة (string) . في الحقيقة يمكننا إستخدام كمفتاح من أي نوع من البيانات الغير قابلة للتغيير : أعداد صحيحة, أعداد حقيقية, سلاسل نصية, و حتى الأنفاق (tuples) .



على سبيل المثال, نريد أن نتعرف على الأشجار الملحوظة التي توجد في حقل مستطيل كبير . لهذا يمكننا إستخدام قاموس, و مفاتيحه ستكون أنفاق (tuples) التي تشير إلى إحداثيات **X** و **Y** لكل شجرة :

```
>>> arb = {}
>>> arb[(1,2)] = 'Peuplier'
>>> arb[(3,4)] = 'Platane'
Comme
>>> arb[6,5] = 'Palmier'
>>> arb[5,1] = 'Cycas'
>>> arb[7,3] = 'Sapin'

>>> print(arb)
{(3, 4): 'Platane', (6, 5): 'Palmier', (5, 1): 'Cycas', (1, 2): 'Peuplier',
 (7, 3): 'Sapin'}

>>> print(arb[(6,5)])
palmier
```

قد تلاحظ أننا قللنا من الكتابة بداية من السطر الثالث, بالإستفادة من أقواس الأنفاق (tuples) الإختيارية (إستخدمها بحذر !)

في هذا النوع من البناء, نأخذ في إعتبارنا أن القاموس يحتوي فقط على بعض العناصر من أزواج الإحداثيات . و علاوة على ذلك, لا يوجد شيئ . و بالتالي, إن كنا نريد الإستعلام في القاموس لمعرفة مكان غير موجود, على سبيل المثال الإحداثيات (2,1), فإننا نحصل على خطأ :

```
>>> print(arb[1,2])
Peuplier
>>> print(arb[2,1])

          ***** Erreur : KeyError: (2, 1)  *****
```

لحل هذا المشكلة الصغيرة, يمكننا إستخدام الأسلوب **get()** :

```
>>> arb.get((1,2), 'néant')
Peuplier
>>> arb.get((2,1), 'néant')
néant
```

البرامتر الأول الذي يتم تمريره هو لهذا الأسلوب هو مفتاح البحث, أما البرامتر الثاني فهي القيمة التي نريد الحصول عليها إذا كان المفتاح غير موجود في القاموس .

القواميس ليست متسلسلة

كما رأيتم أعلاه, لا يتم ترتيب عناصر القاموس في أي ترتيب معين . العمليات مثل التسلسل و الإستخراج (من مجموعة من العناصر المتصلة) لا يمكن تطبيقها هنا ببساطة . في إذا حاولت في أي حال من الأحوال, سيقوم البايثون بطباعة خطأ عند تشغيل كود :

```
>>> print(arb[1:3])

         ***** Erreur : TypeError: unhashable type *****
```

و رأيتم أيضا أنه يكفي لتعيين مؤشر جديد (مفتاح جديد) لإضافة مدخلات إلى القاموس . و هذه لا تعمل مع القوائم[60] :

```
>>> invent['cerises'] = 987
>>> print(invent)
{'oranges': 274, 'cerises': 987, 'poires': 137}

>>> liste =['jambon', 'salade', 'confiture', 'chocolat']
>>> liste[4] ='salami'

         ***** IndexError: list assignment index out of range  *****
```

60تذكير : الأساليب التي تسمح بإضافة عناصر إلى القائمة تم شرحها في صفحة Error: Reference source not found.

هم ليست من المتسلسلات, ثبت أن القواميس إذا قيم خاصة تم صنعها يتجميع معطيات و التي يتعين على المرء إضافة عليها أو حذف منها, في أي ترتيب كانت . و هي تحل محل القوائم عندما يتعلق الأمر بالتعامل مع مجموعات من المعطيات الرقمية, و التي هم غير متسلسلين .

على سبيل المثال :

```
>>> client = {}
>>> client[4317] = "Dupond"
>>> client[256] = "Durand"
>>> client[782] = "Schmidt"
```

... إلخ

تمارين

2.3 أكتب سكريبت الذي يقوم بصنع نظام قاعدة بيانات صغير يعمل بمساعدة القاموس, هذا النظام يقوم بحفظ أسماء عدد من أصدقائك و أعمارهم و طولهم . السكريبت الخاص بك يجب عليه أن يحتوي على دالتان : الأولى لملء القاموس, و الثانية للإطلاع . في دالة الملء, إستخدم حلقة لقبول المعطيات من المستخدم .

في القاموس, إسم الطالب سيكون مفتاح الوصول, و القيم ستكون أنفاق (tuples) (العمر, الطول), حيث يتم التعمير على العمر بالسنوات (عدد صحيح), و الطول بالأمتار (عدد حقيقي) . دالة الإستشارة تشمل أيضا حلقة, حيث يستطيع المستخدم إدخال أي إسم ليتم إرجاع له زوجين (العمر, الطول) المقابل . و ستكون نتيجة الإستعلام سطر منسق جيدا, على سبيل المثال (الإسم: جين دوت - العمر: 15 سنة - الطول: 1.74 متر) . لتحقيق ذلك, إستخدم تنسيق السلاسل النصية التي شرحناها في الصفحة 224 .

3.3 أكتب دالة التي تبدل مفاتيح بقيم القاموس (تسمح لك على سبيل المثال بتحويل قاموس إنكليزي\فرنسي بقاموس فرنسي\إنكليزي) . نفترض أن القاموس لا يحتوي على قيم متطابقة عديدة .

إنشاء رسم بياني بالقاموس

القواميس هي أداة لإنشاء مدرج بياني أنيق .

على سبيل المثال, لنفترض أننا نريد إنشاء رسم بياني يمثل تكرار كل حرف من الأبجدية في نص المقدم . خورزمية التي تقوم بهذا العمل هي بسيطة للغاية إذا كنت تريد بنائها بالإعتماد على قاموس :

```
>>> texte ="les saucisses et saucissons secs sont dans le saloir"
>>> lettres ={}
>>> for c in texte:
...     lettres[c] =lettres.get(c, 0) + 1
...
>>> print(lettres)
{'t': 2, 'u': 2, 'r': 1, 's': 14, 'n': 3, 'o': 3, 'l': 3, 'i': 3, 'd': 1, 'e': 5, 'c':
3, ' ': 8, 'a': 4}
```

نبدأ من خلال إنشاء قاموس فارغ : **lettres** . ثم سنستخدم لملء هذا القاموس الأبجدية كمفتاح . القيم التي نريد تخزينها لك مفاتيح ستكون تكرار الأحرف في النص المقابل . لحساب هذا, سوف نقوم بتدوير سلسلة النصية **texte** . لكل حرف, سوف نطلب من القاموس بمساعدة الأسلوب **()get**, بإستخدام الحرف كمفتاح لقراءة التكرار الموجود لهذا الحرف . إذا كانت القيمة غير موجود الأسلوب **()get** سوف يعيد لنا قيمة فارغة . في جميع الحالات, سنقوم بزيادة القيمة الموجود, و نخزنها في القاموس, في المكان الذي يتوافق مع المفتاح (هذا معناه إلى الحرف الذي تتم معالجته) .

لتحسين عملنا, نستطيع أن نعرض الرسم البياني في ترتيب أبجدي . للقيام بذلك, سوف نفكر على الفور بإستخدام أسلوب الفرز **()sort**, و لكن هذا لا يمكن تطبيقه إلا على القوائم . هذا لا يهم ! لقد رأينا في الأعلى كيف يمكننا تحويل القاموس إلى قائمة أنفاق (tuples) :

```
>>> lettres_triees = list(lettres.items())
>>> lettres_triees.sort()
>>> print(lettres_triees)
[(' ', 8), ('a', 4), ('c', 3), ('d', 1), ('e', 5), ('i', 3), ('l', 3), ('n', 3), ('o',
3), ('r', 1), ('s', 14), ('t', 2), ('u', 2)]
```

تمارين

10.47 لديك تحت تصرفك أي ملف نصي (ليس كبير جدا) . أكتب سكريبت الذي يحسب تكرار كل حرف من الأبجدية في هذا النص (لتسهيل المشكلة تجاهل الحروف المعلمة) .

10.48 قم بتعديل السكريبت بالأعلى بحيث ينشئ جدول تواجد كل كلمة في النص . نصيحة : في أي نص, لا تفصل الكلمات فقط بالمسافات, لكن حتى بأدوات التنقيط المختلفة . لتبسيط المشكلة,

يمكنك البدء بإستبدال جميع الرموز بمسافات, تحويل السلسلة الناتجة في قائمة من الكلمات بمساعدة الأسلوب **split()** .

10.49 لديك تحت تصرفك أي ملف نصي (ليس كبيرا جدا) . أكتب سكريبت الذي يحلل هذا النص, و يقوم تخزين في قاموس المكان الدقيق لكل كلمة (قم بعد عد لبأحرف في البداية). إذا كانت كلمة موجودة في أماكن مختلفة, جميع الأماكن يجب تخزينها : كل قيمة في قاموس يجب أن تكون قائمة الأماكن .

التحكم في تدفق التنفيذ بإستخدام قاموس

كثيرا ما يحدث أننا نقوم بتنفيذ البرامج في إتجاهات مختلفة, إعتمادا على قيمة المتغير . يمكنك بالطبع التعامل مع هذه المشكلة بإستخدام سلسلة من العبارات if - elif -else , لكن هذه يمكن أن تصبح مرهقة جدا و غير أنيقة إذا كنت تتعامل مع عدد كبير من الإحتمالات, على سبيل المثال :

```
materiau = input("Choisissez le matériau : ")

if materiau == 'fer':
    fonctionA()
elif materiau == 'bois':
    fonctionC()
elif materiau == 'cuivre':
    fonctionB()
elif materiau == 'pierre':
    fonctionD()
elif   ... etc ...
```

الغات البرمجة توفر لك عبارات محددة للتعامل مع هذه المشكلة, مثل العبارات switch أو case في السي أو في باسكال . البايثون لا توفر لك أي واحد, لكن يمكنك الحصول عليها بمساعدة قائمة (لقد أعطينا شرح مفصل في الصفحة Error: Reference source not found),أو أفضل من ذلك عن طريق قاموس . على سبيل المثال :

```
materiau = input("Choisissez le matériau : ")

dico = {'fer':fonctionA,
        'bois':fonctionC,
        'cuivre':fonctionB,
        'pierre':fonctionD,
           ... etc ...}

dico[materiau]()
```

التعليمتان في الأعلى يمكن جمعها في تعليمة واحد فقط, لكننا جعلناها منفصلة لتفصيل الألية :

* العبارة الأولى تعرف قاموس **dico** حيث أن المفاتيح هي الإحتمالات المختلفة للمتغير **materiau**, و القيم هي أسماء الدالات التي يجب إستدعائها . لاحظ أنها سوى أسماء الدالات, و من المهم أن لا تضع القوسين في هذه الحالة (و إلا سوف يقوم البايثون بتنفيذ جميع الدالات في وقت صنع القاموس) .

العبارة الثانية تقوم بإستدعاء الدالة المقابلة للإختيار بمساعدة الدالة **materiau** . إسم الدالة سيتم إستخراجه من القاموس بمساعدة المفتاح, ثم يتم ربط بزوج من الأقواس . البايثون ستعرف إذا أنها إستدعاء دالة بشكل تقليدي, ثم يتم تشغيله .

يمكنك تعزيز التقنية في الأعلى بإستبدال هذه التعليمة مع مثيلها في الأسفل, الذي يقوم بإستدعاء الأسلوب **get()** في حالة إذا كان المفتاح المطلوب غير موجود في القاموس(و بهذه الطريقة يمكنك الحصول على ما يعادل تعليمة else في نهاية السلسلة الطويلة من elif) :

```
dico.get(materiau, fonctAutre)()
```

عندما تكون قيمة المتغير **materiau** لا تتطابق مع أي مفتاح في القاموس, يتم إستدعاء الدالة **()fonctAutre** .

تمارين

10.50 إكمل التمرين 10.46 (نظام قاعدة بيانات صغير) بإضافة دالتان : واحدة لحفظ القاموس الناتج في ملف نصي, و الثانية لإعادة بناء هذا القاموس من خلال هذا الملف النصي .

كل سطر من الملف النصي يتوافق مع عنصر من القاموس . سيتم التنسيق ذلك بطريقة مفصول جيدا :

- المفتاح و القيمة (هذا معناه إسم الشخص جزء, و مجموعة : العمر + الطول, في جزء أخر) .

- في مجموعة "العمر + الطول", هذان الإثنان معطيات رقمية . لذا يجب عليك إستخدام رمز للفضل, على سبيل المثال "**@**" للفصل بين المفتاح و القيمة, و "**#**" للفصل بين معطيات هذه القيمة :

```
Jean-Pierre@17#1.78
Delphine@19#1.71
Anne-Marie@17#1.63
```

10.51 حسّن سكريبت التمرين السابق, بإستخدام قاموس بتوجيه تدفق تنفيذ البرنامج في مستوى القائمة الرئيسية . برنامج سوف يعرض على سبيل المثال :

```
Choisissez :
(R)écupérer un dictionnaire préexistant sauvegardé dans un fichier
(A)jouter des données au dictionnaire courant
(C)onsulter le dictionnaire courant
(S)auvegarder le dictionnaire courant dans un fichier
(T)erminer :
```

إعتمادا على إختيار المستخدم, يتم إذا إستدعاء الدالة المقابلة عن طريق إختيار في قاموس الدالات .

11

الأصناف, الكائنات, الصفات

في الفصول السابقة, لقد إلتقيت بالفعل عدة مرات مع مفهوم الكائن . و أنت تعرف أن الكائن هو وحدة التي تم إنشاءها من مثيل من صنف (و هذا معناه هو نوع من "فئة" أو "نوع" كائن) . على سبيل المثال, يمكننا أن نجد في مكتبة Tkinter, صنف **Button()** من خلالها يمكننا صنع في النافذة أي عدد من الأزرار .

سوف ندرس الأن كيف يمكنك تعريف أصناف جديدة من الكائنات . هذا الموضوع صعب نسبيا, و لكننا سوف نقترب تدريجيا, بداية من تعريف أصناف كائنات بسيطة جدا, التي سوف نصقلها .

مثل كائنات الحياة اليومية(الحقيقية), الكائنات الحاسوبية يمكن أن تكون بسيطة جدا أو معقدة جدا . قد تكون متكونة من أجزاء مختلفة, و التي هي في حد ذاتها كائنات, و هذه جعلت بدورها كائنات أخرى أكثر بساطة, إلخ ...

فائدة الأصناف

الأصناف هي الأدوات الرئيسية للبرمجة الشيئية (Object Oriented Programming أو OOP) . هذا النوع من البرمجة يسمح لك بهيكلة البرامج المعقدة عن طريق تنظيمها على أنها مجموعات من كائنات التي تتفاعل مع بعضها و مع العالم الخارجي .

الفائدة الأولى من هذا النهج من البرمجة يكمن في بناء كائنات مختلفة تستخدم بشكل مستقل عن بعضها البعض (على سبيل المثال من قبل مبرمجين مختلفين) بدون خطر تداخلها . هذه النتيجة تحقق من خلال مفهوم التغليف : الوظيفة الداخلية للكائن و المتغيرات التي تستخدم للقيام بعملها,

نوعا ما بشكل مغلق في الكائن . لا يمكن للكائنات الأخرى أو العالم الخارجي الوصول إليها إلا من خلال إجراءات محددة جيدا : واجهة الكائن .

إن إستخدام الأصناف في برامجك يسمح لك - من بين الفوائد الأخرى - تجنب إسنخدام الحد الأقصى من المتغيرات العالمية . يجب أن تعرف أن إستخدام المتغيرات العالمية أمر خطير جدا, حتى أنه أكثر أهمية من البرامج كبيرة الحجم, لأنه من الممكن دائما تغيير هذه المتغيرات, أو حتى إعادة تعريفها, في أي جزء من البرنامج (هذا الخطر يزداد سوء إذا كان هنالك العديد من المبرمجين يعملون على نفس البرنامج) .

الفائدة الثانية الناتجة عن إستخدام الأصناف هي إمكانية بناء كائنات جديدة من الكائنات الموجودة, و بالتالي إعادة إستخدام المساحات البرمجية الكبيرة المكتوبة بالفعل (بدون لمسها !), لإشتقاق ميزة جديدة . هذا سيكون ممكنا بفضل مفاهيم الإشتقاق و تعدد الأشكال :

* الإشتقاق هي ألية تسمح لك ببناء صنف جديد "إبن أو طفل" من صنف "أم أو الأصل" . الطفل سيرث جميع خصائص و جميع وظائف الأم, و يمكنك إذا إضافة ما تريد عليها .

* يمكن لتعدد الأشكال تعيين سلوكيات مختلفة للكائنات المشتقة من بعضها البعض, أو من نفس الكائن أو من وفق لسياق معين .

قبل المضي قدما, لاحظ هنا أن البرمجة الشيئية هي شيئ إختياري في البايثون . يمكنك تطوير العديد من البرامج بدون إستخدامها, مع أدوات أكثر بساطو من دالات . إعلم أن إذا بذلت المزيد من الجهد في تعلم البرمجة بمساعدة الأصناف, سوف تتقن مستوى أعلى, مما يسمح لك التعامل مع مشاكل أكثر تعقيدا . و بعبارة أخرى, سوف تصبح مبرمج مختصا أكثر . لإقناعك بذلك, تذكر التقدم الذي قمت به خلال هذه الدورة :

* في بداية دراستك, بدأت بإستخدام تعليمات بسيطة . لقد كنت إلى حد ما "مبرمج باليد" (و هذا معناه تقريبا دون أدوات) .

* ثم تعرفت على الدالات التي تم تعريفها مسبقا (أنظر للفصل السادس), و تعلمت أن هناك مجموعات واسعة من الأدوات المتخصصة التي قدمت من قبل مبرمجين أخرين .

* تعلمت كتابة دالات خاصة بك (أنظر للفصل السابع و ما بعده), فأصبحت قادر على صنع أدوات جديدة خاصة بك, و هذه يمنحك تحكم كبير إضافي .

* إذا كنت ستبدأ الأن برمجة الأصناف, سوف تتعلم كيفية بناء ألات لإنتاج الأدوات . و هذا من الواضح أكثر تعقيدا من صنع الأدوات مباشرة, و لكن هذه تفتح لك أبواب أوسع من ذلك بكثير !

فهم هذه الأصناف يساعدك على السيطرة على مجال واجهات المستخدم الرسومي (tkinter, wxPython) و إعدادك للتصدي بشكل فعال على لغات حديثة أخرى مثل سي بلس بلس أو الجافا .

تعريف صنف أولي

لإنشاء صنف كائن بايثون, نستخدم العبارة class . سوف نتعلم إستخدام هذه العبارة, بدءا من تعريف نوع كائن أساسي, و الذي سيكون ببساطة مجرد نوع من البيانات الجديدة . و لقد إستخدمنا أنواع مختلفة من البيانات حتى الأن, و لكن كانت كل مرة من نوع مدمج في اللغة نفسها . سوف نقوم الأن يصنع نوع مركب جديد : النوع **Point** .

هذا النوع يتوافق مع مفهوم هندسة مستوى النقطة . في المستوى, يتم تمييز النقطة من رقمين (الإحداثيات x و y) . في التدوين الرياضي, يتم إذا تعريف النقطة من خلال إحداثيات x و y في زوج من الأقواس . على سبيل المثال النقطة (25, 17) . و هنالك طريقة طبيعية لتمثيل النقطة في البايثون ليتم إستخدامها لإحداثيات قيمتين من نوع حقيقي . لكن نحن نريد الجمع بين هاتين القيمتين في كيان واحد, أو في كائن واحد . لتحقيق ذلك سوف نقوم بتعريف الصنف Point() :

```
>>> class Point(object):
...     "Définition d'un point géométrique"
```

تعريف الأصناف يمكن أن يكون في أي مكان في البرنامج, لكن يتم وضعها بشكل عام في البداية (أو في وحدة يتم إستدعاءها) . المثال أعلاه هو على الأرجح الأبسط الذي يمكن تخيله . سطر واحد يكفي لتعريف نوع كائن جديد **Point()** .

نلاحظ ما يلي :* العبارة class هو مثال أخر على العبارة المجمعة . لا تنسى النقطتين في نهاية السطر فهي إلزامية, و مسافة البادئة في كتلة التعليمات التي تليها . هذا المجمع يجب ان يحتوي على الأقل على سطر واحد . في مثالنا المبسط للغاية, هذا السطر لا يحتوى سوى على تعليق بسيط .

كما رأينا سابقا في الدالات (أنظر الصفحة 71), يمكنك إضافة سلسلة نصية مباشرة بعد العبارة class, من أجل وضع تعليق لإدراجها تلقائيا إلى الوثائق الداخلية للبايثون . تعود دائما على وضع سلسلة لوصف الصنف هنا .

* تم وضع الأقواس ليحتوي على مرجع صنف موجود . هذا الأمر مطلوب للسماح لالية الميراث . كل الأصناف الجديدة التي نصنعهايمكن أن ترث من الصنف الأصل مجموعة من المميزات, التي تضاف إليها تلقائيا . عندما نريد إنشاء صنف أساسي (و هذا معناه غير معتمد على أي صنف أخر, مثل في مثالنا الصنف Point()) يجب أن يكون مرجع الإشارة حسب الإتفاقية هو الإسم الخاص object, مما يعني أنه جد جميع الأصناف الأخرى .

* الإتفاقية التي منتشرة بكثرة هي أن يتم إعطاء للأصناف أسماء تبدأ بحرف كبير . في بقية النص, سوف نحترم هذه الإتفاقية, و الاخر الذي يطلب في السرد, و يرتبط مه كل إسم صنف زوج من الأقواس, كما نفعل بالأسماء الدالات .

نحن نريد إذا تعريف الصنف **Point()** . يمكننا الأن صنع كائنات من هذا الصنف, و التي نسميها أيضا مثيل للصنف . و تسمى هذه العملية بالتمثيل . على سبيل االمثال ننشئ كائن جديد **p9** :[61] :

```
>>> p9 = Point()
```

بعد هذه العبارة المتغير **p9** يحتوي على مرجع للكائن الجديد **Point()** . يمككنا أن نقول أن **p9** هو مثيل جديد للصنف **Point()** .

> تنبيه
>
> مثل الدالات, يجب على الأصناف التي يتم إستدعائها في التعليمة أن تكون مصحوبة بقوسين (حتى لو لم يتم تمرير برامتر) . سوف نرى في الواقع أن الأصناف يمكن إستدعائهم مع برامترات .

دعونا نرى ما إذا كنا نستطيع أن نفعل شيئ مع الكائن الجديد **p9** :

61في البايثون, يمكننا إنشاء مثيل كائن بمساعدة تعليمة بسيطة خاصة . في اللغات الأخرى تتطلب إستخدام تعليمة خاصة, مثل **new** ليبين أن يقوم بإنشاء كائن جديد من العفن . على سبيل المثال : **p9 = new Point**() .

```
>>> print(p9)
<__main__.Point object at 0xb76f132c>
```

الرسالة التي تم إرجاعها بواسطة البايثون, سوف تفهم على الفور أن **p9** هو مثيل للصنف **Point()**, و التي تم تعريفها جيدا في المستوى الرئيسي (main) للبرنامج . و هب تم وضعها في موقع محدد في الذاكرة الحية (ram), و التي تظهر الأن بالترقيم السداسي العشري .

```
>>> print(p9.__doc__)
Définition d'un point géométrique
```

كما شرجنا للدالات (أنظر للصفحة 71), يتم ربط سلاسل توثيق كائنات البايثون المختلفة مع سمة المعرفة **__doc__** . و إذا فمن الممكن دائما العثور على الوثائق المرتبطة مع أي كائن بايثون, نت خلال إستدعاء هذه السمة .

سمات (أو متغيرات) المثيل

الكائن الذي صنعناه هو مجرد صدفة فارغة . سنضيف الأن عناصره, بتعيين بسيط, بإستخدام نظام تأهيل الأسماء بالنقاط[62] :

```
>>> p9.x = 3.0
>>> p9.y = 4.0
```

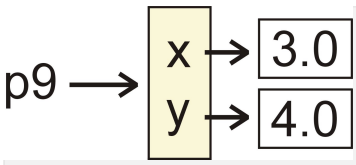
المتغيران x و y التي قمنا بتعريفهم من خلال الربط المباشر مع **p9**, هم الأن سمات للكائن **p9** . يمكن أيضا إستدعاء متغيرات المثيل . في الواقع يتم إدراجها, أو تغليفها في هذا المثيل (أو الكائن) . الرسم البياني على اليمين يظهر نتيجة هذه التعيينات : المتغير p9 يحتوي على مرجع يشير إلى موقع الكائن الجديد في الذاكرة, و الذي يحتوي على سمتين x و y . و هذه تتضمن مراجع للقيم 3.0 و 4.0 المخزنة في أماكن أخرى .

يمكننا إستخدام سمات كائن في أي تعبير, تماما مثل أي متغير عادي :

---

[62] هـذا نظـام الـترقيم مشـابه للـذي إسـتخدمناه لوصـف متغيـر لوحـدة, مثـل math.pi و string.ascii_lowercase . و سوف نعود في وقت لاحق, ولكن نعرف الأن أن الوحدات يمكنها أن تحتوي على دالات, و لكن حتى أصناف و متغيرات . حاول على سبيل المثال :

```
>>> import string
>>> string.capwords
>>> string.ascii_uppercase
>>> string.punctuation
>>> string.hexdigits
```




```
>>> print(p9.x)
3.0
>>> print(p9.x**2 + p9.y**2)
25.0
```

بسبب التغليف في الكائن, السمات هي متغيرات مستقلة عن المتغيرات الأخرى التي قد تحمل نفسالإسم . على سبيل المثال, التعليمة **x = p9.x** معناه : " إستخرج من مرجع الكائن في **p9** قيمة السمة **x** و قم بربط هذه القيمة بالمتغير **x** " .و لا يوجد تعارض بين المتغير المستقل **x**, و بين السمة **x** للكائن **p9** . الكائن **p9** يحتوي على مساحة الأسماء خاصة به, مستقلة عن مساحة أسماء الرئيسية التي تجد المتغير **x** .

> هام / الأمثلة الموجودة هنا هي مؤقتة ,
>
> لقد رأينا أنه من السهل جدا إضافة سمة إلى كائن بإستخدام تعليمة بسيطة مثل *p9.x = 3.0* و هذا يمكن تحمله في البايثون (و هذه نتيجة لطبيعة الحيوية للبايثون), و لكن لا ينصح بذلك حقا, كما سوف تفهم لاحقا , فنحن إذا لن نستخدم هذا النهج سوى للتوضيح, و فقط من أجل تبسيط الشرح لدينا لسمات المثيل . و ستم تطوير الطريقة الصحيحة في الفصل القادم .

تمرير كائن كبرامتر عند إستدعاء دالة

الدالات يمكنها إستخدام الكائنات كبرامترات, و يمكن أيضا إستخدام الكائن كقيمة للعودة . على سبيل المثال, يمكنك تعريف دالة مثل هذه :

```
>>> def affiche_point(p):
...     print("coord. horizontale =", p.x, "coord. verticale =", p.y)
```

البرامتر **p** الذي يستخدم من قبل هذه الدالة يجب أن يكون كائن من نوع **()Point**, الذي يستخدم متغيرات المثيل **p.x** و **p.y** . عندما تستدعى هذه الدالة, يجب عليك إذا توفير كائن من نوع **()Point** كبرامتر . جرب مع الكائن **p9** :

```
>>> affiche_point(p9)
coord. horizontale = 3.0 coord. Verticale = 4.0
```

تمرين

11.1 أكتب دالة **()distance** التي تسمح لك بحساب المسافة بين نقطتين . (يجب أن تتذكر نظرية فيثاغورس !)

هذه الدالة تنتظر كائنين من نوع **()Point** كبرامتر .

التشابه و التفرد

في اللغة التي نتحدث بها, نفس الكلمة يمكن أن يكون لها معان مختلفة إعتمادا على السياق الذي تستخدم فيه . و النتيجة يمكن أن نفهم بعض عبارات التي تستخدم فيها هذه الكلمات بمعان مختلفة (مصطلحات غامضة) .

على سبيل المثال كلمة "نفس - même" لديها معان كثيرة في الجمل : "شارلس و أنا لدينا نفس السيارة" و "شارلس و أنا لدينا نفس الأم" . في الجملة الأول, ما أعنيه أنني و شارلس لدينا نفس نموذج السيارة لكنهما سيارتين مختلفتين . و في الجملة الثانية, ما أعنيه أنني و شارس لدينا نفس الأم (نفس الشخص) .

عندما تتعامل مع كائنات البرامج, يمكن أن تجد نفس الغموض . على سبيل المثال, فإذا تحدثنا عن المساوات بين كائنين **()Point**, هذا معناه هذان الكائنات يحتويان على نفس المعطيات (سماتهم), أو يعني هذا أننا نتحدث عن مرجعين لنفس الكائن ؟ على سبيل المثال, التعليمات التالية :

```
>>> p1 = Point()
>>> p1.x = 3
>>> p1.y = 4
>>> p2 = Point()
>>> p2.x = 3
>>> p2.y = 4
>>> print(p1 == p2)
False
```

هذه التعليمات تقوم بإنشاء كائنين **p1** و **p2** الذين يبقون مستقلين, حتى لو كانو ينتمون إلى نفس الصنف و كانو لديهم نفس المحتويات . التعليمة الأخيرة تجرب مساوات هذان الكائنان (علامة مساوات مزدوجة), و النتيجة **False** (سالبة) : إذا لا يتساوون .

يمكننا تأكيد هذا بطريقة أخرى :

```
>>> print(p1)
<__main__.Point instance at 00C2CBEC>
>>> print(p2)
<__main__.Point instance at 00C50F9C>
```

معلومة واضحة : المتغيرين **p1** و **p2** مراجعهم مختلفة, تم تخزينهم في أماكن مختلفة في ذاكرة الحاسوب .

حاول شيئ أخر, الأن :

```
>>> p2 = p1
>>> print(p1 == p2)
True
```

بناء على التعليمة **p2 = p1**, قمنا بتعيين محتوى **p1** إلى **p2** . و هذا معناه متغيرين يشيران إلى مرجع الكائن نفسه . p1 و p2 هي أسماء مستعارة[63] لبعضها البعض .

إختبار المساواة هذه المرة أرجع لنا القيمة **True** (صحيح), و هذا معناه أن إختبار القوسين صحيح : **p1** و **p2** تم تعيينهم إلى كائن واحد, إذا لم تقتنع حاول :

```
>>> p1.x = 7
>>> print(p2.x)
7
```

عندما نغير سمة **x** ل **p1**, نلاحظ أن سمة **x** ل **p2** تتغير أيضا .

63بشأن ظاهرة الأسماء المستعارة, أنظر أيضا إلى صفحة: Error: Reference source not found.

```
>>> print(p1)
<__main__.Point instance at 00C2CBEC>
>>> print(p2)
<__main__.Point instance at 00C2CBEC>
```

المرجعين **p1** و **p2** يشيران إلى نفس المكان في الذاكرة .

كائنات تتكون من كائنات

الأن لنفترض أننا نريد تعريف دالة التي تمثل مستطيلات . ببساطة, نحن نعتبر أن هذه المستطيلات سوف تكون دائما موجهة أفقيا أو عموديا, أما قطريا فهذا مستحيل .

ما هي المعلومات التي نحتاجها لتعريف هذه المستطيلات ؟

هنالك العديد من الإحتمالات . يمكننا على سبيل المصال تحديد مكان وسط المستطيل (إحداثيتان) و تحديد حجمها (الطول و العرض) . و يمكننا أيضا تحديد أماكن الزوايا أعلى اليسار و إدنى اليمين . أو يمكننا تحديد مكان زاوية أعلى اليسار و الحجم . سوف نستخدم الطريقة الأخيرة .

عرف إذا صنف الخاص بنا الجديد :

```
>>> class Rectangle(object):
        "définition d'une classe de rectangles"
```

و سوف تخدمنا الأن لإنشاء مثيل :

```
>>> boite = Rectangle()
>>> boite.largeur = 50.0
>>> boite.hauteur = 35.0
```

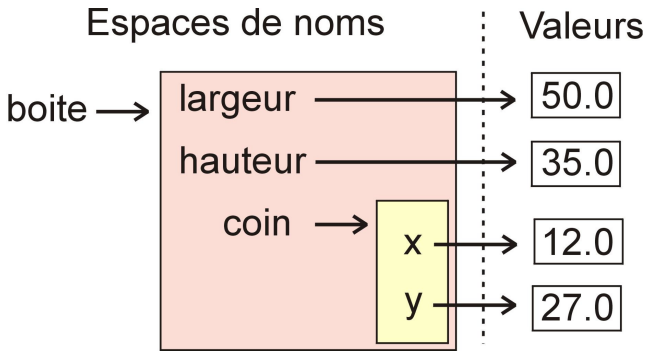
لقد صنعنا الكائن الجديد **Rectangle()** و أعطيناه سمتين . لتعريف الزاوية أعلى اليسار, سوف نستخدم مثيل جديد للصنف **Point()** الذي عرفناه سابقا . و بالتالي فإننا أنشأنا كائن جديد, داخل كائن أخر !

```
>>> boite.coin = Point()
>>> boite.coin.x = 12.0
>>> boite.coin.y = 27.0
```

في أول هذه التعليمات الثلاثة, قمنا بصنع سمة جديدة **coin** للكائن **boite** . ثم, للوصول لهذا الكائن و الذي هو في حد ذاته موجود في كائن أخر, إستخدمنا تصنيف الأسماء الهرمي (بمساعدة النقاط) و الذي تحدثنا عنه عدة مرات سابقا .

التعبير **boite.coin.y** يعني "إذهب إلى مرجع الكائن في المتغير **boite** . و في هذا الكائن, ثم جد السمة **coin**, ثم إذهب إلى مرجع كائن في هذه السمة . فإذا وجد الكائن, قم بتحديد سمته **y** " .

قد تفهم أفضل إذا كان كل هذا في رسم تخطيطي, مثل هذا :



الإسم **boite** يوجد في مساحة الأماكن الرئيسية . و هو يشير إلى مساحة أخرى للأسماء محجوز للكائن المقابل, في هذه المساحة يتم حفظ أسماء **lageur** و **hauteur** و **coin** . و هذه تشير إلى مساحات أخرى للأسماء (مثل الإسم "**cas**"), أو إلى قيم تم تعريفها جيدا, و التي يتم تخزينها في أماكن أخرى .

البايثون يقوم بحجز مساحات لأسماء مختلفة لكل وحدة, كل صنف, كل مثيل و كل كائن . يمكنك الإستفادة من كل هذه الأماكن المجزأة بشكل جيد لتحقيق برامج قوية, و هذا يعني برامج لا مكن أن تتداخل بسهولة .

الكائنات مثل القيم رجوع الدالة

لقد رأينا أعلاه أن الدالات يمكنها إستخدام الكائنات كبرامتران . يمكنها أن تمرر مثيل كقيمة رجوع . على سبيل المثال, الدالة **trouveCentre()** في الأسفل يمكن إستدعائها مع برامتر من نوع **()Rectangle** و هي ترسل كائن من نوع **Point()**, و التي تحتوي على إحداثيات وسط المستطيل .

```
>>> def trouveCentre(box):
...     p = Point()
...     p.x = box.coin.x + box.largeur/2.0
...     p.y = box.coin.y + box.hauteur/2.0
...     return p
```

على سبيل المثال, سمكنك إستدعاء هذه الدالة, بإستخدام كبرامتر الكائن **boite** الذي تم تعريفه بالأعلى :

```
>>> centre = trouveCentre(boite)
>>> print(centre.x, centre.y)
37.0  44.5
```

تعديل الكائنات

يمكننا تعديل خصائص كائن عن طريق تعيين قيم جديدة لسماته . على سبيل المثال, يمكننا تعديل حجم المستطيل (دون تغيير موقعه), عن طريق إعادة تعيين سماتيه **hauteur** و **largeur** :

```
>>> boite.hauteur = boite.hauteur + 20
>>> boite.largeur = boite.largeur - 5
```

يمكننا فعل هذا في البايثون, لأنه في هذه اللغة خصائص الكائن دائما عامة (على الأقل حتى الإصدار الحالي : 3.1) . في اللغات الأخر يوجد فرق واضح بين السمات العامة (يمكن الوصول إليها من خارج الكائن) و السمات الخاصة (الذي يمكن الوصول إليها فقط عن طريق الخوارزمية المدرجة في الكائن نفسه) .

و لكن كما ذكرنا في الأعلى (حول تعريف سمات تعيين بسيطة من خارج الكائن), طريقة تعديل سمات مثيل ليس موصى به, لأنه ينقص أحد أهداف الأساسة للبرمجة الشيئية, و الذي يهدف إلى إنشاء فصل صارم بين وظائف الكائن (حتى التي تم تعريفها في الخارج) و كيف يتم تنفيذها في الواقع هذه الوظيفة في الكائن (و العالم الخارج لا يعرف) .

عمليا, هذا يعني أننا يجب أن نتعلم كيفية عمل تشغيل الكائنات بإستخدام أدوات مناسبة, و التي نسميها الأساليب .

و الأن, بعد أن نفهم هذه العملية, يمكننا تحديد قاعدة بعدم تغيير السمات الكائن عن طريق العالم الخارجي مباشرة كما فعلنا الأن . و بدلا من ذلك نحن نريد إستخدام هذا الأسلوب لإعداده خصيصا لهذا الغرض, كما سوف نشرحه في الفصل القادم . جمع هذه الأساليب يكون لنا ما يسمى بواجهة الكائن .

12

الأصناف و الأساليب و الميراث

الأصناف التي عرفناها في الفصل السابق يمكن إعتبارهم كمساحات لأسماء معينة, و فيها نحن لن نضع حتى الأن سوى المتغيرات (سمات المثيل) . و يجب الأن إعطاء لهذه الأصناف وظيفة .

الفكرة الأساسية للبرمجة الشيئية هي في الواقع جمع الكائنات في نفس المجموعة معا, في كل مرة عدد من البيانات (سمات المثيل), و الخوارزميات تقوم بمختلف المعالجات على هذه البيانات (و هي الأساليب, أي دالات معينة مغلقة في كائن) .

الكائن = [سمات + أساليب]

و بهذه الطريقة يتم جمع في نفس "الكبسولة" خصائص الكائن و الدالات التي تعمل عليها, يتوافق مع مصممي البرامج رغبة في بناء كيانات حاسوبية ذات سلوك مشابه للكائنات في العالم الحقيقي الذي يحيط بنا .

على سبيل المثال الويدجت "زر" في تطبيق رسومي . فإنه يبدو من المعقول أن تتمنى أن الكائن الحاسوبي الذي ندعو سلوكه الذي يشبه أي زر جهاز في العالم الحقيقي . و نحن نعرف عمل الزر الحقيقي (قادر على غلق أو فتح الدارة الكهربائية) هي مدمجة في الكائن نفسه (و كذلك خصائصه الأخرى مثل حجمه و لونه و إلخ ...) .و بنفس الطريقة, نحن نأمل إذا إختلاف خصائص زر البرنامج (حجم, مكانه, لونه, النص المكتوب عليه), و لكن أيضا تعريف ما يحدث عندما القيام بحركات بالفأرة على هذا الزر, سواء أن يتم جمع داخل كيان محدد جدا في البرنامج, حتى لا يكون هنالك أي خلط بين الأزرار, أو حتى أكثر من ذلك بين زر و الكيانات الأخرى .

تعريف أسلوب

لتوضيح كلامنا, سوف نعرف صنف جديد **Time()**, و الذي ينبغي له أن يسمح لنا بأداء مجموعة من العمليات على اللحظات و المدد (جمع مدة), إلخ :

```
>>> class Time(object):
...     "définition d'objets temporels"
```

أصنع الأن كائن من نفس النوع, و أضف له متغيرات المثيل لحفظ الساعات و الدقائق و الثواني :

```
>>> instant = Time()
>>> instant.heure = 11
>>> instant.minute = 34
>>> instant.seconde = 25
```

بإعتباره تمرين, أكتب الأن بنفسك دالة **affiche_heure()**, و التي تستخدم لعرض محتويات كائن من صنف **Time()** في الشكل التقليدي "ساعات:دقائق:ثواني" . عند تطبيق الكائن الذي قمنا بصنعه فوق, هذه الدالة يجب أن تعرض 11:34:25 :

```
>>> affiche_heure(instant)
11:34:25
```

دالتك قد يكون شكلها كهذا :

```
>>> def affiche_heure(t):
...     print(str(t.heure) +":" +str(t.minute) +":" +str(t.seconde))
```

أو أفضل من ذلك, مثل هذا :

```
>>> def affiche_heure(t):
...     print("{0}:{1}:{2}".format(t.heure, t.minute, t.seconde))
```

بتطبيق تقنية تنسيق السلاسل التي شرحناها في الصفحة 138 .

ربما تكون بحاجة إلى إستخدام كائنات الصنف **Time()**, هذه الدالة ربما تكون مفيدة لك للعرض .

قد يكون من الحكمة تغليف هذه الدالة **affiche_heure()** في الصنف **Time()** نفسها, بطريقة نجعلها دائما متوفرة تلقائيا, كلما كنا نريد معالجة كائنات من الصنف **Time()** .

الدالة التي نريد تغليفها في صنف يسمى بشكل تفضيلي أسلوب méthode .

و لقد تحدثنا عن الأساليب في عدة مرات في الفصول السابقة من هذا الكتاب, و أنت تعرف بالعل أن الأساليب هي دالات مرتبطة بصنف محدد بكائنات . بقي فقط تعلم كيفية صنع هذه الدالة .

تعريف محدد لأسلوب في سكريبت

تعريف أسلةب مثل تعريف دالة, هذا معناه كتابة كتلة من التعليمات بعد الكلمة المحجوز def, لكن مع إختلافين :

* تعريف الأسلوب يكون مكانه دائما داخل تعريف الصنف, بطريقة حتى يظهر بوضوح العلاقة بين الأسلوب و الصنف .

* تعريف الأسلوب يجب أن يحتوي دائما على برامتر واحد على الأقل, و التي يجب أن تكون مرجع المثيل, و هذا البرامتر يجب أن يكون دائما الأول .

يمكنك مبدئيا إستخدام أي إسم متغير للبرامتر الأول, لكن يوصى بشدة متابعة الإتفاقية لذا يكون دائما إسمه : self .

البرامتر self ضروري, لأن يجب أن تكون قادر على تحديد المثيل الذي سيتم ربطه, في جزء تعليمات من تعريفه . سوف تفهم هذا بأكثر سهولة مع الأمثلة القادمة .

> نلاحظ أن تعريف الأسلوب يحتوي دائما على برامتر واحد : *self*, في حين أنه يمكن تعريف الدالة بدون أن تحتوي على أي شيئ .

أنظر كيف يحدث هذا عمليا :

للتأكد من أن الدالة **affiche_heure()** ستكون أسلوب للصنف **Time()**, سوف نحرك ببساطة تعريفها إلى داخل الصنف :

```
>>> class Time(object):
...     "Nouvelle classe temporelle"
...     def affiche_heure(t):
...         print("{0}:{1}:{2}".format(t.heure, t.minute, t.seconde))
```

من الناحية الفنية, فإن هذاا يكفي تماما, لأن البرامتر **t** تعين المثيل للسمات المرفقة **heure** و **minute** و **seconde** . نظرا لدورها, فمن المستحسن تغيير إسمها إلى self :

```
>>> class Time(object):
...     "Nouvelle classe temporelle"
...     def affiche_heure(self):
...         print("{0}:{1}:{2}".format(self.heure, self.minute, self.seconde))
```

تعريف الأسلوب **affiche_heure()** أصبح الأن جزء من كتلة التعليمات البادئة التالية للتعليمة الصنف (و هو أيضا جزء من الوثائق "Nouvelle classe temporelle") .

إختبار الأسلوب, في أي مثيل

لدينا الأن الصنف **Time()** مع الأسلوب **affiche_heure()** . من حيث المبدأ, نحن الأن قادرين على صنع كائنات من هذا الصنف, و تطبيق عليها هذا الأسلوب . دعونا نرى ما إذا كان يعمل . و للقيام بذلك, نبدأ من تمثيل كائن :

```
>>> maintenant = Time()
```

إذا حاولنا بسرعة إختبار الأسلوب الجديد على هذا الكائن, فإنه لا يعمل :

```
>>> maintenant.affiche_heure()
AttributeError: 'Time' object has no attribute 'heure'
```

هذا أمر طبيعي : لم نصنع بعد سمات المثيل . يجب أن نفوم على سبيل المثال :

```
>>> maintenant.heure = 13
>>> maintenant.minute = 34
>>> maintenant.seconde = 21
```

و نحاول مرة أخرى . و هذه المرة إشتغل :

```
>>> maintenant.affiche_heure()
13:34:21
```

في عدة مرات, لقد ذكرنا أنه ليس من المستحسن إنشاء سمات مثيل لربطها مباشرة من خارج الكائن نفسه . و من المضايقات الأخرى, فإن هذا يؤدي إلى أخطاء أخرى مثل التي ذكرناها سابقا . دعونا نرى الأن القيام بعمل أفضل .

الأسلوب المنشئ

الخطأ الذي تحدثنا عنه في الفقرة السابقة هل يمكن تجنبه ؟

هذا لا يحدث فعليا, فإذا قمنا بتنظيم الأسلوب **affiche_heure()** لجعله دائما يعرض شيئا ما, دون أي يكون من الضروري تعديل الكائن المنشئ حديثا . و بعبارات أخرى, سيكون من الحكمة أن متغيرات المثيل تكون معرفا مسبقا في البداية الصنف, مع قيمة إفتراضية .

لتحقيق هذا, سوف نقوم بإستدعاء أسلوب معين, و الذي سوف يعين فيما بعد تحت إسم المنشئ . الأسلوب المنشئ لديه طريقة فريدة لكي يتم تشغيله تلقائيا عند تمثيل كائن جديد بداية من الصنف . لذا يمكننا وضع كل ما يبدو ضروريا لتهيئته تلقائيا عند إنشاء كائن .

و هو معروف عند البايثون, الأسلوب المنشئ يدعى ضروري ب **__init__** (رمزي "في أسفل السطر", و كلمة **init**, ثم رمزي "في أسفل السطر) .

مثال :

```
>>> class Time(object):
...     "Encore une nouvelle classe temporelle"
...     def __init__(self):
...         self.heure =12
...         self.minute =0
...         self.seconde =0
...     def affiche_heure(self):
...         print("{0}:{1}:{2}".format(self.heure, self.minute, self.seconde))
```

كما كان سابقا, إنشاء كائن من هذا صنف و تجربة الأسلوب **affiche_heure()** :

```
>>> tstart = Time()
>>> tstart.affiche_heure()
12:0:0
```

لم نحصل على أي خطأ هذه المرة . في الواقع : عندما يتم إنشاء مثيل. الكائن **tstart** يتم تعيين 3 سمات **heure** و **minute** و **seconde** بواسطة الأسلوب المنشئ مع 12 و صفر كقيمة أولية . و لهذا كائن من هذا الصنف موجود, يمكن أن نطلب عرض هذه السمات فورا .

و الإستفادة من هذه التقنية يزداد وضوحا إذا أضفنا شيئا أخر .

مثل أي أسلوب يجب إحترامه, الأسلوب**() __init__** تكون برامتراته جاهزة . و في حالة هذا الأسلوب الخاص من المنشئ, البرامترا يمكن أن تلعب دور مثيرا للإهتمام, لأنها سوف تسمح تهيئة بعض متغيرات التمثيل في وقت تمثيل الكائن .

يرجى إذا الرجوع إلى التمرين السابق, و غير تعريف الأسلوب **()__init__** على النحو التالي :

```
...     def __init__(self, hh =12, mm =0, ss =0):
...         self.heure =hh
...         self.minute =mm
...         self.seconde =ss
```

أسلوبنا الجديد**() __init__** لديه الأن 3 برامترات, و لكل منها قيمة إفتراضية . و بالتالي نحصل على صنف أكثر تقدما . عندما كنا نمثل كائن من هذا الصنف, يمكننا الأن تهيئة سماته الرئيسية بمساعدة البرامترات, ضمن تعليمات التمثيل . و إذا أردنا حذف كل أو جزء منها, السمات تقوم بتسليم بأي شكل من الأشكال القيم الإفتراضية .

عندما نكتب تعليمات لتمثيل كائن جديد, و عندما تريد تمرير برامترات لأسلوب المنشئ, يكفي أن تضعهم بين قوسين التي تصاحب إسم الصنف . لنشرع إذا بالضبط مثل طريقة عندما نريد إستدعاء أي دالة .

هذا مثال لصنع و تمثيل كائن جديد **()Time** في نفس الوقت :

```
>>> recreation = Time(10, 15, 18)
>>> recreation.affiche_heure()
10:15:18
```

منذ أن المتغيرات التمثيل لها الأن قيم إفتراضية, نحن نستطيع أيضا صنع قيم إفتراضية للكائن **() Time** بحذف واحد أو أكثر من برامتر :

```
>>> rentree = Time(10, 30)
>>> rentree.affiche_heure()
10:30:0
```

أو أكثر :

```
>>> rendezVous = Time(hh =18)
>>> rendezVous.affiche_heure()
18:0:0
```

تمارين

12.1 عرف الصنف **()Domino** الذي يسمح بتمثيل كائنت لمحاكات أجزاء للعبة الدومينو . المنشئ في هذا الصنف يهيئ قيم هذه النقاط على جانبي **A** و **B** للدومينو (القيم الإفتراضية = 0) . و قم بتعريف أسلوبين أخرين :

* أسلوب **()affiche_points** الذي يعرض نقاط على الجانبين .

* أسلوب **valeur()** الذي يقوم بإرجاع مجموع النقاط الموجودة على الجانبين .

أمثلة إستخدام لهذا الصنف :

```
>>> d1 = Domino(2,6)
>>> d2 = Domino(4,3)
>>> d1.affiche_points()
face A : 2  face B : 6
>>> d2.affiche_points()
face A : 4  face B : 3
>>> print("total des points :", d1.valeur() + d2.valeur())
15
>>> liste_dominos = []
>>> for i in range(7):
...    liste_dominos.append(Domino(6, i))
>>> print(liste_dominos[3])
<__main__.Domino object at 0xb758b92c>
```

إلخ.

12.2 عرف الصنف **CompteBancaire()**, الذي يسمح بتمثيل كائنات مثل **compte1** و **compte2** إلخ . منشئ هذا الصنف يهيئ سمتي تمثيل **nom** و **solde**, مع قيم إفتراضية ' **Dupont**' و 1000 .

و ثلاثة الأساليب الأخرى التي يجب تعريفها :

- **depot(somme)** يسمح بإضافة كمية معينة على الدفع
- **retrait(somme)** يسمح بإزالة كمية معينة على الدفع
- **affiche()** يسمح بعرض إسم صاحب الحساب و رصيد حسابه .

أمثلة إستخدام لهذا الصنف :

```
>>> compte1 = CompteBancaire('Duchmol', 800)
>>> compte1.depot(350)
>>> compte1.retrait(200)
>>> compte1.affiche()
Le solde du compte bancaire de Duchmol est de 950 euros.
>>> compte2 = CompteBancaire()
>>> compte2.depot(25)
>>> compte2.affiche()
Le solde du compte bancaire de Dupont est de 1025 euros.
```

12.3 عرف الصنف **Voiture()** الذي يسمح بتمثيل كائنات إستنساخ سلوك السيارات . المنشئ لهذا الصنف يهيئ سمات التمثيل التالية, مع القيم الإفتراضية المبينة :

```
.marque = 'Ford',  couleur = 'rouge',  pilote = 'personne',  vitesse = 0
```

عند إنشاء مثيل لكائن جديد **Voiture()**, يمكننا إختيار شركته و لونه لكن ليس سرعته و ليس إسمه و ليس إسم سائقها .

سيتم تعريف هذه الأساليب :

- **choix_conducteur(nom)** يسمح بتحديد (أو تغيير) إسم السائق .
- **accelerer(taux, duree)** يسمح بتبديل سرعة السيارة . تبديل السرعة يساوى حسب سرعة المنتج : معدل × المدة . على سبيل المثال, إذا كان تسارع السيارة بمعدل 1,3 متر في الثانية لمدة 20 ثانية, سوف تحصل على سرعة 26 متر في الدقيقة . و يسمح بالمعدلات السلبية (و هو التباطئ) . الإختلاف في السرعة لم يسمح له إذا كان السائق "شخص" .
- **affiche_tout()** يسمح بإظهار خصائص الموجودة في السيارة, و هذا معناه شركتها, لونه و إسم سائقها و سرعتها .

أمثلة إستخدام لهذا الصنف :

```
>>> a1 = Voiture('Peugeot', 'bleue')
>>> a2 = Voiture(couleur = 'verte')
>>> a3 = Voiture('Mercedes')
>>> a1.choix_conducteur('Roméo')
>>> a2.choix_conducteur('Juliette')
>>> a2.accelerer(1.8, 12)
>>> a3.accelerer(1.9, 11)
Cette voiture n'a pas de conducteur !
>>> a2.affiche_tout()
Ford verte pilotée par Juliette, vitesse = 21.6 m/s.
>>> a3.affiche_tout()
Mercedes rouge pilotée par personne, vitesse = 0 m/s.
```

12.4 عرف الصنف **Satellite()** الذي يسمح بتمثيل كائنات تحاكي الأقمار الصناعية التي تطلق في الفضاء حول الأرض . المنشئ لهذا الصنف يهيئ سمات التمثيل التالية, مع القيم الإفتراضية المبينة :

.masse = 100, vitesse = 0

عند إنشاء مثيل لكائن جديد **Satellite()**, يمكن إختيار إسمه و كتلته و سرعته .

- الأساليب التالية سوف يتم تعريفها :
- **impulsion(force, duree)** تسمح بتفاوة سرعة القمر الصناعي . لمعرفة كيفية فعل هذا, تذكر دروس الفيزياء : سرعة التفاوة Δv التي يمر بها الكائن بكتلة m تخضع لحركة قوة F في غضون وقت t : $\Delta v = \frac{F \times t}{m}$. على سبيل المثال : قمر صناعي يزن 300 كيلوغرام يخضع لقوة 600 نيوتن لمدة 10 ثواني سوف تزداد سرعته(أو تنقص) 20م/ثانية .
- **affiche_vitesse()** عرض إسم القمر الصناعي و سرعته الحالية
- **energie()** تقوم بإرجاع إلى البرنامج الذي إستدعاها قيمة الطاقة الحركية للقمر الصناعي .
- تذكير / يتم حساب الطاقة الحركية بإستخدام الصيغة : $E_c = \frac{m \times v^2}{2}$

أمثلة إستخدام لهذا الصنف :

```
>>> s1 = Satellite('Zoé', masse =250, vitesse =10)
>>> s1.impulsion(500, 15)
>>> s1.affiche_vitesse()
vitesse du satellite Zoé = 40 m/s.
>>> print (s1.energie)
200000
>>> s1.impulsion(500, 15)
>>> s1.affiche_vitesse()
vitesse du satellite Zoé = 70 m/s.
>>> print (s1.energie)
612500
```

مساحات أسماء الأصناف و المثيل

لقد تعلمت سابقا (أنظر للصفحة 66) أن المتغيرات التي يتم تعريفها داخل دالة هي متغيرات خاص(محلي), لا يمكنها الوصول إلى تعليمات الموجودة خارج الدالة . و هذا يتيح لم إستخدام نفس الأسماء في أجزاء مختلفة من البرنامج, من دون خطر تداخلهم .

لوصف هذا بطريقة أخرى, يمكن أن نقول أن كل دالة لديها مساحة أسمائها, بغض النظر عن مساحة الإسماء الرئيسية .

و تعلمت أيضا أن التعليمات الموجودة داخل الدالة يمكنها الوصول إلى متغيرات التي تم تعريفها على مستوى الرئيسي, لكن فقط بالتشاور : يمكنها إستخدام قيم هذه المتغيرات, لكن ليس تعديلها (إلا لو إستدعيت عبارة global) .

لذا يوجد تسلسل هرمي بين مساحات الأسماء . سوف نرى نفس الشيئ عن الأصناف و الكائنات . و في الحقيقة :

* كل صنف لديه مساحة الأسماء الخاصة به . المتغيرات التي هي جزء من الصنف تسمى بمتغيرات الصنف أو سمات الصنف .

* كل كائن مثيل (تم صنعه من صنف) يحصل على مساحة الأسماء الخاصة به . المتغيرات التي هي جزء منها تسمى متغيرات المثيل أو سمات المثيل .

* الأصناف يمكنها إستخدام (لكن ليس تعديل) متغيرات التي تم تعريفها في المستوى الرئيسي .

* المثيل يمكنه إستخدام (لكن ليس تعديل) متغيرات التي تم تعريفها في المستوى الرئيسي للصنف و المتغيرات التي تم تعريفها في المتسوى الرئيسي للبرنامج .

على سبيل المثال أنظر إلى الصنف **Time()** الذي تم تعريفه سابقا . في الصفحة 177, مثلنا 3 كائنات من هذا الصنف : **recreation** و **rentree** و **rendezVous** . كل واحد تم تمثيله مع قيم مختلفة و مستقلة . يمكننا تعديل و إعادة عرض هذه المتغيرات كل واحدة من هذه الكائنات, دون أن يؤثر أي واحد على الأخر :

```
>>> recreation.heure = 12
>>> rentree.affiche_heure()
10:30:0
>>> recreation.affiche_heure()
12:15:18
```

أرجو الأن أن تقوم بترميز و تجربة المثال أدناه :

```
>>> class Espaces(object):                    # 1
...     aa = 33                               # 2
...     def affiche(self):                    # 3
...         print(aa, Espaces.aa, self.aa)    # 4
...
>>> aa = 12                                   # 5
>>> essai = Espaces()                         # 6
>>> essai.aa = 67                             # 7
>>> essai.affiche()                           # 8
12 33 67
>>> print(aa, Espaces.aa, essai.aa)           # 9
12 33 67
```

في هذا المثال, الإسم **aa** يستخدم لتعريف ثلاثة متغيرات مختلفة : الأولى في مساحة أسماء الصنف (في السطر الثاني), و واحدة أخرى في مساحة الأسماء الرئيسية (في السطر الخامس), و أخيرا في مساحة أسماء المثيل (في السطر السابع) .

في السطر الرابع و السطر التاسع يبين كيفية يمكنك الوصول إلى هذه الفضائات الثلاثة للأسماء (في داخل الصف, أو في المستوى الرئيسي), و ذلك بإستخدام تأهيل بالنقاط . و لاحظ أيضا مرة أخرى إستخدام self للإشارة إلى المثيل في داخل تعريف الصنف .

الإرث

الأصناف هي الأداة الرئيسية للبرمجة الشيئية (أو OOP), التي تعتبر اليوم تقنية البرمجة الأكثر فعالية . واحدة من المزايا الرئيسية لهذا النوع من البرمجة يكمن في أن نتمكن من إستخدام دائما فئة موجود لإنشاء واحدة أخرى, و التي سوف ترث جميع خصائصه لكن تستطيع تغيير بعضها أو إضافة خصائص جديدة . و هذه العملية تدعى الإشتقاق . و يقوم بإنشاء تسلسل هرمي للأصناف من العامة إلى الخاصة .

على سبيل المثال سوف نعرف الصنف **Mammifere()**, الذي يحتوي على خصائص هذه المجموعة من الحيوانات . و من هذا الصنف الأصل (الأم), يمكننا إشتقاق واحدة أو أكثر من هذه الأصناف الفرعية مثل صنف : **Primate()** و صنف **Rongeur()** و **Carnivore()** ...إلخ و التي سوف ترث جميع خصائص الصنف **Mammifere()** ثم نضيف إليها خصائص هذه المجموعة .

بداية من الصنف **Carnivore()**, نستطيع إشتقاق صنف **Belette()** و صنف **Loup()** و صنف **Chien()** ... إلخ . و هم يرثون جميع خصائص صنف الأصل و الصنف الذي فوقه . مثال :

```
>>> class Mammifere(object):
...     caract1 = "il allaite ses petits ;"

>>> class Carnivore(Mammifere):
...     caract2 = "il se nourrit de la chair de ses proies ;"

>>> class Chien(Carnivore):
...     caract3 = "son cri s'appelle aboiement ;"

>>> mirza = Chien()
>>> print(mirza.caract1, mirza.caract2, mirza.caract3)
il allaite ses petits ; il se nourrit de la chair de ses proies ;
son cri s'appelle aboiement ;
```

في هذا المثال, نرى أن الكائن **mirza**, لديه مثيل لصنف **Chien()**, و لم يرث فقط سمات التي تم تعريفها في هذا الصنف, بل حتى سمات التي تم تعريفها للأصناف الأصل .

لقد رأينا في هذا المثال كيف نشرع في إشتقاق صنف من الصنف الأصل (الأم) : بإستخدام تعليمة class, ثم كالعادة إسم الذي نريد تعيينه لهذا الصنف, و في ما لبن قوسين إسم الصنف الأصل . الأصناف الأول التي تستمد منها الأصناف الأخرى تسمى الأسلاف .

لاحظ أن السمات المستخدمة في هذا المثال هي سمات أصناف (و ليست سمات المثيل) . المثيل **mirza** يمكنه الوصول لهذه السمات, لكن ليس تغييرها :

```
>>> mirza.caract2 = "son corps est couvert de poils ;"   # 1
>>> print(mirza.caract2)                                 # 2
son corps est couvert de poils ;                         # 3
>>> fido = Chien()                                       # 4
>>> print(fido.caract2)                                  # 5
il se nourrit de la chair de ses proies ;                # 6
```

في المثال الجديد, في السطر الأول, لم يغير سمة **caract2** للصنف **Carnivore()**, على عكس ما تراه في السطر الثالث . يمكنك التحقق من خلال صنع مثيل جديد **fido** (من السطر 4 إلى السطر 6).

إذا كنت قد فهمت جيدا الفقرات السابقة, فإنك سوف تفهم أن التعليمة في السطر 1 تصنع متغير جديد لمثيل مرتبط فقط مع الكائن **mizra** . ولذلك يوجد الأن متغيرين لهما نفس الإسم **caract2** : واحدة في مساحة أسماء للكائن **mizra**, و الثانية في مساحة أسماء الصنف **Carnivore()**.

لكن كيف يمكننا أن نفسر ما يحدث في السطرين 2 و 3 ؟

كما رأينا أعلاه, المثيل **mizra** يمكنه الوصول للمتغيرات التي تقع في مساحة الأسماء الأصناف الأصل . فإذا وجد متغيرات بنفس الإسم في العديد من هذه المساحات, فأي واحدة سيختار أثناء تشغيل التعليمة مثل التي في السطر الثاني ؟

لحل هذا التعارض, يتبع البايثون قاعدة الأولويات في غاية البساطة . فعندما تسأله لإستخدام قيمة متغير يدعى **alpha**, على سبيل المثال, فإنه يبدأ في البحث عن هذا الإسم في المساحة المحلية (الداخلية) . فإذا وجد المتغير **alpha** في المساحة المحلية, يستخدمه و يوقف البحث . و إذا لم

يجده, يقوم البايثون بفحص مساحة الأسماء هيكل الأصل, ثم هيكل فوق الأصل, و إلخ إلى أن يصل إلى المستوى الرئيسي للبرنامج .

في السطر الثاني من مثالنا, فإنه متغير المثيل هو الذي يتم إستخدامه . و في السطر الخامس, فإنه في المستوى الصنف فوق الأصل يوجد به متغير بإسم **caract2** . لذا هذا الذي قام بعرضه .

الميراث و التعدد

حلل بعناية السكريبت في الصفحة التالية . أنها تفذ عدة مفاهيم مذكورة أعلاه, و لا سيما مفهوم الميراث .

لفهم السكريبت, يجب عليك تذكر بعض المفاهيم الكيميائية . في مادة الكيمياء الخاصة بك, فإنك بالتأكيد قد تعلمت أن الذرات هي كيانات تتكون من عدد من البروتونات(جسيمات مشحونة بطاقة كهربائية موجبة), و الإلكترونات (شحنتها سالبة) و النيوترونات (محايدة) .

نوع الذرة (أو العنصر) يتم تحديده من خلال عدد البروتونات, و الذي يسمى أيضا بالعدد الذري . في حالته الأساسية, الذرة تحتوي على إلكترونات بنفس عدد البروتونات, و بالتالي فهي متعادلة كهربائيا . كما أن لها عدد متغير من النيوترونات, لكن هذا لا يؤثر بأي شكل من الأشكال على الشحنة الكهربائية عموما .

في ظروف معينة, يمكن للذكرة أن تكتسب أو أن تفقد إلكترون . و في هذه الحالة يكتسب شخنة كهربائية عامة و يصبح أيون (الأيون السلبي إذا إكتسبت الذرة إلكترون أو أكثر و الأيون الإيجابي إذا فقدت) . الشحنة الكهربائية للأيون تساوي الفرق بين عدد البروتونات و عدد الإلكترونات التي تحتويها .

السكريبت في الصفحة التالية يصنع كائنات **Atome()** و كائنات **Ion()**. ذكرنا بالأعلى أن الأيون هو ببساطة ذرة تم تغييرها . في برمجتنا, الصنف الذي يقوم بتعريف كائنات **Ion()** سيكون صنف فرعي من الصنف **Atome()** : سوف يرث جميع سماته و أساليبه, ثم يضيف عليها خصائصه .

واحدة من هذه الأساليب التي تمت إضافتها (الأسلوب **affiche()**) تستبدل الأسلوب من نفس إسم الموروث من الصنف **Atome()**. الأصناف **Atome()** و **Ion()** لديهم كل واحد منهم أسلوب بنفس

الإسم, لكن يعمل بشك مختلف . لنتحدث عن هذه الحالة من تعدد الأشكال . يمكننا القول أن الأسلوب **affiche()** من الصنف **Atome()** كان فوق الطاقة .

من الواضح أنه من الممكن صنع مثيل لأي عدد من الذرات و الأيونات من هذين الصنفين . أو واحدة داخل الاخرى, الصنف **Atome()**, يجب عليه أن يحتوي على نسخة مبسطة من جدول الدوري للعناصر (الجدول الدوري), بحيث يمكنك تعيين إسم العنصر الكيميائي, و عدد النيوترونات, لكل كائن صنعته . كما أنه ليس من المرغوب نسخ هذا الجدول لكل مثيل, سوف نضعه في سمة الصنف . و بالتالي هذ الجدول لن يتواجد إلا في مكان واحد في الذاكرة, حتى تبقى في متناول جميع الكائنات التي سيتم إنتاجها من هذا الصنف .

```
class Atome:
    """atomes simplifiés, choisis parmi les 10 premiers éléments du TP"""
    table =[None, ('hydrogène',0),('hélium',2),('lithium',4),
            ('béryllium',5),('bore',6),('carbone',6),('azote',7),
            ('oxygène',8),('fluor',10),('néon',10)]

    def __init__(self, nat):
        "le n° atomique détermine le n. de protons, d'électrons et de neutrons"
        self.np, self.ne = nat, nat           # nat = العدد الذري
        self.nn = Atome.table[nat][1]

    def affiche(self):
        print()
        print("Nom de l'élément :", Atome.table[self.np][0])
        print("{0} protons, {1} électrons, {2} neutrons".\
              format(self.np, self.ne, self.nn))

class Ion(Atome):
    """les ions sont des atomes qui ont gagné ou perdu des électrons"""

    def __init__(self, nat, charge):
        "le n° atomique et la charge électrique déterminent l'ion"
        Atome.__init__(self, nat)
        self.ne = self.ne - charge
        self.charge = charge

    def affiche(self):
        Atome.affiche(self)
        print("Particule électrisée. Charge =", self.charge)

### : البرنامج الرئيسي ###

a1 = Atome(5)
a2 = Ion(3, 1)
a3 = Ion(8, -2)
a1.affiche()
a2.affiche()
a3.affiche()
```

عند تشغيل هذا البرنامج سوف يظهر التالي :

```
Nom de l'élément : bore
5 protons, 5 électrons, 6 neutrons

Nom de l'élément : lithium
3 protons, 2 électrons, 4 neutrons
Particule électrisée. Charge = 1

Nom de l'élément : oxygène
8 protons, 10 électrons, 8 neutrons
Particule électrisée. Charge = -2
```

في مستوى البرنامج الرئيسي, يمكنك أن ترى أننا مثلنا كائنات **Atome()** بتوفير عددها الذري (الذي يجب أن يكون ما بين 1 و 10) . لتمثيل كائنات **Ion()**, يجب أن نوفر العدد الذري و شحنته الكهربائية العامة (موجبة أو سالبة) . نفس الأسلوب **affiche()** يبين خصائص هذه الكائنات, سواء أن كانت ذرات أو أيونات, و في حالة الأيون خط إضافي (تعدد الأشكال) .

التعليقات

تعريف الصنف **Atome()** يبدأ بتعيين المتغير **table** . متغير يتم تعريفه هنا هو جزء من مساحة أسماء الصنف . إذا هذا هو سمة الصنف, الذي نضع فيها قائمة المعلومات حول 10 أول عناصر الجدول الدور لمندليف(إسم الشخص الذي إخترع هذا الجدول) .

لكل واحدة من هذه العناصر, قائمة تحتوي على نفق (tuple) : (إسم العنصر, عدد النيوترونات), و المؤشر الذي يتوافق مع العدد الذري . و بما أنه لا توجد عناصر عددها الذري صفر, لذا وضعنا للمؤشر صفر في القائمة, الكائن الخاص **None** . يمكننا وضع هناك أي قيمة أخرى, لأن هذا المؤشر لن يستخدم الكائن None للبايثون .

تليها تعارف الأسلوبين :

* المنشئ **__init__()** يستخدم أساسا هنا لصنع ثلاثة سمات المثيل, لتخزين في الذاكرة أعداد الروتونات و الإلكترونات و النيوترونات على التوالي لكل كائن ذرة صنع من هذا الصنف (تذكر أن سمات المثيل هي متغيرات مرتبطة برامتر self) .

لاحظ أن تقنية المستخدمة للحصول على عدد النيوترونات من سمة الصنف, مما يدل على إسم الصنف مؤهل بنقاط, مثل في التعليمة : **self.nn = Atome.table[nat][1]** .

* الأسلوب **affiche()** يستخدم سمات المثيل, لإيجاد عدد البروتونات و الألكترونات و النيوترونات للكائن الحالي, و سمة الصنف (التي هي مشتركة بين جميع الكائنات) لإستخراج إسم العنصر المطابق .

تعريف صنف **Ion()** يتضمن في أقواسه إسم الصنف **Atome()** أعلاه .

أساليب هذا الصنف هي بدائل الصنف **Atome()** . سيقومون بإحتمال إستدعاء هذه . هذه الملاحظة مهمة : كيف يمكننا, في إطار تعريف الصنف, إستدعاء الأسلوب الذي تم تعريفه في صنف أخر .

لا ينبغي أن نغفل, في الحقيقة, عن الأسلوب الذي يرتبط دائما بالمثيل الذي سيتم صنعه من هذا الصنف (المثيل يمثل بواسطة self في تعريفه) . إذا كان الأسلوب يجب عليه إستدعاء أسلوب أخر تم تعريفه في صنف أخر, يجب أن يكون قادر على نقل المرجع المثيل الذي ينبغي أن يقترن بها . كيف نفعل هذا ؟ هذا بسيط جدا :

> في تعريف الصنف, قد نرغب بإستدعاء أسلوب تم تعريفه في صنف اخر, يمكننا ببساطة إستدعائها مباشرة, عن طريق صنف اخر, بتمرير مرجع المثيل كبرامتر أول .

و بالتالي في سكريبتنا, على سبيل المثال, الأسلوب للصنف **affiche()** للصنف **Ion()** يمكنه إستدعاء الأسلوب **affiche()** للصنف **Atome()** : المعلومات المعروضة ستكون الكائن-الأيون الحالي, لأن مرجعه تم تمريره في تعليمة تدعى :

```
        Atome.affiche(self)
```

في هذه التعليمة, self بالطبع هو مرجع المثيل الحالي . بنفس الطريقة (سوف نرى أمثلة أخرى كثيرة فيما بعد), الأسلوب المنشئ للصنف **Ion()** إستدعى الأسلوب المنشئ للصنفه الأصل, في :

```
        Atome.__init__(self, nat)
```

هذا الإستدعاء ضروري, بحيث يتم تهيئة كائنات الصنف **()Ion** بنفس الطريقة كائنات الصنف **()Atome**. إذا كنا لم نقم بهذا الإستدعاء, الكائنات-الأيونات لن يرثوا تلقائيا سمات **ne** و **np** و **nn**, لأن هذه سمات المثيل تم صنعها بواسطة أسلوب المنشئ للصنف **()Atome**, و ذلك لن يتم إستدعائه تلقائيا عند إنشاء كائنات من صنف مشتق .

إفهم إذا أن الميراث لا ينطبق إلا على الأصناف, و ليس على مثيل هذه الأصناف . و عندما نقول أن صنف مشتق يرث جميع خصائص صنفه الأصل, هذا لا يعني أن الخصائص المثيل للصنف الأصل ينقل تلقائيا للأمثال للصنف الإبن . و وفقا لذلك, تذكر :

> في أسلوب منشئ الصنف المشتق, يجب تقريبا دائما أن تقوم بإستدعاء أسلوب المنشئ من صنف الأصل .

Résumé : définition et utilisation d'une classe

```
#####################################
# Programme Python type             #
# auteur : G.Swinnen, Liège, 2009   #
# licence : GPL                     #
#####################################

class Point(object):
    """point géométrique"""
    def __init__(self, x, y):
        self.x = x
        self.y = y

class Rectangle(object):
    """rectangle"""
    def __init__(self, ang, lar, hau):
        self.ang = ang
        self.lar = lar
        self.hau = hau

    def trouveCentre(self):
        xc = self.ang.x + self.lar /2
        yc = self.ang.y + self.hau /2
        return Point(xc, yc)

class Carre(Rectangle):
    """carré = rectangle particulier"""
    def __init__(self, coin, cote):
        Rectangle.__init__(self,
               coin, cote, cote)
        self.cote = cote

    def surface(self):
        return self.cote**2

###########################
## Programme principal : ##

# coord. de 2 coins sup. gauches :
csgR = Point(40,30)
csgC = Point(10,25)

# "boîtes" rectangulaire et carrée :
boiteR = Rectangle(csgR, 100, 50)
boiteC = Carre(csgC, 40)

# Coordonnées du centre pour chacune :
cR = boiteR.trouveCentre()
cC = boiteC.trouveCentre()

print("centre du rect. :", cR.x, cR.y)
print("centre du carré :", cC.x, cC.y)

print("surf. du carré :", end=' ')
print(boiteC.surface())
```

La classe est un moule servant à produire des objets. Chacun d'eux sera une instance de la classe considérée.

Les instances de la classe Point() seront des objets très simples qui possèderont seulement un attribut 'x' et un attribut 'y' ; ils ne seront dotés d'aucune méthode.

Le paramètre SELF désigne toutes les instances qui seront produites à partir de cette classe.

Les instances de la classe Rectangle() posséderont trois attributs. Le premier ('ang') doit être lui-même un objet de classe Point(). Il servira à mémoriser les coordonnées de l'angle supérieur gauche du rectangle. Les deux autres contiendront sa largeur et sa hauteur.

La classe Rectangle() comporte une méthode, qui renverra un objet de classe Point() au programme appelant.

Carre() est une classe dérivée, qui hérite les attributs et méthodes de la classe Rectangle().
Son constructeur doit faire appel au constructeur de la classe parente, en lui transmettant la référence de l'instance en cours de création (self) comme premier argument.

La classe Carre() comporte une méthode de plus que sa classe parente.

Pour créer (ou instancier) un objet, il suffit d'appeler une classe comme on appelle une fonction. La valeur renvoyée est une nouvelle instance de cette classe. Les instructions ci-contre créent donc deux objets de la classe Point() ...

... et celles-ci, encore deux autres objets.

Note : par convention, on donne aux classes des noms commençant par une majuscule.

La méthode trouveCentre() fonctionne pour les objets des deux types, puisque la classe Carre() a hérité de la classe Rectangle().

La méthode surface(), par contre, ne peut être invoquée que pour les objets Carre().

وحدات تحتوي على مكتبات الأصناف

أنت تعرف بالفعل منذ وقت طويل إستخدام وحدات البايثون (أنظر للصفحات 50 و 71) . و أنت تعرف أنهم يتم تجميعهم في مكتبات متكونة من الأصناف و الدالات . كتمرين مرجعة, سوف تقوم بصنع وحدة جديدة من الأصناف, بترميز أسطر التعليمات في الأسفل في ملف وحدة التي سوف تسميها **formes.py** :

```
class Rectangle(object):
    "Classe de rectangles"
    def __init__(self, longueur =0, largeur =0):
        self.L = longueur
        self.l = largeur
        self.nom ="rectangle"

    def perimetre(self):
        return "({0:d} + {1:d}) * 2 = {2:d}".\
                format(self.L, self.l, (self.L + self.l)*2)
    def surface(self):
        return "{0:d} * {1:d} = {2:d}".format(self.L, self.l, self.L*self.l)

    def mesures(self):
        print("Un {0} de {1:d} sur {2:d}".format(self.nom, self.L, self.l))
        print("a une surface de {0}".format(self.surface()))
        print("et un périmètre de {0}\n".format(self.perimetre()))

class Carre(Rectangle):
    "Classe de carrés"
    def __init__(self, cote):
        Rectangle.__init__(self, cote, cote)
        self.nom ="carré"

if __name__ == "__main__":
    r1 = Rectangle(15, 30)
    r1.mesures()
    c1 = Carre(13)
    c1.mesures()
```

عند حفظ هذه الوحدة, يمكنك إستخدامه بطريقتين : إما أن تقوم بتشغيله مثل أي برنامج, و إما من خلال إستدعائه في أي سكريبت أو من سطر الأوامر, لإستخدام أصنافه . أنظر لهذا المثال :

```
>>> import formes
>>> f1 = formes.Rectangle(27, 12)
>>> f1.mesures()
Un rectangle de 27 sur 12
a une surface de 27 * 12 = 324
et un périmètre de (27 + 12) * 2 = 78

>>> f2 = formes.Carre(13)
>>> f2.mesures()
Un carré de 13 sur 13
```

```
a une surface de 13 * 13 = 169
et un périmètre de (13 + 13) * 2 = 52
```

نحن نرى في هذا السكريبت أن الصنف **Carre()** تم إنشائه من الصنف **Rectangle()** لذا هو يرث جميع خصائصه . و بعبارة أخرى, الصنف **Carre()** هو إبن الصنف **Rectangle()**.

قد تلاحظ مرة أخرى أن منشئ الصنف **Carre()** يجب عليه إستدعاء منشئ الصنف الأصل (**Rectangle.__init__(self**, ...), و أن يمرر له مرجع المثيل (self) كأول برامتر .

أما بالنسبة للتعليمة :

```
if __name__ == "__main__":
```

تم وضعها في نهاية الوحدة, و تستخدم لمعرفة إذا كانت الوحدة تم تشغيلها كبرنامج مستقل (في هذه الحالة يجب أن يتم تنفيذ التعليمات التي تليها), أو تم إستخدامه كمكتبة لصنف تم إستدعائه في مكان أخرى . في هذه الحالة هذا الجزء من الكود ليس له أي تأثير .

تمارين

12.5 عرف الصنف **Cercle()**. و الأصناف التي تم صنعا من هذا الصنف تكون دوائر بأحجام مختلفة . بالإضافة إلى أسلوب منشئ (الذي سوف يستخدم البرامتر **rayon**), يمكنك تعريف الأسلوب **surface()**, الذي يقوم بإرجاع مساحة الدائرة .

قم بتعريف الصنف **Cylindre()** المشتق من الذي قبله . المنشئ لهذا الصنف الجديد يتضمن برامترين **rayon** و **hauteur** . أضف الأسلوب **volume()** الذي سيقوم بإرجاع حجم الأسطوانة (تذكير : حجم الإسطوانة = مساحة المقطع × الإرتفاع) .

مثال إستخدام لهذا الصنف :

```
>>> cyl = Cylindre(5, 7)
>>> print(cyl.surface())
78.54
>>> print(cyl.volume())
549.78
```

12.7 أكمل التمرين السابق بإضافة صنف جديد **Cone()**, الذي سوف يشتق هذه المرة من الصنف **Cylindre()**, و المنشئ يتضمن أضا برامترين **rayon** و **hauteur** . هذا الصنف الجديد لديها

أسلوبها **volume()**, و التي ستقوم بإرجاع حجم المخروط (تذكر حجم المخروط = حجم الإسطوانة المقابلة مقسومة على 3) .

مثال لإستخدام لهذا الصنف :

```
>>> co = Cone(5,7)
>>> print(co.volume())
183.26
```

12.7 عرف الصنف **JeuDeCartes()** الذي يسمح بتمثيل الكائنات التي سلوكها مشابه لسلوك لعبة ورق حقيقية . الصنف يجب عليه أن يحتوي على الأقل على أربعة الأساليب التالية :

* أسلوب المنشئ : إنشاء و ملئ قائمة ثم 52 عنصر, و التي كل واحد منها متكونة من نفق (tuple) من عددين صحيحين . هذه قائمة الأنفاق (tuple) تحتوي على خصائص لكل واحدة من 52 بطاقة . لكل واحدة منها, يجب تخزين بشكل منفصل عدد صحيح يشير إلى قيمة (2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, الأربعة الأخيرة قيم جاك و الملكة و الملك و الأس), و العدد الصحيح الأخر يشير إلى لون الورقة (و هذا معناه 0,1,2,3 للقلب و الماس و و النوادي و بستوني) .

في هذه القائمة, العنصر (11,2) معناه جاك النوادي, و القائمة يجب أن تكتمل بنوع :

[(2, 0), (3,0), (3,0), (4,0), ... (12,3), (13,3), (14,3)]

* الأسلوب **nom_carte()** : هذا الأسلوب يجب عليه أن تقوم بإرجاع في شكل سلسلة, بها هوية البطاقة و يجب أن تكون نفق (tuple) لوصف البرامتر . على سبيل لامثال, التعليمة : **print(jeu.nom_carte((14, 3))** يجب أن تعرض : As de pique .

* الأسلوب **méthode battre()** : كما يعلم الجميع, معناه خلط الأوراق . هذا الأسلوب سوف يخلط قائمة العناصر التي تحتوي على الأروراق, بغض النظر عن العدد .

* الأسلوب **tirer()** : عندما يتم إستدعاء هذا الأسلوب, يتم سحب ورقة . النفق (tuple) الذي يحتوي على القيمة و اللون يتم إرساله للبرنامج الذي إستدعاه . يتم دائما سحب(إزالة من القائمة) أول ورقة في القائمة . فإذا تم إستدعاء هذا الأسلوب و لم يعد يوجد أوراق في القائمة, يجب إرسال الكائن الخاص **None** إلى البرنامج الذي إستدعاه .

أمثلة إستخدام للصنف **JeuDeCartes()** :

```
jeu = JeuDeCartes()                # تمثيل كائن
jeu.battre()                       # خلط الأوراق
for n in range(53):                # صنع 52 ورقة
   c = jeu.tirer()
   if c == None:                   # لا يوجد أي ورقة في القائمة
      print('Terminé !')
   else:
      print(jeu.nom_carte(c))      # قيمة و لون الورقة
```

12.8 أكمل التمرين السابق : عرف لاعبين **A** و **B** . قم بتمثيل ورقتي لعب (واحد لكل لاعب) و قم بخلطها . ثم بمساعدة حلقة التكرار, إسحب 52 مرة ورقة لكل واحد من هذان اللاعبان و قارن قيمتهما . إذا كانت الأول الأكبر قيمة يتم إضافة نقطة للاعب **A** . فإذا حدث العكس فيتم إضافة نقطة للاعب **B** . فإذا كانت القيمتين متعادلتين, يتم مرور إلى السحب التالي . عند إنتهاء الحلقة, أحسب نقاط **A** و **B** لمعرفة الفائز .

12.9 أكتب سكريبت جديد الذي يقوم بإسترداد كون التمرين 12.2 (الحساب المصرفي) عن طريق إستدعاءه كوحدة . و عرف صنف **CompteEpargne()**, مشتقة من صنف **()** **CompteBancaire** التي تم إستدعائها, و التي تسمح بصنع حسابات مدخرات الذي يضاف إليه بعض الفائدة بعد مرور الوقت . للتبسيط, نحن نفترض أنه يتم حساب فائدة شهرية .

منشئ صنفك الجديد يجب عليه أن يهيئ فائدة شهرية بالتقصير تساوي 0.3% . و أسلوب **changeTaux(valeur)** يسمح بتغيير معدل الفائدة .

الأسلوب **capitalisation(nombreMois)** يجب أن يكون :

* يعرض عدد الأشهر و فائدة التي يتم أخذها في الحساب .

حساب المال المتحصل عليه من خلال الفائدة و الأشهر التي تم إختيارها .

أمثلة إستخدام هذا الصنف :

```
>>> c1 = CompteEpargne('Duvivier', 600)
>>> c1.depot(350)
>>> c1.affiche()
Le solde du compte bancaire de Duvivier est de 950 euros.
>>> c1.capitalisation(12)
Capitalisation sur 12 mois au taux mensuel de 0.3 %.
>>> c1.affiche()
Le solde du compte bancaire de Duvivier est de 984.769981274 euros.
>>> c1.changeTaux(.5)
>>> c1.capitalisation(12)
Capitalisation sur 12 mois au taux mensuel de 0.5 %.
>>> c1.affiche()
Le solde du compte bancaire de Duvivier est de 1045.50843891 euros.
```

13

الأصناف و واجهات المستخدم الرسومية

البرمجة الشيئية هي مناسبة خاصة لتطوير التطبيقات مع واجهات المستخدم الرسومية . مكتبات الأصناف مثل tkinter أو wxPython توفر أساس ودجات واسعا جدا, التي نستطيع أن نكيف إحتياجاتنا من الإشتقاق . في هذا الفصل, سوف نستخدم مرة أخرى مكتبة tkinter, لكن سوف نطبق المفاهيم الموضحة في الصفحات السابقة, و سنسعى لتسليط الضوء على مزايا البرمجة الشيئية في برامجنا .

كود الألوان : مشروع صغير مغلف بشكل جيد

سنبدأ مع مشروع صغير مستوحى من الدورات في مجال الألكترونيات . التطبيق الذي سنصفه أدناه يمكنه العثور بسرعة على رمز 3 ألوان المطابقة للمقاوم الكهربائي لقيمة محددة جدا .

للتذكير, إن وظيفة المقاوم هي مقاومة (إعتراض) قليل أو كثير من تدفق التيار الكهربائي . المقاوميظهر بشكل ملموس في شكل أنبوبي من أجزاء صغيرة بها ثلاثة خطوط من الألوان (عادة 3) . هذه الخطوط تشير إلى القيمة العددية للمقاومة, إعتمادا على التالي :

كل لون يتوافق مع رقم (الأرقام تبدأ من 0 إلى 9) :

أسود = 0؛ بني = 1؛ أحمر = 2؛ برتقالي = 3؛ أصفر = 4؛ أخضر = 5؛ أزرق = 6؛ بنفسجي = 7؛ رمادي = 8؛ أبيض = 9

المقاومة موجهة بحيث يتم وضع الخطوط (الشرائح) الملونة على اليسار . قيمة المقاومة - تعرب بالأوم (Ω) - يتم قراءة الخطوط (الأشرطة) من اليسار : أول خطين يشيران إلى أول رقمين من القيمة الرقمية . ثم يجب إلحاق هذان الرقمين بعدد من الأصفار مساوي للخط الثالث .

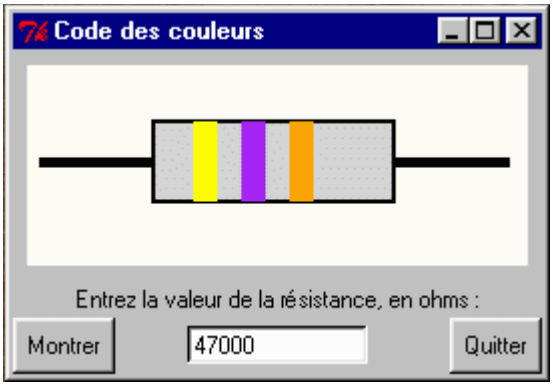
مثال : الألوان من اليسار هي : أصفر و بنفسجي و أخضر .

قيمة المقاوم هي : 4700000 أوم أو 4700 كيلوأوم أو 4,7 ميغاأوم .

هذا النظام لا يسمح بتوضيح القيمة الرقمية إلا مع رقمين فقط . و مع ذلك هذا منتشر على نطاق واسع . و هو كافيا بالنسبة لمعظم التطبيقات "العادية" (الراديو, التلفاز, إلخ) .

مواصفات برنامجنا

يجب على برنامجنا أن يقوم بعرض نافذة تحتوي على رسم للمقاوم, و يجب على المستخدم إدخال قيمة العددية للمقاوم ثم الضغط على <Montrer> و هذا سيتسبب في تغيير رسم المقاوم, بطريقة لتتوافق ألوان الخطوط مع القيمة التي أدخلها المستخدم .



عائق : يجب على البرنامج قبول أي قيمة رقمية (عدد صحيح أو حقيقي) و ذلك في حدود من 10 إلى 10^11 أوم . على سبيل المثال, القيمة 4.78e6 يجب أن تكون مقبولة و يجب تحويلها إلى 4800000 Ω .

تطبيق ملموس

سوف نبني جسم هذا التطبيق البسيط على شكل صنف . نحن نريد أن نظهر لك كيف أن الصنف يمكنه خدمة مساحة الأسماء المشتركة, حيث يمكنك تغليف المتغيراتنا و دالاتنا . و الميزة الرئيسية هو أن يسمح لم بتمرير متغيرات عامة . في الواقع :

* بدء تشغيل التطبيق لتلخيص مثيل كائن هذا الصنف .

* الدالات التي نريد تنفيذعا ستكون ساليب لهذا كائن-التطبيق .

* و ضمن هذه الأساليب, يمكنك ببساطة إرفاق إسم المتغير للبرامتر self لكي يمكنك الوصول لهذا المتغير من أي مكان من داخل الكائن . المتغير المثيل يشبه تماما متغير عمومي (لكن فقط داخل الكائن), لأن جميع الأساليب للكائن يمكنهم الوصول إلى self .

```
1#    class Application(object):
2#        def __init__(self):
3#            """Constructeur de la fenêtre principale"""
4#            self.root =Tk()
5#            self.root.title('Code des couleurs')
6#            self.dessineResistance()
7#            Label(self.root, text ="Entrez la valeur de la résistance, en ohms :").\
8#                    grid(row =2, column =1, columnspan =3)
9#            Button(self.root, text ='Montrer', command =self.changeCouleurs).\
10#                   grid(row =3, column =1)
11#           Button(self.root, text ='Quitter', command =self.root.quit).\
12#                   grid(row =3, column =3)
13#           self.entree = Entry(self.root, width =14)
14#           self.entree.grid(row =3, column =2)
15#           #9 إلى 0 الألوان للقيمة من :
16#           self.cc = ['black','brown','red','orange','yellow',
17#                      'green','blue','purple','grey','white']
18#           self.root.mainloop()
19#
20#       def dessineResistance(self):
21#           """Canevas avec un modèle de résistance à trois lignes colorées"""
22#           self.can = Canvas(self.root, width=250, height =100, bg ='ivory')
23#           self.can.grid(row =1, column =1, columnspan =3, pady =5, padx =5)
24#           self.can.create_line(10, 50, 240, 50, width =5)          # fils
25#           self.can.create_rectangle(65, 30, 185, 70, fill ='light grey', width =2)
26#           # رسم ثلاثة خطوط ملونة (أسود في البداية) :
27#           self.ligne =[]              # تخزين الخطوط الثلاثة في قائمة واحدة
28#               self.ligne.append(self.can.create_rectangle(x,30,x+12,70,
29#                                                           fill='black',width=0))
30#
31#       def changeCouleurs(self):
32#           """Affichage des couleurs correspondant à la valeur entrée"""
33#           self.v1ch = self.entree.get()        # هذا الأسلوب يقوم بإرجاع سلسلة واحدة
34#           try:
35#               v = float(self.v1ch)             # تحويلها إلى قيمة رقيمة
36#           except:
37#               err =1                           # خطأ : المعطيات ليست رقمية
38#           else:
39#               err =0
40#           if err ==1 or v < 10 or v > 1e11 :
41#               self.signaleErreur()             # المدخلات خاطئة أو خارجة عن النطاق
42#           else:
43#               li =[0]*3                        # قائمة من 3 رموز لعرض
44#               logv = int(log10(v))             # لعرض الجزء الصحيح من الخوارزمية
45#               ordgr = 10**logv                 # ترتيب الحجم
46#               # إستخراج أول أرقام هامة :
47#               li[0] = int(v/ordgr)             # جزء صحيح
48#               decim = v/ordgr - li[0]          # جزء حقيقي
49#               # إستخراج ثاني رقم هام :
50#               li[1] = int(decim*10 +.5)        # +.5 لتقريب بشكل صحيح
51#               # عدد الأصفار المرتبطة بالرقمين الكبيرين
52#               li[2] = logv -1
```

```
53#                 # تلوين الخطوط الثلاثة :
54#                 for n in range(3):
55#                     self.can.itemconfigure(self.ligne[n], fill =self.cc[li[n]])
56#
57#         def signaleErreur(self):
58#             self.entree.configure(bg ='red')            # تبوين خلفية الحقل
59#             self.root.after(1000, self.videEntree)      # قم بمسحه بعد 1 ثانية
60#
61#         def videEntree(self):
62#             self.entree.configure(bg ='white')          # إستعد الخلفية البيضاء
63#             self.entree.delete(0, len(self.v1ch))    # إزالة الحروف المكتوبة
64#
65#    # البرنامج الرئيسي :
66#    if __name__ == '__main__':
67#        from tkinter import *
68#        from math import log10                  # خوارزمية بأساس 10
69#        f = Application()                       # تمثيل كائن التطبيق
```

تعليقات

* السطر الأول : الصنف تم تعريفه من صنف مستقل (أي أنه لا يستمد من أي صنف أصل, لكن فقط الكائن, و هو سلف جميع الأصناف الأخرى) .

* الأسطر من 2 إلى 14 : منشئ صنف المثيل للويدجات المطلوبة : مساحة رسوم, ملصق (Label) و أزرار . و لتحسين إمكانية قراءة البرنامج, وضعنا مثيل للوحة (مع رسم للمقاوم) في أسلوب منفصل : **dessineResistance()**. يرجى ملاحظة أنه للحصول على كود أصغر حجما, نحن لا نقوم بحفظ الأزرار و الملصقات (Label) في متغيرات (كما شرحنا سابقا في الصفحة 98), و ذلك لأننا لا نريد صنع مرجع لها في أماكن مختلفة من البرنامج . نحن إستحدمنا لأماكن الويدجات في النافذة الأسلوب grid() الذي وصفناه في الصفحة 95 .

* الأسطر من 15 إلى 17 : يتم تخزين كود الألوان في قائمة بسيطة .

* السطر 18 : التعليمة الأخيرة لمنشئ بداية البرنامج . فإذا كنت تفضل بدء البرنامج بشكل مستقل عن صنعه, يجب عليك في هذه الحالة حذف هذا السطر, و نستدعي **mainloop()** في المستوى الرئيسي للبرنامج, بإضافة التعليمة **f.root.mainloop()** في السطر 71 .

* الأسطر 20 إلى 30 : رسم المقاوم يتكون من خط و أول شريحة رمادية فاتحة, لجسم المقاوم و إبنيه . و ثلاثة مستطيلات أخرى كشرائح ملونة و تتغير ألوانه إعتمادا على مدخلات المستخدم . هذه الشرائح تكون لونها أسود في البداية و مرجعهم في قائمة **self.ligne** .

* الأسطر من 32 إلى 53 : هذه الأسطر تحتوي على أساس وظائف البرنامج . مدخلات المستخدم يتم قبولها في شكل سلاسل نصية . في السطر 36, فإننا نحاول تحويل هذه السلسلة إلى قيمة رقمية من نوع حقيقي . فإذا فشل التحويل, يتم تخزين الخطأ . فإذا تحول إلى قيمة رقمية, نتأكد من أنه داخل النطاق المسموح به (من 10 Ω إلى 11^10 Ω). فإذا تم الكشف عن خطأ, فإننا ننبه المستخدم على أن مدخلاته لها خطأ عن طريق تلوين حقل الإدخال بلون أحمر, و يتم إفراغ محتوياته (الأسطر من 55 إلى 61) .

* الأسطر 45 و 46 : إن الرياضيات أتت لنجدتنا لإستخراج القيمة الرقمية من ترتيبها الكبير (هذا معناه, من أقرب 10 أس (قوة)) . يرجي الرجوع إلى كتاب رياضيات لمزيد من التوضيح عن اللوغاريتمات .

* Lignes 47-48 : Une fois connu l’ordre de grandeur, il devient relativement facile d’extraire du nombre traité ses deux premiers chiffres significatifs. Exemple : supposons que la valeur entrée soit 31687. Le logarithme de ce nombre est 4,50088... dont la partie entière (4) nous donne l’ordre de grandeur de la valeur entrée (soit 104). Pour extraire de celle-ci son premier chiffre significatif, il suffit de la diviser par 104, soit 10000, et de conserver seulement la partie entière du résultat (3).

* الأسطر من 49 إلى 51 : نتيجة عملية القسمة التي قمنا بها في الفقرة السابقة هي 3,1687 . Nous récupérons la partie décimale de ce nombre à la ligne 49, soit 0,1687 dans notre exemple . فإذا ضربنا في عشرة, ce nouveau résultat comporte une partie entière qui n’est rien d’autre que notre second chiffre significatif (1 dans notre exemple).

Nous pourrions facilement extraire ce dernier chiffre, mais puisque c’est le dernier, nous souhaitons encore qu’il soit correctement arrondi. Pour ce faire, il suffit d’ajouter une demi unité au produit de la multiplication par dix, avant d’en extraire la valeur entière. Dans notre exemple, en

effet, ce calcul donnera donc 1,687 + 0,5 = 2,187 , dont la partie entière (2) est bien la valeur arrondie recherchée.

* السطر 53 : عدد الأصفار التي ترتبط برقمين مهمين الموافقة لحساب ترتيب الحجم . يمكنك ببساطة إزالة وحدة في الخوارزمية .

* السطر 56 : لتعيين لون جديد لكائن يمكن رسمه على اللوحة, نستخدم الأسلوب **()** **itemconfigure** . سوف نستخدم إذا هذا الأسلوب لتبديل خيار التلوين لكل شريحة ملونة, بإستخدام الألوان التي تم إستخراجها من **self.cc** بمؤشرات الثلاثة **li[1]** و **li[2]** و **li[3]** التي تحتوي على ثلاثة أرقام .

تمارين

13.1 قم بتعديل السكريبت بالأعلى بحيث يصبح صورة الخلفية أزرق فاتح (light blue), و يصبح جسم المقاومة لونه بيج (يشبه اللون البني - beige), و السلك المقاومة يصبح أنحف, و شرائح الألوان التي تدل على القيمة تصبح أكبر .

13.2 عدل السكريبت في الأعلى بحيث تصبح صورة المرسومة أكبر مرتان .

13.3 عد السكريبت أعلاه بحيث يصبح من الممكن إدخال قيمة المقاوم التي ما بين 1 و 10 Ω . و لهذه القيم الشريحة الأولى الملونة يجب أن تبقى سوداء, و الشريحتان المتبقيتان يشيرات القيم ب Ω و الثانية ب Ω .

13.4 عدل السكريبت أعلاه بحيث أن الزر <Montrer> يصبح غير إلزامي . في سكريبتك المعدل, يكفي أن تضغط على <Enter> بعد إدخال قيمة المقاوم, لتنشط الشاشة .

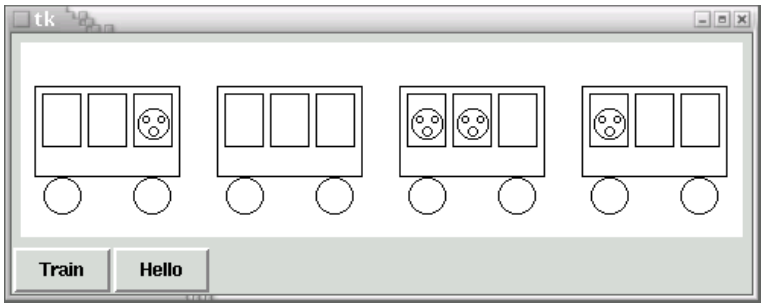
13.5 عدل السكريبت في الأعلى بحيث الشرائح الثلاثة الملونة ترجع سوداء في حالة أدخل المستخدم ملاخلات غير مقبولة .

صغير: الميراث, تبادل المعلومات بين الكائنات

في التمرين السابق, نحن لم نستغل سوى ميزة واحدة من الأصناف و هي التغليف . و هذا يسمح لنا بكتابة برامج بمختلف دالاتها (و التي أصبحت أساليب) يمكن كل واحد منها الوصول إلى نفس المجموعة من المتغيرات : جميع التي تم تعريفها كمرفق لـ self . و يمكن إعتبار كل هذه المتغيرات كنوع من المتغيرات العامة داخل الكائن .

يجب عليك أن تفهم أنه ليست متغيرات عامة حقيقية . بل هي مقتصر فقط على داخل الكائن, و ينصح بالوصول إليها من الخارج[64]. من ناحية أخرى, فإن جميع الكائنات التي مثلتها من نفس الصنف تمتلك كل واحدة خصائص مجموعة هذه المتغيرات . و التي هي تغليف لهذه الكائنات . و هذا ما يطلق عليه بسمات المثيل .

سوف ننتقل الأن إلى السرعة القصوى, و سننشئ تطبيق صغير على أساس عدة أصناف, لدراسة كيفية تبادل المعلومات بين عدة أصناف من خلال أساليبهم . و سوف نستخدم أيضا هذا التمرين لنبين لك كيف يمكنك تعريف الصنف الأصل لبرنامجك الغرافيكيك بإشتقاق من صنف tkinter, و بالتالي الإستفادة من الية الميراث .



[64] كما ذكرنا سابقا, يسمح لك البايثون بالوصول إلى سمات المثيل بإستخدام تأهيل الأسماء بالنقاط . اللغات البرمجة الأخرى تحظر, أو تسمح لك سوى من خلال إعلان خاص لهذه السمات (سمات التمييز بين الخاص و العام) .
يرجى ملاحظة أنه في أي حال فمن الغير المستحسن : إن إستخدام السليم للبرمجة الشيئية تنص على أن تكون قادرا على الوصول إلى سمات الكائنات من خلال طرق محددة (واجهة) .

المشروع الذي سنطوره هنا بسيط جدا, لكن لكنه يمكن أن يكون الخطوة الأولى في إنشاء الألعاب : كما سنوفر الأمثلة أدناه (أنظر للصفحة Error: Reference source not found). و مشروعنا عبارة عن نافذة تحتوي على لوحة و زران . عندما ينشط الزر الأول, يظهر قطار صغير على اللوحة . و عندما ننشط الزر الثاني, بعض الأشخاص الصغار يظهرون في بعض نوافذ العربات .

المواصفات

البرنامج سوف يتضمن صنفين :

* الصنف **Application()** ستم الحصول عليه من خلال إشتقاق أصناف أساسية من tkinter و سوف يقوم بعرض النافذة الرئيسية و اللوحة و الزران .

* الصنف المستقل **Wagon()**, يسمح بتمثيل في اللوحة 4 كائنات-عربات, و لكل واحدة منها أسلوب **perso()** . و هذا سيتسبب في ظهور الأشخاص الصغار في أي واحدة من النوافذ الثلاثة للعربة . البرنامج الرئيسي يستدعي هذا الأسلوب بشكل مختلف لمختلف الكائنات-العربات,ثم ليظهر بعض الأشخاص .

التطبيق

```
1#     from tkinter import *
2#
3#     def cercle(can, x, y, r):
4#        "dessin d'un cercle de rayon <r> en <x,y> dans le canevas <can>"
5#        can.create_oval(x-r, y-r, x+r, y+r)
6#
7#     class Application(Tk):
8#        def __init__(self):
9#           Tk.__init__(self)      # منشئ الصنف الأصل
10#          self.can =Canvas(self, width =475, height =130, bg ="white")
11#          self.can.pack(side =TOP, padx =5, pady =5)
12#          Button(self, text ="Train", command =self.dessine).pack(side =LEFT)
13#          Button(self, text ="Hello", command =self.coucou).pack(side =LEFT)
14#
15#       def dessine(self):
16#          "instanciation de 4 wagons dans le canevas"
17#          self.w1 = Wagon(self.can, 10, 30)
18#          self.w2 = Wagon(self.can, 130, 30)
19#          self.w3 = Wagon(self.can, 250, 30)
20#          self.w4 = Wagon(self.can, 370, 30)
21#
22#       def coucou(self):
23#          "apparition de personnages dans certaines fenêtres"
```

```
24#         self.w1.perso(3)      #العربة الأولى, النافذة الثالثة
25#         self.w3.perso(1)      #العربة الثالثة, النافذة الأولى
26#         self.w3.perso(2)      #العربة الثالثة, النافذة الثانية
27#         self.w4.perso(1)      #العربة الرابعة, النافذة الأولى
28#
29#   class Wagon(object):
30#      def __init__(self, canev, x, y):
31#         "dessin d'un petit wagon en <x,y> dans le canevas <canev>"
32#         #: تخزين برامترات في متغيرات المثيل
33#         self.canev, self.x, self.y = canev, x, y
34#         #: المستطيل الأساسي : 95×60 بيكسل
35#         canev.create_rectangle(x, y, x+95, y+60)
36#         #3: نوافذ من 34×40 بيكسل, و خمسة بيكسلات فاصلة
37#         for xf in range(x+5, x+90, 30):
38#            canev.create_rectangle(xf, y+5, xf+25, y+40)
39#         #: عجلتين نصف قطرها يساوي 12 بكسل
40#         cercle(canev, x+18, y+73, 12)
41#         cercle(canev, x+77, y+73, 12)
42#
43#      def perso(self, fen):
44#         "apparition d'un petit personnage à la fenêtre <fen>"
45#         #: حساب إحداثيات مركز كل نافذة
46#         xf = self.x + fen*30 -12
47#         yf = self.y + 25
48#         cercle(self.canev, xf, yf, 10)     #الوجه
49#         cercle(self.canev, xf-5, yf-3, 2)  #العين اليسرى
50#         cercle(self.canev, xf+5, yf-3, 2)  #العين اليمنى
51#         cercle(self.canev, xf, yf+5, 3)    #الفم
52#
53#   app = Application()
54#   app.mainloop()
```

* الأسطر من 3 إلى 5 : نحن نخطط لرسم سلسلة من الدوائر الصغيرة . و هذه الدالة الصغيرة تسهل عملنا من خلال تعريف الدوائر من خلال وسطها و نصف قطرها .

* الأسطر من 7 إلى 13 : تم صنع الصنف الرئيسي لبرنامجنا بالإشتقاق عن صنف النوافذ **Tk()** التي تم إستدعائها من وحدة tkinter[65] كما شرحنا في الفصل السابق, منشئ الصنف المشتق يقوم بتنشيط بنفسه منشئ الصنف الأصل, و يمرر له مرجع المثيل كأول برامتر . الأسطر من 10 إلى 13 تستخدم لوضع اللوحة و الأزرار في أماكنها .

* الأسطر من 15 إلى 20 : هذه الأسطر تقوم بتمثيل 4 كائنات-عربات, المنتجة من الصنف المقابل . و يمكننا البرمجة بأكثر أناقة بمساعد حلقة و قائمة, لكننا لا نريد أن نتعب أنفسنا بالتفسيرات الغير

65سوف نرى لاحقا أن tkinter يسمح أيضا ببناء نافذة رئيسية لتطبيق بإشتقاق من صنف لويدجت (في معظم الأحيان ستكون بالإشتقاق من ويدجت Frame()) . النافذة التي تشمل هذا الويدجت سيتم إضافتها تلقائيا (أنظر إلى صفحة :) .

الضرورية . نحن نريد وضع كائنات-عرباتنا في اللوحة, في مواقع محددة : لذا يجب علينا أن نمرر بعض المعلومات لمنشئ لهذه الكائنات : على الأقل مرجع اللوحة, و الإحداثيات المطلوبة . و هذه الإعتبارات تكون تلمبح عندما نعرف صنف **Wagon()** , نبتعد قليلا, يجب علينا أن نربط مع أسلوب منشئها عدد مساوي من البرامترات من أجل الحصول على هذه البرامترات .

* الأسطر من 22 إلى 27 : بتم إستدعاء هذا الأسلوب عندما ننشط الزر الثاني . و هي تستدعي بنفسها الأسلوب **perso()** لبعض أجسام- العربات, مع برامترات مختلفة, لإظهار الأشخاص في نوافذ معينة . هذه السطور القليلة من الكود تظهر لك كيفية كائن يمكن التواصل مع كائنات أخرى, و ذلك عن طريق أساليبه . و بعد هذه الالية المركزية للبرمجة بواسطة الكائنات :

> الكائنات هي كيانات برمجية التي تتفاعل من خلال تبادل الرسائل و أساليبها .

من الناحية المثالية, يجب على الأسلوب **coucou()** أن يشمل بعض التعليمات الإضافية, التي من شأنها أن تحقق أولا ما إذا كانت الكائنات-العربات موجودة بالفعل أو لا, قبل أن تسمح بتفعيل أساليبها . نحن لن نضع هذا النوع من الحماية للحفاظ على أكبر قدر ممكن من البساطة, لكن هذه نتيجة التسلسل التي لا تستطيع بها تفعيل الزر الثاني قبل الأول .

* الأسطر 29 و 30 : لا يشتق الصنف **Wagon()** من أي صنف أخر, لأنه صنف من الكائنات الرسومية, و مع ذلك يجب علينا أن نجلب البرامترات من المنشئ من أجل الحصول على مرجع اللوحة سنضع بها الرسوم, بالإضافة إلى إحداثيات هذه الرسوم . في التجاربك يمكنك من هذا التمرين, إضافة المزيد من البرامترات مثل : حجم الرسم, التوجيه, اللون, السرعة ...إلخ

* الأسطر من 31 إلى 51 : هذه الأسطر تتطلب القليل من التعاليق . الأسلوب **perso()** لديه برامتر الذي يشير إلى 3 النوافذ التي يجب أن يظهر بها الأشخاص الصغار . هنا أيضا ليس لدينا حماية : يمكنك إستدعاء هذا الأسلوب مع برامتر يساوي 4 أو 5, على سبيل المثال, و هذا يحدث أشياء خاطئة .

* الأسطر 53 و 54 : لهذا التطبيق, على عكس سابقا, فضلنا الفصل بصنع الكائن **app**, و يبدأ من خلال إستدعاء **mainloop()**, في هذين التعلمتين المنفصلتين (كمثال) . يمكنك أيضا تقليل هتين

التعليمتين في تعليمة واحدة, و التي ستكون : **Application().mainloop()**, و بالتالي وفرنا متغير .

تمارين

13.6 إجعل السكريبت الذي وصفناه في الأعلى مثاليا, بإضافة برامتر اللون إلى منشئ الصنف **Wagon()**, التي سوف تحدد لون مقصورة العربة . إجعل النوافذ سوداء في البداية, والعجلات رمادية (لتحقيق هذا الهدف, أضف أيضا برامتر اللون إلى دالة **cercle()**).

في صنف **Wagon()**, أضف أيضا الأسلوب **allumer()**, و الذي سيستخدم لتغيير لون لثلاثة نوافذ (مبدئيا سوداء) إلى الأصفر, لمحاكاة الإضاءة الداخلية .

أضف زر إلى النافذة الرئيسية, التي تشغل الأضواء . حسن **الدالة cercle()** لتلوين وجوه الأشخاص الصغار إلى اللون الوردي (pink), أعينهم و أفواههم إلى اللون الأسود, ومثل الكائنات-العربات مع ألوان مختلفة .

13.7 أضف إلى البرنامج السابق, بحيث يمكننا إستخدام أي زر في الإضطراب, على الرغم من أن هذا سيؤدي إلى خطأ أو تأثير غريب .

مخطط الذبذبات : ويدجت مخصص

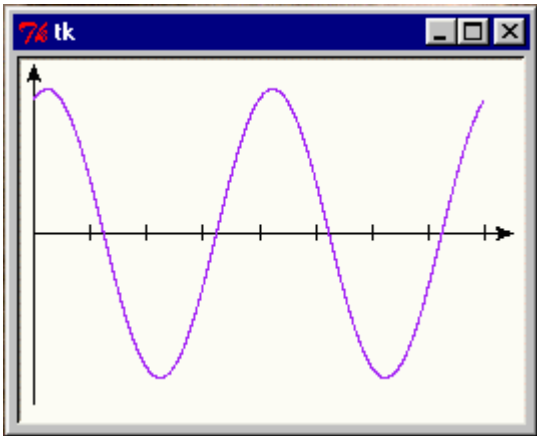
المشروع الذي سنقوم به الأن سيقودنا خطوة إلى الأمام . سنبني صنف ويدجت جديد, و الذي سيكون من الممكن إضافته إلى مشاريعنا القادمة مثل أي ويدجت عادي . مثل الصنف الأساسي في التمرين السابق, هذا الصن الجديد سيتم بناءه بالإشتقاق لصنف tkinter الموجود مسبقا .

هذا الموضوع لهذا التطبيق سوف نستلهم بعض منه من درس الفيزياء .

Pour rappel, un mouvement vibratoire harmonique se définit comme étant la projection d’un mouvement circulaire uniforme sur une droite. Les positions successives d’un mobile qui effectue ce type de mouvement sont traditionnellement repérées par rapport à une position

centrale : on les appelle alors des élongations. L'équation qui décrit l'évolution de l'élongation d'utel mobile au cours du temps est toujours de la forme $e = A \sin(2 \pi f t + \varphi)$, dans laquelle e représente l'élongation du mobile à tout instant t . Les constantes A, f et φ désignent respectivement l'amplitude, la fréquence et la phase du mouvement vibratoire.

الهدف من هذا المشروع توفير أداة بسيطة لعرض هذه المفاهيم المختلفة, و هي نظام لعرض تلقائيا رسومات إطالة/الوقت . يستطيع المستخدم أن يختار بحرية قيم البرامترات **A** و **f** و **φ** و يجب عليه أن يراعي المنحنيات الناتجة عن ذلك .



يرجى ترميز السكريبت بالأسفل و حفظه في ملف بإسم **oscillo.py** . أنت تعرف أن الوحدة الحقيقية تحتوى على صنف (يمكنك إضافة أصناف أخرى في وقت لاحق في نفس الوحدة) .

```
1#    from tkinter import *
2#    from math import sin, pi
3#
4#    class OscilloGraphe(Canvas):
5#      "Canevas spécialisé, pour dessiner des courbes élongation/temps"
6#      def __init__(self, boss =None, larg=200, haut=150):
7#        "Constructeur du graphique : axes et échelle horiz."
8#        #منشئ ويدجت الأصل
9#        Canvas.__init__(self)                    #إستدعاء منشئ
10#       self.configure(width=larg, height=haut)      #الصنف الأصل
11#       self.larg, self.haut = larg, haut              #حفظ
12#       #مسار محاور المرجع
13#       self.create_line(10, haut/2, larg, haut/2, arrow=LAST)  #محور X
14#       self.create_line(10, haut-5, 10, 5, arrow=LAST)       #محور Y
15#       #: رسم سلم مع 8 درجات
```

```
16#         pas = (larg-25)/8.        # فترات سلم أفقي
17#         for t in range(1, 9):
18#            stx = 10 + t*pas       # +10 للبدأ من المنشئ
19#            self.create_line(stx, haut/2-4, stx, haut/2+4)
20#
21#      def traceCourbe(self, freq=1, phase=0, ampl=10, coul='red'):
22#         "tracé d'un graphique élongation/temps sur 1 seconde"
23#         curve =[]                 # قائمة الإحداثيات
24#         pas = (self.larg-25)/1000.    # الموافق ل 1 ثانية X السلم
25#         for t in range(0,1001,5):     # التي تنقسم إلى 100 ميلي ثانية
26#            e = ampl*sin(2*pi*freq*t/1000 - phase)
27#            x = 10 + t*pas
28#            y = self.haut/2 - e*self.haut/25
29#            curve.append((x,y))
30#         n = self.create_line(curve, fill=coul, smooth=1)
31#         return n                  # n = الرقم التسلسلي للمسار
32#
33#    ####: كود لتجربة الصنف ####
34#
35#   if __name__ == '__main__':
36#      root = Tk()
37#      gra = OscilloGraphe(root, 250, 180)
38#      gra.pack()
39#      gra.configure(bg ='ivory', bd =2, relief=SUNKEN)
40#      gra.traceCourbe(2, 1.2, 10, 'purple')
41#      root.mainloop()
```

المستوى الرئيسي للسكريبت يتكون من السطر 35 إلى السطر 41 .

كما سبق أن أوضحنا في الصفحة 237, الأسطر الكود التي بعد التعليمة **if __name__ == '__main__':** لن يتم تنفذها إذا كان السكريبت تم إستدعائه كوحدة في تطبيق أخر . فإذا شغلت هذا السكريبت كبرنامج رئيسي, سيتم تنفيذ هذه التعليمات . و بالتالي لدينا الية مثيرة للإهتمام, فهي تسمح لنا بدمج التعليمات الإختبار في إطار وحدة, حتى لو كانت معدة ليتم إستيرادها إلى سكريبتات أخرى .

قم إذا بتشغيل السكريبت بالطريقة العادية . يجب عليك أن تحصل على عرض يشبه الذي قمنا به في الصفحات السابقة .

تجربة

سوف نقوم بالتعليق على الأسطر المهمة للسكريبت . لكن لنبدأ أولا من خلال تجربة قليلا الصنف الذي صنعناه للتو .

إفتح إذا طرفيتك, و أدخل التعليمات بالأسفل مباشرة إلى سطر الأوامر :

```
>>> from oscillo import *
>>> g1 = OscilloGraphe()
>>> g1.pack()
```

بعد إستدعاء صنف الوحدة oscillo, قمنا بتمثيل كائن أول **g1**, للصنف **OscilloGraphe()** .

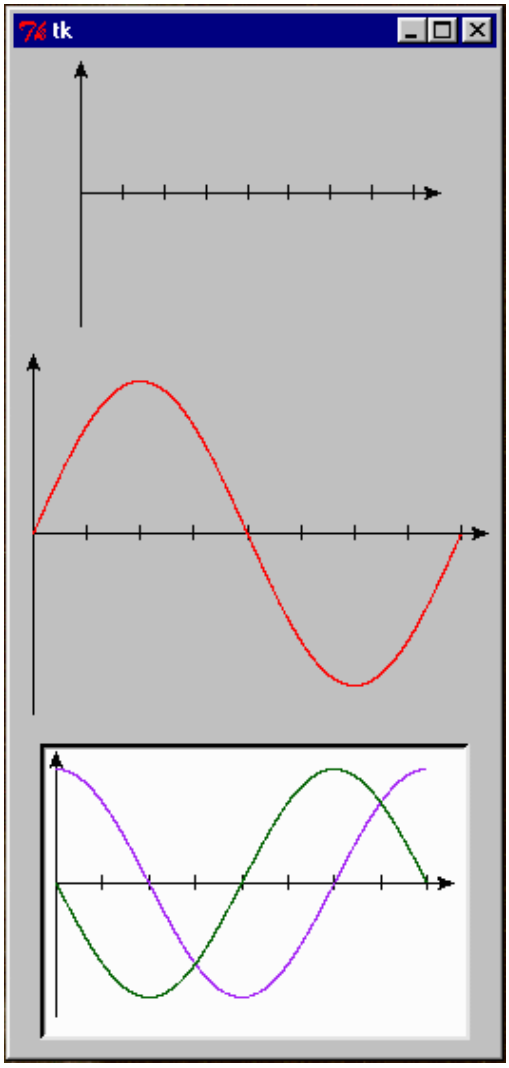
بما أننا لم نضع أي برامتر, الكائن لديه حجمه الإفتراضي, تم تعريفه في منشئ الصنف . لاحظ أننا لم نقم بتعريف أولا نافذة الأصل لكي نضع بها الويدجات . tkinter يسامح هذا النسيان و هو وفره لنا تلقائيا !

```
>>> g2 = OscilloGraphe(haut=200, larg=250)
>>> g2.pack()
>>> g2.traceCourbe()
```

لهذه التعليمات, صنعنا ويدجت ثاني من نفس الصنف, و هذه المرة مع تحديد أبعادها (الطول و العرض, لا يهم الترتيب) .

ثم قمنا بتفعيل الأسلوب **traceCourbe()** المرتبط بهذا الويدجت . لأننا لم نوفر لها أي برامتر, و يظهر الشرط الموجب الذي يتوافق مع القيم الإفتراضية للبرامترات **A** و **f** و **φ** .

```
>>> g3 = OscilloGraphe(larg=220)
>>> g3.configure(bg='white', bd=3, relief=SUNKEN)
>>> g3.pack(padx=5,pady=5)
>>> g3.traceCourbe(phase=1.57, coul='purple')
>>> g3.traceCourbe(phase=3.14, coul='dark green')
```



لفهم تكوين الويدجت الثالث, يجب أن تتذكر الصنف **OscilloGraphe()** تم صنعه بإشتقاق من الصنف **Canvas()** . و هو يرث جميع خصائصه, و الذي يتيح لنا إختيار لون الخلفية, و الحدود ... إلخ, بإستخدام نفس البرامترات التي وضعناها عندما كونا اللوحة .

ثم قمنا بعرض قطعتين متعاقبتين, و ذلك عن طريق إستدعاء الأسلوب **traceCourbe()** مرتين, و التي نقدم البرامترات للتطور و اللون .

تمرين

قم بصنع ويدجت رابعة, بقياس 400 × 300, بلون خلفية أصفر, و قم بإظهار العديد من المنحنيات الموافقة لترددات مختلفة .

لقد حان الوقت لتحليل هيكل الصنف الذي سمحنا له بتمثيل هذه الويدجات . سوف نقوم الأن بحفظ هذا الصنف في وحدة تدعى **oscillo.py** (أنظر للصفحة 252).

المواصفات

نحن نريد الأن تعريف صنف لويدجت جديد, قادر على إظهار تلقائيا رسوم بيانية إطالة/وقت متوافقة مع حركات الذبذبات .

يجب على هذا الويدجت أن يقوم قادر على صنع مثيل في أي وقت . و يجب عليه أن يظهر محورين ديكارتي **X** و **Y** مع أسهم . و يمثل المحور **X** مرور الوقت بالثانية الواحدة و سوف تكون مجهزة ب 8 فترات .

و سوف يتم ربط الأسلوب **traceCourbe()** بهذا الويدجت . و هذا قد يكون سبب الرسوم البيانية إستطالة/الزمن لحركة الإهتزازية, و التي يجب علينا تقديم التواتر (هزة تترواح ما بين 0.25 إلى 10 هرتز) و مرحلة (ما بين 0 و 2π راديان) و السعة (بين 1 إلى 10 : نطاق تعسفي) .

التطبيق

* السطر 4 : تم صنع الصنف **OscilloGraphe()** بالإنشقاق من الصنف **Canvas()** . و هو يرث جميع خصائصه : يمكن تكوين كائنات لهذا الصنف الجديد بإستخدام العديد من الخيارات المتاحة بالفعل للصنف **Canvas()** .

* السطر 6 : إن الأسلوب المنشئ يستخدم ثلاثة برامترات, و كلها إختيارية, لأن لكل واحدة منها قيمة إفتراضية . يستخدم البرامتر **boss** لتلقي الإشارة فقط لمرجع نافذة (أنظر الأمثلة أدناه) .

البرامترا **larg** و **haut** (للعرض و الإرتفاع) لتعيين قيم لخيارات **width** و **height** للوحة الأصل, في لحظة تمثيله .

* السطران 9 و 10 : كما قلنا مرارا و تكرارا, منشئ الصنف المشتق سوف يبدأ تقريبا دئما بتفعيل المنشئ للصنف الأصل . نحن لا نستطيع في الحقيقة إرث جميع الوظائف للصنف الأصل, إذ هذه الوظائف تم تنفيذها و تهيئتها .

- لقد قمنا بتفعيل المنشئ للصنف **Canvas()** في السطر 9, و قمنا بإضافة خيارين له في السطر 10 . لاحظ أننا كنا نستطيع أن نختصر هذان السطران في سطر واحد و هو

Canvas.__init__(self, width=larg, height=haut)

و كما شرحنا (أنظر إلى الصفحة Error: Reference source not found), يجب علينا تمرير للمنشئ مرجع المثيل self كبرامتر أول .

* السطر 11 : إنه من الضروري حفظ البرامترات **larg** و **haut** في متغير مثيل, لأننا يجل علينا الوصول إليه أيضا في الأسلوب **traceCourbe()** .

* السطران 13 و 14 : لرسم محاور **X** و **Y**, سوف نستخدم البرامترات **lang** و **haut**, و يتم وضع هذه المحاول تلقائيا على الأبعاد . يستخدم الخيار **arrow=LAST** لعرض سهم صغير في نهاية كل خط .

* الأسطر من 16 إلى 19 : لرسم مقياس أفقي, نبدأ من خفض 25 بكسل من عرض المتاح, لكي يتم تشكيل مساحات في كل الجانبين . ثم نقسمها إلى 8 أقسام, وهو تصور شكل عمودي لثمانية خطوط صغيرة .

* السطر 21 : يمكن إستدعاء الأسلوب **traceCourbe()** مع أربعة برامترات . كل واحد منها يمكن حذفه, لأن كل البرامترات لديها قيم إفتراضية . و يمكن توفير البرامترات في أي ترتيب , كنا شرحنا في الصفحة 76 .

* الأسطر من 23 إلى 31 : لرسم منحني, المتغير **t** يتم القيم من 0 إلى 1000, و يحسب كل مرة إستطالة **e** المقابلة, بمساعدة الصيغة النظرية (السطر 26) . تم العثور على أزواج القيم **t** و **e** و تم تحجيمهم و تحويلهم إلى إحداثيات **x, y** في الأسطر 27 و 28, ثم تراكمت في قائمة **curve** .

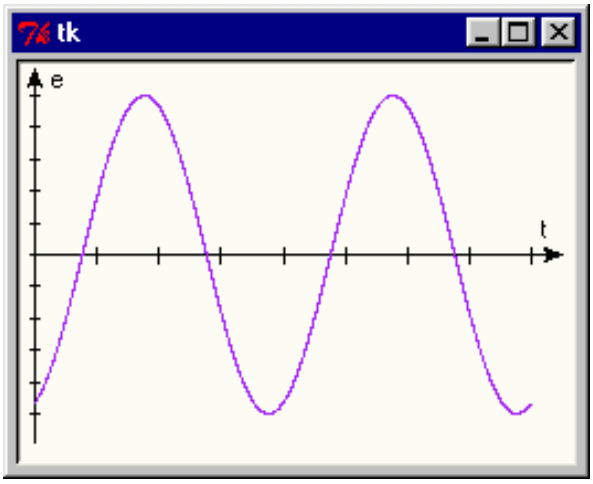
* الأسطر 30 و 31 : يرسم الأسلوب **create_line()** المنحنى المقابل في عملية واحدة, و يقوم بإرجاع رقم ترتيب للكائن الجديد الذي تم تمثيله في اللوحة (و هذا رقم الترتيب سوف يسمح لنا بالوصول إليه بعد ذلك : لمسحه, على سبيل المثال) . الخيار **smooth =1** يحسن مظهر النهائي بتجانسه .

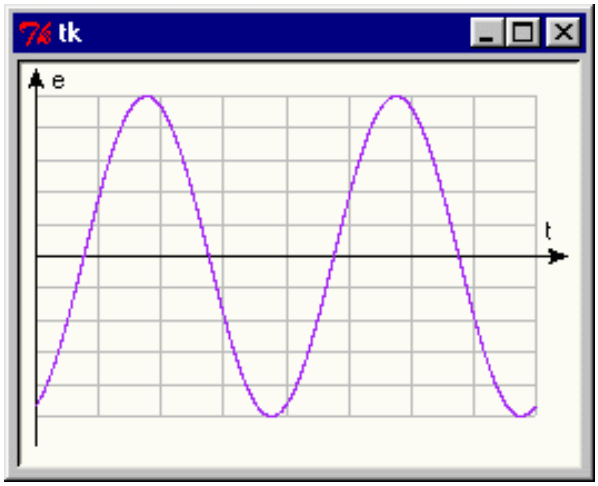
تمارين

13.9 عدل السكريبت بحيث يصبح المحور هو المرجع العمودي يضم إليه أيضا مقياس, مع 5 أجزاء et d'autre de l'origine .

13.10 مثل ويدجات الصنف **Canvas()** الذي يشتق منه, ويدجت الهاص بك بمكنه دمج مؤشرات نصية . ببساطة إستخدم الأسلوب **create_text()** . هذا الأسلوب ينتطر على الأقل ثلاثة برامترات . الإحداثيات **x** و **y** لمكان الذي تريد أن تظهر نصك فيه ثم النص الذي تريد إظهاره بطبيعة الحال . و يمكن تمرير برامترات أخرى بشكل خيارات, لتحديد على سبييل المثال الخط و الحجم . لنرى كيف يعمل هذا, أضف مؤقتا السطر التالي في منشئ الصنف **OscilloGraphe()**, ثم قم بإعادة تشغيل السكريبت :

```
self.create_text(130, 30, text = "Essai", anchor =CENTER)
```





إستخدم هذا الأسلوب لإضافة إلى ويدجت المؤشرات التالية القصوى لمحاور المرجع : **e** (للإستطالة) على طور محور العمودي, و **t** (للوقت أو الزمن) على طول المحور الأفقي . قد تبدو النتيجة على الشكل الأيسر .

13.11 يمكنك إكمال مرة أخرى الويدجت الخاص بك بإظهار شبكة المرجع plutôt que de simples tirets le long des axes. Pour éviter que cette grille ne soit trop visible, vous pouvez colorer ses traits en gris (option fill = 'grey'), comme dans la figure de droite.

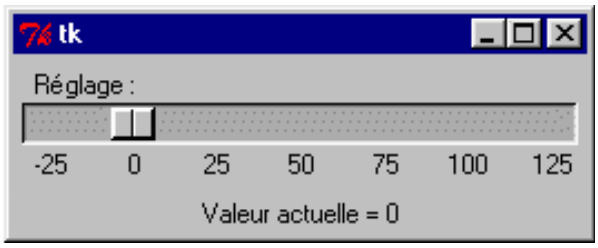
13.12 أكمل الويدجت الخاص بك بإظهار الأرقام .

المؤشرات ، ويدجت مركب

في التمرين السابق, قمت بصنع نوع جديد من الويدجات التي قمت بحفظها في وحدة **oscillo.py** . حافظ على هذه الوحدة سوف تنضم قريبا لمشروع أكبر تعقيدا .

في الوقت الراهن, سوف تقوم بصنع ويدجت أخرى, و هذه المرة أكثر تفاعلا . ستكون نوع من أنواع لوحات التحكم تحتوي على ثلاثة متزلجات و خانة إختيار . مثل سابقتها, يهدف هذا الويدجت لإعادة إستخدامه في تطبيق تجميعي .

عرض ويدجت المقياس (Scale)



دعونا نبدأ أولا من خلال إستكشاف ويدجت القاعدة, و التي لم نستخدمها حتى الأن : الويدجت Scale يشبه المتزحلق الذي يتحلق أمام مقياس . يسمح للمستخدم إختيار سرعة القيمة بأي برامتر .

السكريبت الضغير بالأسفل يوضح لك كيف وضع برامتر و إستخدامها في نافذة :

```
from tkinter import *

def updateLabel(x):
    lab.configure(text='Valeur actuelle = ' + str(x))

root = Tk()
Scale(root, length=250, orient=HORIZONTAL, label ='Réglage :',
      troughcolor ='dark grey', sliderlength =20,
      showvalue =0, from_=-25, to=125, tickinterval =25,
      command=updateLabel).pack()
lab = Label(root)
lab.pack()

root.mainloop()
```

هذه الأسطر لا تتطلب الكثير من التعليق .

يمكنك إنشاء ويدجات Scale بأي حجم كان (الخيار **length**) في إتجاه أفقي (كمافي مثالنا) أو عمودي (الخيار **orient = VERTICAL**) .

الخيار **from_** (إنتبه : لا تنسى الرمز "تحت السطر", لأن هذا إلزامي لكي لا يتم الخلط بين هذه الكلمة و كلمة **from** المحجوزة !) و to لضبط النطاق . يتم تعريف الفترة بين الأرقام في الخيار **tickinterval** ...إلخ

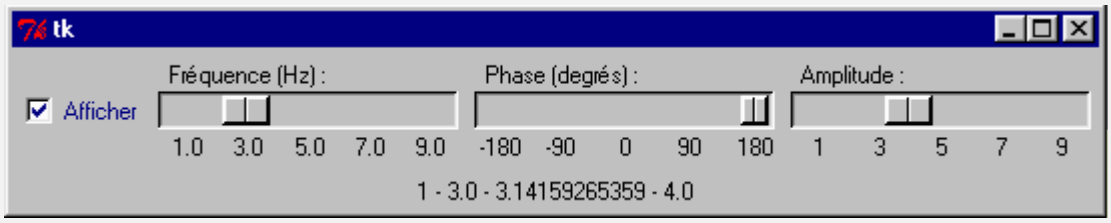
و الدالة تحدد في خيار **command** بإنه يستدعي تلقائيا في كل مرة عندما ستم وضع المؤشر, و المكان الحالي للمؤشر حسب المقيار يتم تمريره كبرامتر . لذا فهو من السهل جدا إستخدام هذه القيمة لتعيين أي معالجة . أنظر على سبيل المثال إلى البرامتر **x** لدالة **updateLabel()** في مثالنا .

الويدجت Scale هي واجهة بديهية للغاية و توفر للمستخدمي برامجك العديد من الإعدادات . سوف نقوم الأن بدمج عدة نسخ منها في صنف ويدجت جديد : لوحة تحكم لإختيار التردد, و الحركة و الطور لحركات الذبذبات, ثم سنقوم بعرض ريم بياني إستطالة/الوقت بمساعدة ويدجت **oscilloGraphe** الذي درسناه في الصفحات السابقة .

بناء لوحة تحكم بثلاثة متزلجات/مؤشرات

مثل سابقتها, السكريبت الموصوف بالأسفل يجب أن يتم حفظه في وحده, و الذي يجب أن يتم تسميته ب **curseurs.py** . و الأصناف التي قمت بحفظها سيتم إعادة إستخدامها (بالإستدعاء) في تطبيق مركب و الذي سوف نتحدث عنه في وقت لاحق . و لاحظوا أننا تستطيع تقصير الكوج بالأسفل لطرق مختلفة (سوف نتحدث عنها لاحقا[66]. نحن لم نحسن الكود, لأن هذا يتطلب دمج مفهومات إضافية (العبارة **lambda**, و التي نفصل أن نتجنبها في الوقت الحالي . أنت تعرف بالفعل أن أسطر الكود الموجود في أسفل السكريبت : لإختبار عملها . سوف تحصل على نافذة مثل هذه

66يمكنك بالطبع حفظ العديد من الأصناف في نفس الوحدة .




```
1#    from tkinter import *
2#    from math import pi
3#
4#    class ChoixVibra(Frame):
5#      """Curseurs pour choisir fréquence, phase & amplitude d'une vibration"""
6#      def __init__(self, boss =None, coul ='red'):
7#        Frame.__init__(self)      # منشئ الصنف الأصل
8#        # تهيئة بعض سمات المثيل
9#        self.freq, self.phase, self.ampl, self.coul = 0, 0, 0, coul
10#       #: متغيرة حالة خانة الإختيار
11#       self.chk = IntVar()          # 'كائن-متغير' tkinter
12#       Checkbutton(self, text='Afficher', variable=self.chk,
13#              fg = self.coul, command = self.setCurve).pack(side=LEFT)
14#       #: تعريف ثلاثة ويدجات من نوع متزلجات
15#       Scale(self, length=150, orient=HORIZONTAL, sliderlength =25,
16#           label ='Fréquence (Hz) :', from_=1., to=9., tickinterval =2,
17#           resolution =0.25,
18#           showvalue =0, command = self.setFrequency).pack(side=LEFT)
19#       Scale(self, length=150, orient=HORIZONTAL, sliderlength =15,
20#           label ='Phase (degrés) :', from_=-180, to=180, tickinterval =90,
21#           showvalue =0, command = self.setPhase).pack(side=LEFT)
22#       Scale(self, length=150, orient=HORIZONTAL, sliderlength =25,
23#           label ='Amplitude :', from_=1, to=9, tickinterval =2,
24#           showvalue =0, command = self.setAmplitude).pack(side=LEFT)
25#
26#     def setCurve(self):
27#       self.event_generate('<Control-Z>')
28#
29#     def setFrequency(self, f):
30#       self.freq = float(f)
31#       self.event_generate('<Control-Z>')
32#
33#     def setPhase(self, p):
34#       pp =float(p)
35#       self.phase = pp*2*pi/360      # تحويل الدرجة -> راديان
36#       self.event_generate('<Control-Z>')
37#
38#     def setAmplitude(self, a):
39#       self.ampl = float(a)
40#       self.event_generate('<Control-Z>')
41#
42#   ####: كود لتجربة الصنف ###
43#
44#   if __name__ == '__main__':
45#     def afficherTout(event=None):
46#       lab.configure(text = '{0} - {1} - {2} - {3}'.\
47#                 format(fra.chk.get(), fra.freq, fra.phase, fra.ampl))
48#     root = Tk()
49#     fra = ChoixVibra(root,'navy')
50#     fra.pack(side =TOP)
51#     lab = Label(root, text ='test')
```

```
52#     lab.pack()
53#     root.bind('<Control-Z>', afficherTout)
54#     root.mainloop()
```

هذه لوحة التحكم تسمح للمستحدمين بضبط القيم البرامترات المعينة (تردد و طور وسعة), و من ثم يمكن إستخدامه لعرض رسم بياني إطالة/الزمن في ويدجت للصنف **OscilloGraphe()** الذي قمنا بصنعه سابقا, كما وضحنا في التطبيق .

تعليقات

* السطر 6 : يستخدم الأسلوب المنشئ برامتر الإختياري **coul** . هذا البرامتر يسمح بإختيار لون غرافيك للوحة التحكم للويدجت . البرامتر **boss** لتلقي مرجع النافذة الأصل الممكنة (سوف نراها لاحقا) .

* السطر 7 : تفعيل منشئ الصنف الأصل (لوراثة وظائفه) .

* السطر 9 : بيان ببعض المتغيرات المثيل . و سيتم تحديد قيمهم بواسطة الأساليب في السطور 29 إلى 49 (معالجة الأحداث) .

* السطر 11 : هذه التعليمة تقوم بتمثيل كائن من صنف **IntVar()**, والذي هو جزء من وحدة tkinter و هو مشابه للأصناف **DoubleVar()** و **StringVar()** و **BooleanVar()** . كل هذه الأصناف تسمح بتعريف متغيرات tkinter, و التي هي في الواقع كائنات, لكنها تتصرف مثل المتغيرات في ويدجات tkinter (أنظر لاحقا) . و بالتالي المرجع في **self.chk** يحتوي على ما يعادل متغير من نوع صحيح, في شكل مستخدم من قبل tkinter . للوصول إلى قمته من البايثون, يجب إستخدام الأساليب الخاصة بهذا الكائن : الأسلوب **set()** يسمح بتعيين قيمة, و الأسلوب **get()** يسمح بإسترجاعها (سوف نراه في السطر 47) .

* السطر 12 : الخيار **variable** لمتغير **checkbutton** يقوم بربط المتغير tkinter الذي قمنا بتعريفه في السطر السابق . نحن لا يمكننا أن نقوم برمجع مباشر لمتغير عادي في تعريق ويدجت tkinter, لأن tkinter كتب بلغة لا تستخدم نفس إتفاقيات البايثون لتنسيق المتغيرات . الكائنات تم بنائها من أصناف متغيرات tkinter ضرورية لضمان الواجهة .

* السطر 13 : الخيار **command** يشير إلى أسلوب الذي يجب على النظام إستدعاءه عندما يقون المستخدم بالنقر من الفأرة على خانة الإختيار .

* السطور من 14 إلى 24 : هذه الأسطر تقوم بتعريف ثلاثة ويدجات متزلجات, في ثلاثة تعليمات متشابهة . و سيكون من الأفضل إختصار هذه تعليمات إلى واحدة فقط, يتم تكرارها ثلاثة مرات بمساعدة حلقة . و هذا يتطلب مفهوم لم أقم بشرحه بعد (الدالات أو العبارات **lambda**), و تعريف معالج الأحداث الذي يتم ربطه بهذه الويدجات ستصبح أكثر تعقيدا . حتى هذا الوقت سوف تبقى التعليمات منفصلة : سنحاول تحسينها لاحقا .

* السطور من 26 إلى 40 : الويدجات الأربعة تم تعريفهم في الأسطر سابقة لدى كل واحدة منها خيار **command** . لكل واحدة منها, أسلوب إستدعاء الخيار **command** مختلف : مربع الإختيار يفعل الأسلوب **setCurve()**, فيقوم المتزلج الأول بتنشيط الأسلوب **()** **setFrequency**, و المتزلج الثاني يفعل الأسلوب **setPhase()**, و أما الثالث فيقعل الأسلوب **setAmplitude()** . لاحظ أن الخيار **command** لويدجات **Scale** تمرر برامتر للأسلوب المرتبط (موضع الحالي للمتزلج), لذا فقط الخيار **command** لا يقوم بتمرير شيئ في حالة الويدجت **Checkbutton** .هذه الأساليب الأربعة (و التي هي معالجة الأحداث التي تم إنشاؤها بواسطة خانة و 3 متزلجات) و تسبب كل واحدة منها مهمة عنصر جديد[67], و ذلك بإستدعاء الأسلوب **event_generate()** .

عندما يتم إستدعاء هذا الأسلوب, يقوم البايثون بإرجاع لنظام التشغيل نفس رسالة-العناصر التي سوف تظهر إذا ضغط المستخدم على **<Ctrl>** و **<Maj>** و **<Z>** في وقت واحد من لوحة المفاتيح .

و سوف نقوم بإظهار رسالة-العناصر خاصة جدا, الذي تقوم بكشف و معالجة بواسطة معالج الأحداث المتربط بويدجت أخر (أنظر للصفحة التالية) . بهذه الطريقة, وضعنا في المكان نظام الصحيح للإتصال بين الويدجات : في كل مرة يقوم بها المستخدم بتنفيذ إجراء على لوحة التحكم الخاصة بنا, فإنه ينشئ حدث معين, مما يدل على العمل لإنتباه الحاجيات الأخرى .

67في الواقع, يجب علينا أن نسميه رسالة (و الذي هو في حد ذاته إعلام حدث) . يرجى قراءة الشرح في الصفحة برامج يتم التحكم بها بواسطة الأحداث .

> يمكننا إختيار تركيبة مفاتيح أخرى (أو نوع أخر من الأحداث) . و لكم فإن هنالك فرصة ضئيلة جدا أن يقوم المستخدم بإستخدامها عندما يستخدم برنامجنا . و مع ذلك, يمكننا أن ننتج مثل هذا الحدث بأنفسنا كما, كتجربة, عندما يحي الوقت للتحقق من معالج هذا الحدث, و الذي سوف نعيد إستخدامه في وقت لاحق .

* السطور من 42 إلى 54 مثلما فعلنا ل **oscillo.py**, أكملنا هذه الوحدة الجديدة ببضعة أسطر في المستوى الرئيسي . هذه الأسطر تسمح بإختبار التشغيل الصنف : و هي لا تعمل إلا لو شغلت الوحدة مباشرة, كتطبيق مستقل . تأكد من إستخدام هذه التقنية في وحدات الخاصة بك, لأن هذه الممارسة جيد في البرمجة : مستخدم وحدات البناء يمكن في الواقع قد (يعيد) إكتشاف دالاتها بسهولة جدا (عن طريق تشغيل) و كيفية إستخدامها (من خلال تحليل أسطر تعليماتها البرمجية) .

في هذه أسطر للإختبار, قمنا ببناء نافذة أساسية **root** و التي تحتوي على ويدجتان : ويدجت لصنف جديد **ChoixVibra()** و ويدجت للصنف **Label()** .

في السطر 52, قمنا بربط النافذة الرئيسية بمعالج الأحداث : جميع الأحداث من نوع محدد سوف تؤدي إلى إستدعاء الدالة **afficherTout()** .

هذه الدالة هي معالج الأحداث الخاص, و التي يتم تطبقها في كل مرة عندما يكون نوع الحدث **<Maj-Ctrl-Z>** تم كشفه من قبل نظام التشغيل .

كما شرحنا أعلاه, علينا أن نضمن أن يتم صنع مثل هذه الأحداث من قبل كائنات الصنف **ChoixVibra()** , في كل مرة قام المستخدم بتعديل حالة أو أخرى من المتزلجات الثلاثة, أو من خانة الإختيار .

* صمم فقط لإختبار الدالة **afficherTout()** لا تفعل شيئا لكنها تتسبب في عرض قيم المتغيرات المرتبطة بويدجاتنا الأربعة, بإعادة تكون خيار **text** لويدجت من صنف **Label()** .

* السطر 47, التعبير **fra.chk.get()** : لقد رأينا أعلاه أن متغير تخزين حالة كائن خانة الإختيار هو كائن-متغير tkinter . البايثون لا يمكنه قراءة مباشرة محتوى لمتغير مثل هذا, و الذي هو في الواقع واجهة-كائن . لإستخراج القيمة, يجب علينا إستخدام الأسلوب الخاص لهذا الصنف من الكائنات : الأسلوب **get()** .

نشر الأحداث

الية الإتصال التي تم وصفها في الأعلى تتوافق مع تدرج أصناف الويدجات . ستلاحظ أنه يرتبط بأسلوب الذي يطرح هذا الحدث المرتبطه مع ويدجت التي نحن نريد تعرف الصنف, عن طريق self . عامة, رسالة-الأحداث هي في الواقع مرتبطو بويدجت خاصة (على سبيل المثال, يتم ربط ضغطة الفأرة على زر مع هذا الزر), و هذا يعني أن النظام التشغيل سوف يفحص أولا ما إذا كان هناك معالج لهذا النوع من حدث, الذي يرتبط أيضا مع هذا الويدجت . فإذا وجد, فسوف ينشطه, و ينشر رسالة توقف . و إلا, رسالة-الحدث هي التي تعرض على لتوالي على ويدجت الأصل, برتيب تدرجيا, إلى أن يجدها معالج الأحداث, أو حتى يتم الوصول إلى النافذة الرئيسية .

الأحداث الموافقة لضغطات على لوحة المفاتيح (مثل ضغطة **<Maj-Ctrl-Z>** التي تم إستخدامها في تمريننا) يتم دائما شحنها مباشرة إلى النافذة الرئيسية للتطبيق . في مثالنا, معالج الأحدصل لهذا العنصر يجب عليه يرتبط مع نافذة **root** .

تمارين

13.13 الويدجت الجديد الخاص بك يرث جميع خصائص الصنف **Frame()** . يمكنك إذا تعديل مظهره عن طريق تعديل خيارات الإفتراضية لهذا الصنف, بمساعدة الأسلوب **configure()** . حاول على سبيل المثال أن تقوم بإحاطة لوحة التحكم بأربعة بيكسلات بعد إظهار الأخدود (**bd = 4, relief = GROOVE**) . فإذا لم تفهم ما يجب عليك فعله, إستوحي من السكريبت **oscillo.py** (السطر العاشر) .

13.14 إذا قمنا بتعيين القيمة 1 لخيار **showvalue** في ويدجات **Scale()**, فسيتم عرض نطاق موقع الدقيق للمتزحلق . لذا قم بتفعيل هذه الميزة للمتزحلق الذي يتحكم في برامتر "phase - طور" .

13.15 الخيار **troughcolor** لويدجات **Scale()** تسمح بتعريق لون الشريحة . إستخدم هذا الخيار للتأكد من أن لون المتزلجات الثلاثة يتم إستخدامهم كبرامتر لتمثيل ويدجت جديد .

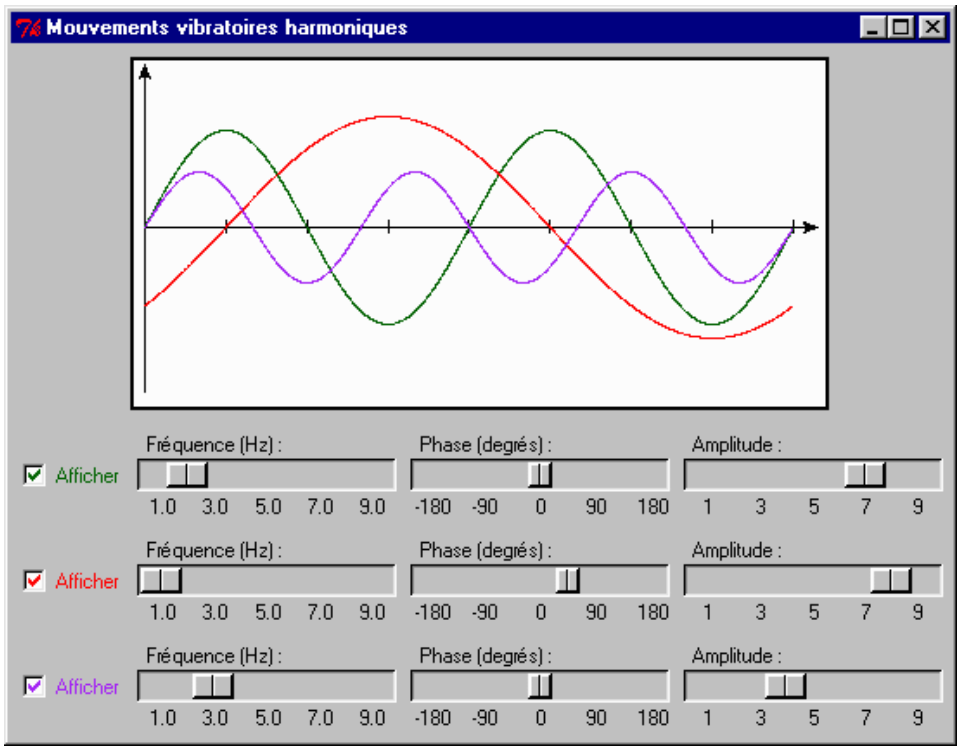
13.16 قم بتعديل السكريبت بحيث يتم إزالة ويدجات المتزلجات ؟؟؟؟؟ (الخياران padx و pady للأسلوب pack()) .

دمج ويدجات المركبة في تطبيق تجميعي

في التمارين السابقة, قمنا ببناء صنفين ويدجت جديدين : الويدجت **OscilloGraphe()**, و هي لوحة خاصة لرسم المخططات الذبذبات, و الويدجت **ChoixVibra()**, هو لوحة تحكم بثلاثة متزلجات يسمحون بإختيار البرامترات الإهتزاز .

هذه الويدجات متوفرة الأن في وحدتي **oscillo.py** و **curseurs.py**[68].

سوف نقوم الأن بإستخدامهم في تطبيق التركيبي : الويدجت **OscilloGraphe()** سوف يقوم بعرض رسم أو رسمين أو 3 رسوم لمخططات متذبذبة, بألوان مختلفة, كل واحدة منهم تحت تحكم الويدجت **ChoixVibra()** .



68و يمكننا أيضا جمع جميع الأصناف التي نصنعها في واحدة واحدة .

السكريبت المقابل يصنع التالي .

نلفت إنتباهكم إلى أن تشغيل التقنية لإحداث تحديث للعرض في لوحة من خلال حدث, في كل مرة يقوم المستخدم بتنفيذ أي حركة في مستوى في أي واحدة من لوحات التحكم .

تذكر أن التطبيقات مصممة لتعمل في واجهة المستخدم الرسومية و هذا يعني "البرنامج يتم التحكم به من خلال الأحداث" (أنظر للصفحة 83) .

في الإعداد لهذا المثال, قررنا بشكل تعسفي بأن عارض الرسومات يسبب حدث معين, مماثل للذي يتم إنشاؤه بواسط نظام التشغيل عندما يقوم المستخدم بعمل أي حركة . في نطاق (كبير جدا) من الأحداث الممكنة, إخترنا واحدا الذي من غير المرجح أن يستخدم لأسباب أخرى, في حين برنامجنا يعمل : بتركيبة المفاتيح **<Maj-Ctrl-Z>** .

عندما قمنا بصنع صنف ويدجت **ChoixVibra()**, و قمنا بدمج التعليمات اللازمة ليتم صنع حدث في كل مرة يقوم فيها المستخدم بتحريك أحدى المتزلجات أو تعديل حالة خانة الإختيار . سوف نقوم الأن بتعريف معالج لهذه الأحداث و نقوم بإدراجه في صنف جديد : الذي سنسميه **montreCourbes()** و يقوم بتحديث الشاشة . بالنظر لهذا الحدث هو ذات صلة به < **enfoncement** الضغطة>, و مع ذلك يجب أن نكتشف في مستوى النافذة الرئيسية للتطبيق .

```
1#    from oscillo import *
2#    from curseurs import *
3#
4#    class ShowVibra(Frame):
5#       """Démonstration de mouvements vibratoires harmoniques"""
6#       def __init__(self, boss =None):
7#          Frame.__init__(self)       # منشئ الصنف الأصل
8#          self.couleur = ['dark green', 'red', 'purple']
9#          self.trace = [0]*3         # قائمة المسارات (منحنيات لرسمها)
10#         self.controle = [0]*3      # قائمة لوحات التحكم
11#
12#         # تمثيل لوحة مع محاول X و Y:
13#         self.gra = OscilloGraphe(self, larg =400, haut=200)
14#         self.gra.configure(bg ='white', bd=2, relief=SOLID)
15#         self.gra.pack(side =TOP, pady=5)
16#
17#         # تمثيل ثلاثة لوحات تحكم (متزبجلت)
18#         for i in range(3):
19#            self.controle[i] = ChoixVibra(self, self.couleur[i])
20#            self.controle[i].pack()
21#
22#         # تعيين الحدث الذي يقوم بعرض المسارات:
23#         self.master.bind('<Control-Z>', self.montreCourbes)
24#         self.master.title('Mouvements vibratoires harmoniques')
25#         self.pack()
26#
27#       def montreCourbes(self, event):
28#          """(Ré)Affichage des trois graphiques élongation/temps"""
29#          for i in range(3):
30#
31#             # أولا, إمسح المسار السابق (إن وجد)
32#             self.gra.delete(self.trace[i])
33#
34#             # ثم, أرسم مسار جديد
35#             if self.controle[i].chk.get():
36#                self.trace[i] = self.gra.traceCourbe(
37#                            coul = self.couleur[i],
38#                            freq = self.controle[i].freq,
39#                            phase = self.controle[i].phase,
40#                            ampl = self.controle[i].ampl)
41#
42#    ####: كود لتجربة الصنف  ###
43#
44#    if __name__ == '__main__':
45#       ShowVibra().mainloop()
```

تعليقات

* السطران 1 و 2 : نحن لسنا بحاجة إلى إستدعاء وحدة tkinter : كم واحدة من هذه الوحدات تدعمه .

* السطر 4 : و بما أننا بدأنا بالتعرف على التقنيات الجديدة, قررنا إنشاء تطبيق نفسه كصنف ويدجت جديد, مشتق من الصنف **Frame()** : حتى نتمكن من دمجة في وقت لاحق في مشاريع أخرى .

* السطور من 8 إلى 10 : لقد قمنا بتعريف بعض متغيرات المثيل (3 قوائم) : المنحنيات الثلاثة تصبح كائنات رسومية, و الألون تم تعريفها مسبقا في قائمة **self.couleur**, و يجب علينا أن نقوم بإعداد قائمة **self.trace** لتخزين المراجع في هذه الكائنات الرسومية, و أخيرا قائمة **self.controle** لتخزين مراجع لوحات التحكم الثلاثة .

* السطور من 13 إلى 15 : تمثيل ويدجت العرض . لأن الصنف **OscilloGraphe()** تم إشتقاقه من صنف **Canvas()**, وهو دائما يمكنه تكوين هذا الويدجت بإعادة تعريف الخيارات الخاصة لهذا الصنف (السطر 13) .

* السطور من 18 إلى 20 : لتمثيل ثلاثة ويدجات "لوحة تحكم", نستخدم حلقة . و مراجعهم يتم حفظهم في قائمة **self.controle** التي تم تحضيرها في السطر 10 . هذه لوحات التحكم يتم إشتقاقهم كعبيد من هذا الويدجت, عن طريق البارمتر **self** . و البرامتر الثاني يقوم بتمرير اللون للمتحكم .

* السطران 23 و 24 : في وقت تمثيله, كل ويدجت tkinter يتلقى تلقائيا سمة **master** التي تحتوي على مرجع النافذة الرئيسية للتطبيق . هذه السمة هي مفيدة بشكل خاص إذا تم إنشاء مثيل للنافذة الرئيسية بواسطو tkinter, كما هو الحال هنا .

تذكر أنه عندما نشغل تطبيق مثيل مباشر على ويدجت مثل **Frame()** على سبيل المثال (و هذا ما فعلناه في السطر 4), tkinter يقوم بتمثيل تلقائيا نافذة الأصل لهذا الويدجت (كائن من صنف **Tk()**).

مثل هذا الكائن الذي تم صنعه تلقائيا, ليس لدينا أي مرجع في الكود للوصول إليه, إذا كان من خلال هذه السمة الرئيسية التي قام tkinter بربطها تلقائيا لكل ويدجت . و نحن نستخدم هذا المرجع لإعادة تحديد عنوان لافتة النافذة الرئيسية (في السطر 24), و إرفاق معالج الأحداث (في السطر 23) .

* السطور من 27 إلى 40 : الأسلوب الذي تم وصفه هنا هو معالج الأحداث **<Maj-Ctrl-Z>** الخاص بتسبب بويدجاتنا **ChoixVibra()** (أو "معالج الأحداث"), في كل مرة يقوم فيها المستخدم بتنفيذ أي حركة على المتزلج أو في خانة الإختيار . و في جميع الأحوال, قد تكون بعض

الرسوم التي يجب حذفها أولا (السطر 28) بمساعدة الأسلوب **delete()** : الويدجت **()OscilloGraphe** يرث هذا الأسلوب لهذا الصنف الأصل **Canvas()** .

ثم, يتم رسم منحنيات جديدة, لكل من لوحات التحكم Afficher « coché la case » . كل واحد من الكائنات لديها في اللوحة رقم مرجعها, يتم إرجلعه عن طريق الأسلوب **traceCourbe()** للويدجت الخاص بنا **OscilloGraphe()** .

أرقام مراجع رسوماتنا يتم تخزينها في قائمة **self.trace** . و هو يسمح بحذف كل واحدة منها على حدة (أنظر إلى التعليمة في السطر 28) .

* السطور من 38 إلى 40 : قيم التذبذب و التردد و السعة يتم إرسالها إلى الأسلوب **()traceCourbe** هي سمات المثيل المقابل لكل واحد من لوحات التحكم الثلاثة, و يتم تخزينهم في قائمة **self.controle** . يمكننا إسترداد هذه السمات بإستخدام صفات الأسماء بالنقاط .

تمارين

13.17 عدل السكريبت, بطريقة للحصول على الشكل بالأسفل (تعرض الشاشة مع شبكة المرجع, ولوحات التحكم المحاطة بالأخدود) :

13.18 عدل السكريبت, بحيث يتم إظهار 4 متحكمات رسومية بدل من 3, بالنسبة للون الرسم الرابع, أختر على سبيل المثال : أزرق, بني ...

13.19 في السطور من 33 إلى 35, لقد حصلنا على قيم الطور و التردد و السعة التي تم إخيارها من قبل المستخدم بكل واحدة من ثلاثة لوحات التحكم, بالوصول المباشر لسمات المثيل المقابل . البايثون يسمح بهذا الإختصار (و هذا معناه تطبيقي) لكن هذه التقنية خطيرة . فهي تنتهك واحدة من التوصيات النظرية العامة "للبرمجة الشيئية", التي توصي بأن الوصول إلى خصائص الكائن بطرق محددة . للإمتثال لهذه التوصية, أضف للصنف **ChoixVibra()** أسلوب التي تستدعي **()valeurs**, و التي تقوم بإرجاع نفق (tuple) يحتوي على قيم الترددات و الطور و السعة التي تم إختيارها . السطور من 33 إلى 35 من هذا السكريبت يجب أن نستبدلها بهذا :

freq, phase, ampl = self.control[i].valeurs()

13.20 أكتب تطبيق صغير الذي يقوم بعرض نافذة مع لوحة و ويدجت متزلج (Scale) . في اللوحة, قم برسم دائرة, و التي يمكن للمستخدم تغيير حجمها بمساعدة المتزحلق .

13.21 أكتب سكريت الذي يقوم بصنع صنفين : صنف التطبيق, الذي يشتق من **Frame()**, و الذي يمثل المنشئ لوحة بحجم 400 × 400 بيكسل, بالإضافة لزران . في اللوحة, قم بعمل مثيل الكائن للصنف **Visage** الذي تم وصفه أدناه .

الصنف **Visage** يعرف الكائنات الرسومية لتمثيل وجوه أشخاص بسيطة . هذه الوجوه تتكون من دائرة رئيسية فيها ثلاثة دائراة بيضاوية الشكل و التي تمثل العينات و الفم (مفتوح) . الأسلوب (إغلاق) يسمح بإستبدال الدائرة البيضاوية التي تمثل الفم بخط أفقي . و الأسلوب (فتح) يسمح بإرجاع الفم على شكل الدائرة البيضاوية .

إثنين من الأزرار يتم تعريفها في صنف التطبيقات الذي ستسمح بفتح و غلق الفم للكائن **Visage** الموجود في اللوحة . يمكنك أن تتعلم من المثال في الصفحة 88 لإقتباس جزء من الكود .

13.22 تكرين التركيب : تطوير قاموس ألوان .

الهدف : صنع برنامج صغير الذي يمكنه أن يساعدك بسرعة و سهولة على بناء قاموس ألوان جديد, التي ستسمح بالوصول إلى أي تقنية لون من خلال الإسم المستخدم بالفرنسية .

السياق : يجب أن نقوم بالتلاعب مع الكائنات الملونة مع tkinter, أنت تعرف أن هذه المكتبة الرسومية تقبل يشير إلى الألوان الأساسية في شكل سلاسل نصية تحتوي على الإسم بالغة الإنكليزية : red و blue و yellow و ... إلخ .

أنت تعرف أن الحاسوب يمكنه معالجة البيانات الرقمية فقط . و هذا يعني عاجلا أم اجلا تعيين أي لون بترميزه إلى رقم . و ينبغي بالطبع إعتماد إتفاقية لهذا, و تختلف من نظام لأخر . واحدة من هذه الإتفاقيات, و هي الكثير شيوعا, تمثيل اللون بثلاثة بيتات, و التي تشير إلى شدة اللون من مكوناته الثلاثة : الأحمر و الأخضر و الأزرق .

و يمكن إستخدام هذه الإتفاقية مع tkinter للوصول إلى أي لون . يمكنك تعيين اللون لأي عنصر رسومي, بمساعدة 7 رموز مثل "#00FA4E" . في هذه السلسلة, الرمز الأول (#) يعني أن القيمة

بالنظام السداسي العشري . و الستة الأرقام التي تليها تشير إلى 3 قيم سداسية عشرية لثلاثة مركبات من الأحمر و الأخصر و الأزرق .

لعرض المراسلات بين أي لون و الكود, يمكنك إكتشاف برامج مختلفة لمعالجة الصور, مثل جيمب و إنكسكيب فهي برامج حرة و مفتوحة المصدر .

لأنه ليس من السهل علينا كبشر حفظ الكودات الساداسية العشرية, لدى tkinter قاموس للتحويل, و الذي يسمح بإستخدام الأسماء الشائعة للعديد من الألوان الأكثر شيوعا, لكن هذا لا بعمل لأسماء الألوان باللغة الإنجليزية . الهدف نمن هذا التمرين هو سهولة صنع برنامج قاموس بالفرنسية, و التي يمكنك إدراجها بعد إذن في أي برنامج خاص بك . تبنيه مرة واحدة, هذا القاموس سيتكون شكله كالتالي :

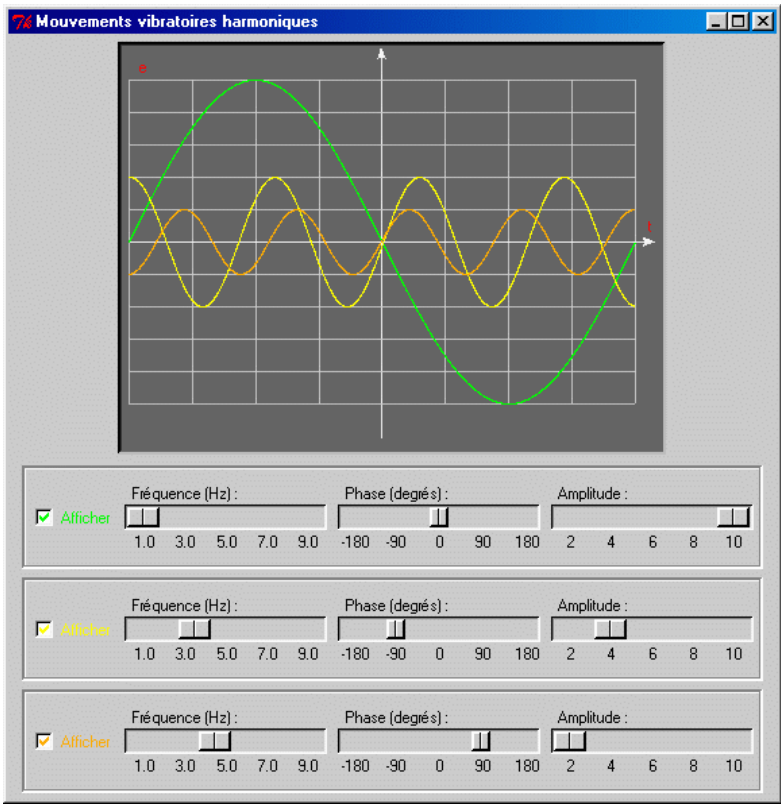
```
{'vert':'#00FF00', 'bleu':'#0000FF', ... etc ...}.
```

المواصفات :

البرنامج الذي نريد تنفيذ هو برنامج رسومي, الذي يتمحور حول صنف . و التي سوف تشمل نافذة مع عدد من حقول الإدخال و الأزرار, بحيث يمكن للمستخدم ترميز ألوان جديدة في كل مرة بكتابة إسمها باللغة الفرنسية في الحقل, و كود السداسي العشري في الحقل الأخر .

عندما يكون القاموس يحتوي على عدد من البيانات, فيكون من الممكن إختباره, و هذا معناه إدخال إسم اللون بالفرنسية و يقوم بإرجاع رمزه السداسي العشري بمساعدة الزر (مع عرض منطقة ملونة) . سوف يتسبب زر في حفظ القاموس في ملف نصي . و زر أخر في إسترجاع القاموس من هذا الملف .

13.23 السكريبت بالأسفل هي أداة مسودة لمشروع رسم مجموعة من الأزهار على الشاشة بطرق مختلفة (و هذا المشروع قد يكون الخطوة الأولى لصنع لعبة) . التمرين هو فحص هذا السكريبت و إكماله . و سوف يكون مكانك هو مواصلة العمل الذي قد بدأه شخص أخر, أو شخص طلب مساعدتك في المشاركة في العمل كفريق .



أ) أبدأ بفحص السكريبت و إضافة التعليقات, و التي هي في سطور الملحوظة : #***, لإظهار أنك فهمت ما يجب القيام به في هذه الأماكن :

```
from tkinter import *
class FaceDom(object):
    def __init__(self, can, val, pos, taille =70):
        self.can =can
        # ***
        x, y, c = pos[0], pos[1], taille/2
        can.create_rectangle(x -c, y-c, x+c, y+c, fill ='ivory', width =2)
        d = taille/3
        # ***
        self.pList =[]
        # ***
        pDispo = [((0,0),), ((-d,d),(d,-d)), ((-d,-d), (0,0), (d,d))]
        disp = pDispo[val -1]
        # ***
        for p in disp:
            self.cercle(x +p[0], y +p[1], 5, 'red')

    def cercle(self, x, y, r, coul):
        # ***
        self.pList.append(self.can.create_oval(x-r, y-r, x+r, y+r, fill=coul))

    def effacer(self):
        # ***
```

```
        for p in self.pList:
            self.can.delete(p)

class Projet(Frame):
    def __init__(self, larg, haut):
        Frame.__init__(self)
        self.larg, self.haut = larg, haut
        self.can = Canvas(self, bg='dark green', width =larg, height =haut)
        self.can.pack(padx =5, pady =5)
        # ***
        bList = [("A", self.boutA), ("B", self.boutB),
                 ("C", self.boutC), ("D", self.boutD),
                 ("Quitter", self.boutQuit)]
        for b in bList:
            Button(self, text =b[0], command =b[1]).pack(side =LEFT)
        self.pack()

    def boutA(self):
        self.d3 = FaceDom(self.can, 3, (100,100), 50)

    def boutB(self):
        self.d2 = FaceDom(self.can, 2, (200,100), 80)

    def boutC(self):
        self.d1 = FaceDom(self.can, 1, (350,100), 110)

    def boutD(self):
        # ***
        self.d3.effacer()

    def boutQuit(self):
        self.master.destroy()

Projet(500, 300).mainloop()
```

ب) عدل هذا السكريبت, ليتناسب مع هذه المواصفات التالية : اللوحة يجب أن تكون حجمها أكبر : 600 × 600 بيكسل .

أزرار التحكم يجب نقلهم إلى اليمين .

حجم النقاط على الوجه يحب أن يكون يختلف نسبته لهذا الوجه .

الخيار 1 :

أبقي فقط زران **A** و **B** . مع كل إستخدان للزر **A** سوف يظهر 3 نردات جديدة (بنفس الطول, صغيرة نوعا ما) رتبت على عمود (عمودي), القيم لهذه النردات يتم إختيارها عشوائيا ما بين 1 و 6 . سيتم وضع كل عمود جديد على يمين سابقه . فإذا طلع 3 نردات و هي 1,2,4 (بأي ترتيب), تظهر في

النافذة (أو في اللوحة) رسالة "ربحت" . سوف يقوم الزر **B** بحذف كل شيئ (لكن ليس النقاط !) من جميع النردات المعروضة .

الخيار 2 :

أبقي فقط زران **A** و **B** . الزر **A** سوف يقوم بعرض 5 نردات متداخلة (هذا معناه مثل نقاط لوجه القيمة 5) . القيم ستكون عشوائية بين 1 و 6, لكن لا يمكن أن تكون مكررة . الزر **B** يقوم بمسح كل شيئ (لكن ليس النقاط) من جميع النردات المعروضة .

الخيار 3 :

أبقي فقط فقط 3 أزرار **A** و **B** و **C** . الزر **A** سيقوم بعرض 13 نرد من نفس الحجم مرتبة في شكل دائرة . كل إستخدام للزر **B** يقوم بتغيير قيمة الزر الأول, ثم الثاني, ثم الثالث ... إلخ . القيمة الجديدة للنرد ستكون قيمة النرد السابق +1, إلا في حالات تكون فيها قيمة السابقة 6 : في هذه الحالة ستكون قيمة المتغير الجديد 1, و إلخ ... . الزر **C** يقوم بمسح كل شيئ (لكن ليس النقاط) من جميع النردات المعروضة .

الخيار 4 :

أبقي فقط فقط 3 أزرار **A** و **B** و **C** . الزر **A** سيقوم بعرض 12 نرد من نفس الحجم مرتبة في شكل سطرين من 6 . قيم النردات ستكون في السطر الأول في ترتيب 1, 2, 3, 4, 5, 6 . قيم النرد الثاني ستكون عشوائية بين 1 و 6 . مع كل إستخدام للزر **B** يقوم بتغيير قيمة نرد من السطر الثاني, و هذه القيمة ستبقى مختلفة بين نرد السطر الأول و السطر الثاني .

إذا كان النرد الأول في السطر الثاني تم الحصول على قيمة مقابله, سوف يتم تغيير القيمة النرد الثاني للسطر الثاني عشوائيا, و هكذا, إلى أن نحصل على 6 وجوه مشابهة للتي فوقها . الزر **C** يقوم بمسح كل شيئ (لكن ليس النقاط) من جميع النردات المعروضة .

14

و بعض الويدجات الإضافية

سوف نقدم هنا بعض الويدجات الجديدة, فضلا عن إستخداماتها المتسخدمة التي تعرفها . نحن لا نتظاهر في كل مرة أننا نبني مرجع ل tkinter : يمكنك العثور على المزيد على المواقع المخصصة . لك إنتبه : ماوراء منظر الوثائق, هذه الصفحات مصممة لتعليمك حسب المثال كيف صنع تطبيق بمساعدة الأصناف و الكائنات . و سوف تكتشف بعض التقنيات البايثون التي لم نتناولها بعد, مثل تعبيرات lambda أو تحديد المهام الضمنية .

أزرار الراديو

ويدجت "أزرار الراديو" يمكن أن يوفر للمستخدم مجموعة من الخيارات الخاصة التبادلية . الذي يطلق عليها قياسا على أزرار الإختيار التي نجدها في الراديوات . تم تصميم هذه الأزرار بحيث تسمح بإختيار إختيار واحد في كل مرة : البقية ظهرت بشكل تلقائي .

الميزة الأساسية من هذه الويدجات هي إستخدامها دائما في مجموعات . جميع أزرار الراديو ينتمون إلى نفس المجموعة المرتبطة بواحدة و هي أيضا مرتبط ب متغير tkinter, لكن كل واحدة تعين قيمة معينة .

عندما يقوم المستخدم بإختيار أحد الأزرار, القيم المقابلة لهذا الزر يتم تعيينه لمتغير tkinter المشترك .

```
1#      from tkinter import *
```

```
2#
3#    class RadioDemo(Frame):
4#      """Démo : utilisation de widgets 'boutons radio'"""
5#      def __init__(self, boss =None):
6#        """Création d'un champ d'entrée avec 4 boutons radio"""
7#        Frame.__init__(self)
8#        self.pack()
9#        #حقل إدخال يحتوي على نص صغير :
10#       self.texte = Entry(self, width =30, font ="Arial 14")
11#       self.texte.insert(END, "La programmation, c'est génial")
12#       self.texte.pack(padx =8, pady =8)
13#       #الإسم الفرنسي و الإسم التقني للأنماط الأربعة للخطوط
14#       stylePoliceFr =["Normal", "Gras", "Italique", "Gras/Italique"]
15#       stylePoliceTk =["normal", "bold", "italic"  , "bold italic"]
16#       #النمط الحالي يتم تخزينه في 'objet-variable' tkinter ;
17#       self.choixPolice = StringVar()
18#       self.choixPolice.set(stylePoliceTk[0])
19#       #صنع أربعة أزرار راديو :
20#       for n in range(4):
21#         bout = Radiobutton(self,
22#                    text = stylePoliceFr[n],
23#                    variable = self.choixPolice,
24#                    value = stylePoliceTk[n],
25#                    command = self.changePolice)
26#         bout.pack(side =LEFT, padx =5)
27#
28#     def changePolice(self):
29#       """Remplacement du style de la police actuelle"""
30#       police = "Arial 15 " + self.choixPolice.get()
31#       self.texte.configure(font =police)
32#
33#   if __name__ == '__main__':
34#     RadioDemo().mainloop()
```

تعليقات

* السطر 3 : هذا المرة أيضا, فضلنا بناء تطبيقنا الصغير كصنف مشتق من صنف **Frame()**, و الذي يسمح لنا بالإندماج بسهولة في تطبيق أكثر أهمية .

* السطر 8 : عامة, نطبق أساليب تحديد المواقع الويدجات (**pack()** أو **grid()** أو **place()**) بعد تمثيلهم, الذي يسمح لنا بإختيار بحرية موقعه في داخل النافذة الأصل . و كما يظهر هنا, فمن الممكن التنبؤ بمواقعهم في منشي الويدجت .

* السطر 11 : ويدجات صنف الإدخال لديها العديد من الأساليب للوصول إلى سلسلة لنصية المعروضة . الأسلوب **get()** يسمح بإسترداد السلسلة بأكملها . الأسلوب **insert()** يسمح بإضافة حروف جديدة إلى أي مكان (و هذا يعني في البداية أو في النهاية أو حتى ضمن السلسلة الموجود إن وجدت) . هذا الأسلوب يستخدم برامترين, الأول يشير إلى مكان الإضافة (إستخدم 0 لإضافته

إلى البداية, **END** لإضافته إلى نهاية السلسلة) . الأسلوب **delete()** يسمح بمسح كل أو جزء من السلسلة . و هي تستخدم نفس البرامترات السابقة (أنظر صفحة مشروع "رمز الألوان", صفحة 191) .

* السطران 14 و 15 : بدل من تمثيل في تعليمات منفصلة, نحن نفضل بصنع 4 أزرار بمساعدة حلقة . الخيارات الخاصة لكل واحد منهم يتم أولا تحظيرهم في قائمتين **stylePoliceFr** و **stylePoliceTk** : الأولى تحتوي على نصوص صغيرة التي يجب أن تظهر بجانب كل زر, و الثانية تحتوي على قيم التي ينبغي أن ترتبط معها .

* السطران 17 و 18 : كما شرحنا في الصفحات السابقة, الأزرار الأربعة تشكل فريق حول المتغير المشترك . هذا المتغير يأخذ القيمة المرتبطة مع زر الراديو الذي إختاره المستخدم . نحن لا نستطيع إيتخدام متغير مستقل لملئ هذا الدور, لأن السمات كائنات tkinter الداخلية لا يمكن الوصول إليه إلا من أساليب مخصصة . مرة أخرى, نحن إستخدمنا هنا كائن-متغير tkinter, من نوع سلسلة نصية, والذي قمنا بتمثيله من صنف **StringVar()**, و التي أعطيناها قيمة إفتراضية في السطر 18 .

* السطور من 20 إلى 26 : نحن قمنا بتمثيل 4 أزرار راديو . كل واحد منه يتم تعيينه مع ملصق (Label) و قيمة مختلفة, لكن جميعهم مرتبطين بنفس متغير tkinter مشترك (**self.choixPolice**) . لكنها تستدعي أسلوب **self.changePolice()**, في كل مرة بقوم فيها المستخدم بإجراء ضغطة من الفأرة على واحدة أو أخرى .

* الأسطر من 28 إلى 31 : يتم تغيير الخط بواسطة إعادة تكوين خيار الخط لويدجت الإدخال . هذا الخيار ينتظر نفق (tuple) تحتوي عى إسم الخط, و حجمه و ربما أسلوبه . فإذا كان إسم الخط لا يحتوي على فراغات, النفق (tuple) يمكنه أيضا إستبداله بسلسلة نصية . على سبيل المثال :

('Arial', 12, 'italic')

('Helvetica', 10)

('Times New Roman', 12, 'bold italic')

"Verdana 14 bold"

"President 18 italic"

(أنظر أيضا الأمثلة في صفحة Error: Reference source not found).

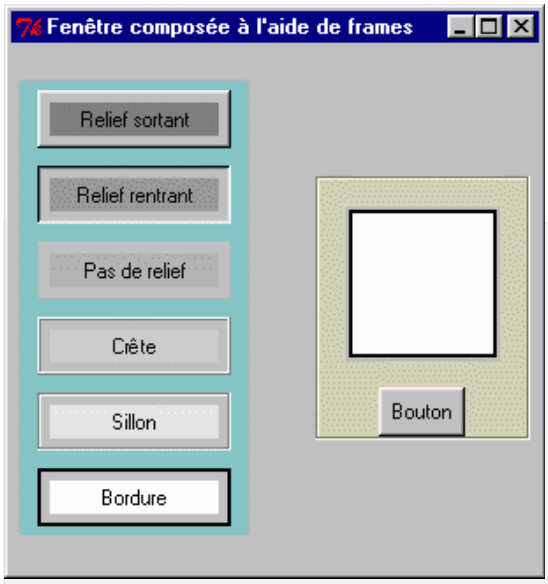
إستخدام الإطارات لتركيب نافذة

أنت تعرف بالفعل إستخدام أصناف ويدجات **Frame()** (إطار باللغة العربية), بما في ذلك إنشاء ويدجات جديدة معقدة بالإشتقاق .

السكريبت الصغير بالأسفل يبين لك فائدة هذا الصنف لإعادة تجميع مجموعات من الويدجات و ترتيبها بطريقة معينة داخل النافذة . و هذا يظهر لك أيضا إستخدام خيارات زخرفية معينة(الحدود...إلخ) .

لإنشاء نافذة التي على الجانب, إستخدمنا إطارين **f1** و **f2**, بطريقة لصنع مجموعتين من الويدجات, واحدة على اليمين و الأخرى على اليسار . و لقد لونا هذان الإطارات لتسليط الضوء عليهم بشكل جديد, ولكن هذا ليس ضروريا .

الإطار **f1** يحتوي على 6 إطارات أخرى, و التي تحتوي كل واحدة منها على ويدجت من صنف **Label()** . الإطار **f2** يحتوب على ويدجت **Canvas()** و ويدجت **Button()** . الألوان و التقليم هي خيارات بسيطة .




```
1#     from tkinter import *
2#
3#     fen = Tk()
```

```
4#    fen.title("Fenêtre composée à l'aide de frames")
5#    fen.geometry("300x300")
6#
7#    f1 = Frame(fen, bg = '#80c0c0')
8#    f1.pack(side =LEFT, padx =5)
9#
10#   fint = [0]*6
11#   for (n, col, rel, txt) in [(0, 'grey50', RAISED, 'Relief sortant'),
12#                              (1, 'grey60', SUNKEN, 'Relief rentrant'),
13#                              (2, 'grey70', FLAT, 'Pas de relief'),
14#                              (3, 'grey80', RIDGE, 'Crête'),
15#                              (4, 'grey90', GROOVE, 'Sillon'),
16#                              (5, 'grey100', SOLID, 'Bordure')]:
17#       fint[n] = Frame(f1, bd =2, relief =rel)
18#       e = Label(fint[n], text =txt, width =15, bg =col)
19#       e.pack(side =LEFT, padx =5, pady =5)
20#       fint[n].pack(side =TOP, padx =10, pady =5)
21#
22#   f2 = Frame(fen, bg ='#d0d0b0', bd =2, relief =GROOVE)
23#   f2.pack(side =RIGHT, padx =5)
24#
25#   can = Canvas(f2, width =80, height =80, bg ='white', bd =2, relief =SOLID)
26#   can.pack(padx =15, pady =15)
27#   bou =Button(f2, text='Bouton')
28#   bou.pack()
29#
30#   fen.mainloop()
```

تعليقات

* الأسطر من 3 إلى 6 : لتبسيط المظهر إلى أقصى حد ممكن, لم نبرمج هذا المثال كصنف جديد . لاحظ في السطر 5 فائدة الأسلوب **geometry()** لتعيين حجم النافذة الرئيسية .

* السطر 7 : تمثيل الإطار الأيسر . يتم تحديد لون الخلفية (الون أزرق فاتح) عن طريق البرامتر **bg** (الخلفية -background) . هذه السلسلة النصية تحتوي وصف سداسي العشري لثلاثة مركبات الأحمر و الأخضر و الأزرق من اللون المطلوب . بعد الرمز **#** للإشارة إلى أن القيمة الرقمية هي بالنظام السداسي العشري, و هنالك ثلاثة مجموعات مع الأرقام و الحروف . كل واحد من هذه المجموعات يحتوي على رقم ما بين 1 و 255 . 80 يقابل 128 و c0 يقابل 192 بالنظام العشري . في مثالنا, المركبات الأحمر و الأخضر و الأزرق التي تحتاجها لتمثيل اللون هي على التوالي 128 و 192 و 192. **padx =5**

لنطبق هذه التقنية, يتم الحصول على الأسود مع **#000000** , و الأبيض مع **#ffffff** , و الأحمر **#ff0000**, و الأزرق الداكن مع **#000050** ... إلخ .

* السطر 8 : بما أننا طبقنا الأسلوب **pack()**, سيتم تغيير إطار تلقائيا حسب محتوياته . الخيار **side =LEFT** ليضع المحتويات على يسار النافذة الرئيسية . و الخيار **padx =5** يضع مساحة 5 بيكسلات على اليمين و اليسار (فارغة) (و يمكننا ترجم "**padx**" ب التباعد الأفقي) .

* السطر 10 : في إطار **f1** الذي نقوم بإعداده, محن سنقنو بجمع 5 إطارات متشابهة تحتوي على واحدة منها على ملصق . الكود المقابل سيكون أكثر بساطة و أكثر فعالية إذا قمنا بتمثيل هذه الويدجات في قائمة بدل من متغيرات مستقلة . نحن نعد الأن قائمة مع 6 عناصر و التي سنسبدلها لاحقا .

* السطور من 11 إلى 16 : لبناء 6 إطارات متشابهة, سوف نقوم بإستعراض قائمة من 6 أنفاق تحتوي على الخاصيات التي ينفرد بها كل إطار . كب واحد من هذه الأنفاق تتكون من 4 عناصر : مؤشر و ثابت tkinter يعرف نوع تخفيف, سلسلتين نصيتين تصف لون و الكتابة في الملصق .

الحلقة for يتم تنفيذها 6 مرت على 6 عناصر من القائمة . مع كل تكرار, محتوى إحدى الأنفاق يتم تعيينه للمتغيرات **n** و **col** و **rel** و **txt** (ثم يتم تنفيذ تعليمات الأسطر من 17 إلى 20) .

> تدوير قائمة من الأنفاق بإستخدام حلقة *for* و الذي هو بناء مدمج خاص, الذي يسمح بأداء العديد من المهام مع عدد قليل جدا من التعليمات .

* السطر 17 : الإطارات 6 يتم تمثيلهم كعناصر للقائمة **fint** . كل و كل واحدة تم تزنيه بحدود من 2 بيكسل واسع مع وجود تأثير التخفيف .

* السطور من 18 إلى 20 : الملصقات جميعها بنفس الحجم, لكن نصوص التي بداخلها و ألوانها هي المختلفة . و نحن إستخدمنا الأسلوب **pack()**, لحجم الملصق التي يتم تحديدهها بإطارات صغير . و التي بدورها تحدد طول الإطار الذي يتم تجميعه (الإطار **f1**) . الخيارات **padx** و **pady** يسمحون بحجز مساحة صغيرة حول كل ملصق, و مساحة صغيرة حول كل إطار صغير . الخيار **side =TOP** يضع كل الإطارات الستو الصغير واحد تحت الأخر في إطار الذي يحتويهم **f1** .

* السطور 22 و 23 : إعداد الإطار **f2** (الإطار الأيمن) . لونه سيكون أصفر, و سنقوم بإحاطة حدود حول زحرفة الأخدود .

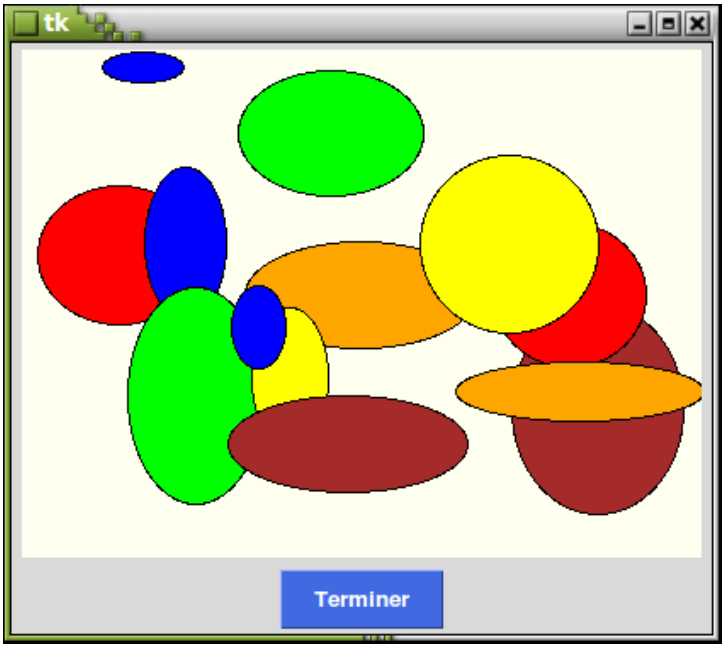
* الأسطر من 25 إلى 28 : الإطار **f2** تحتوي على لوحة و زر . لاحظ مرة أخرى أن إستخدام الخيارات **padx** و **pady** لترك مساحة حول الويدجات (على سبيل المثال, أنظر لحالة أحد الزر, التي لم تستخدم هذا الخيار, و بالتالي فهو يأتي مع حافة الإطار المحيطة به) . كما فعلنا للإطارين السابقين, وضعنا حافة حول اللوحة . إعلم أن الويدجات الأخرى تقبل هذا النوع من الزخرفة مثل : الأزرار و حقول الإدخال ...إلخ .

كيفية نقل رسومات بمساعدة الفأرة

الويدجت اللوحة هو واحدة من نقاط القوة لمكتبة الرسومية tkinter . فهو يشمل على مجموعة كبيرة جدا من الأدوات الفعالة التي تعالج الرسوم . السكريبت أدناه ليظهر لك بعض التقنيات الأساسية . فإذا أردت معرفة المزيد, خاصة فيما يتعلق بمعالجة الرسومات التي تتكون من عدة أجزاء, يرجى الرجوع إلى كتاب مرجي للتعامل مع tkinter .

في بداية تطبيقنا الصغير, مجموعة من الرسومات المرسومة عشوائيا على اللوحة (و هي مجموعة من الدوائر البيضاوية الملونة) . و يمكنك نقل أي واحدة من هذه الرسومات من خلال فأرتك .

عندما يتم نقل أحد الرسومات, il passe à l’avant-plan par rapport aux autres, et sa bordure apparaît plus épaisse pendant toute la durée de sa manipulation.



لفهم هذه التقنية المستخدمة, يجب أن تتذكر برنامج يستخدم الواجهة الرسومية (يتم التحكم به بواسطة الأحداث), (أعد النظر إذا أردت للتفسيرات في صفحة 82) . في هذا التطبيق, سوف نقوم بوضع تقنية التي تتفاعل مع الأحداث : "ضغط على الزر الأيمن للفأرة", "تحريك الفأرة و الزر الأيمن لا يزال مضغوط", "تحرير الزر الأيمن" .

يتم إنشاء هذه الأحداث من قبل نظام التشغيل و بدعم من واجهة tkinter . عملنا البرمجي هو ربطها بمعالجات مختلفة (دالات أو أساليب) .

لتطوير هذا التطبيق الصغير بعد "فلسفة الكائن", و نحن نفضل صنع صنف جديد **Bac_a_sable**, الذي يشتق من لوحة الأساسية, و يتم إدراج الوظائف المطلوبة, بجلا من برمجة هذه الوظائف في مستوى البرنامج الرئيسي, سوف نبنيها عل لوحة عادية . و بالتالي ننتج كود قابل لإعادة الإستخدام .

```
from tkinter import *
from random import randrange


class Bac_a_sable(Canvas):
```

```
"Canevas modifié pour prendre en compte quelques actions de la souris"
def __init__(self, boss, width=80, height=80, bg="white"):
    #إستدعاء منشئ الصنف الأصل
    Canvas.__init__(self, boss, width=width, height=height, bg=bg)
    #ربط الأحداث " الفأرة" بعذا الويدجت :
    self.bind("<Button-1>", self.mouseDown)
    self.bind("<Button1-Motion>", self.mouseMove)
    self.bind("<Button1-ButtonRelease>", self.mouseUp)

def mouseDown(self, event):
    "Opération à effectuer quand le bouton gauche de la souris est enfoncé"
    self.currObject =None
    #event.x و event.y تحتوي على إحداثيات نقرة الفأرة
    self.x1, self.y1 = event.x, event.y
    #<find_closest> تثوم بإرجاع مرجع الرسم الأقرب
    self.selObject = self.find_closest(self.x1, self.y1)
    #تغيير سمك محيط الرسم "
    self.itemconfig(self.selObject, width =3)
    #<lift> تمرير الرسم إلى المقدمة
    self.lift(self.selObject)

def mouseMove(self, event):
    "Op. à effectuer quand la souris se déplace, bouton gauche enfoncé"
    x2, y2 = event.x, event.y
    dx, dy = x2 -self.x1, y2 -self.y1
    if self.selObject:
        self.move(self.selObject, dx, dy)
        self.x1, self.y1 = x2, y2

def mouseUp(self, event):
    "Op. à effectuer quand le bouton gauche de la souris est relâché"
```

```
    if self.selObject:
      self.itemconfig(self.selObject, width =1)
      self.selObject =None

if __name__ == '__main__':     # ---- Programme de test ----
  couleurs =('red','orange','yellow','green','cyan','blue','violet','purple')
  fen =Tk()
  #وضع في اللوحة - رسم من 15 شكل بيضوي ملون
  bac =Bac_a_sable(fen, width =400, height =300, bg ='ivory')
  bac.pack(padx =5, pady =3)
  #: زر الخروج
  b_fin = Button(fen, text ='Terminer', bg ='royal blue', fg ='white',
          font =('Helvetica', 10, 'bold'), command =fen.quit)
  b_fin.pack(pady =2)
  #: رسم 15 شكل بيضوي مع إحدثيات و لون عشوائيان
  for i in range(15):
    coul =couleurs[randrange(8)]
    x1, y1 = randrange(300), randrange(200)
    x2, y2 = x1 + randrange(10, 150), y1 + randrange(10, 150)
    bac.create_oval(x1, y1, x2, y2, fill =coul)
  fen.mainloop()
```

تعليقات

السكريبت يحتوي على أساسيات تعريف صنف رسومي مشتق من **Canvas()** .

هذا الصنف الجديد من المرجح أن يتم إعادة إستخدامه في مشاريع أخرى, و بالتالي إحتبار هذه الصنف في البنية الكلاسيكية **if __name__ ="__main__":** و بالتالي يمكن إستخدام هذا السكريبت كما هو, و وحدة لإستدعاءها لتطبيقات أخرى .

منشئ ويدجت الخاص بنا الجديد **Bac_a_sable()** ينتظر مرجع لويدجت الأصل (**boss**) كبرامتر أول, كما في الإتفاقية المعتادة . و يستدعي منشئ الصنف الأصل, ثم ينفذ الالية المحلية .

في هذه الحالة, فإنه تم ربط ثلاثة معرفات أحداث **<Button-1>** و **<Button1-Motion>** و **<Button1-ButtonRelease>** مع أسماء ثلاثة أساليب مختارة كمعالجة لهذه الأحداث[69].

عندما يضغط المستخدم على الزر الأيمن للفأرة, فإنه يتم تفعيل الأسلوب **mouseDown()**, و يقوم النظام التشغيل بتمرير في البرامتر كائن **event**, الذي يحتوي على سمات **x** و **y** و التي هي إحداثيات المؤشر في موقع اللوحة, الذي حدد في وقت الضغطة .

نحن نقوم بتخزين مباشرة هذه الإحداثيات في متغير مثيل **self.x1** و **self.x2**, لأننا في حاجة إليها في مكان أخر . ثم قمنا بإستخدام الأسلوب **find_closest()** لويدجت اللوحة, التي قمنا بإرجاع مرجع رسم الأقرب . هذا الأسلوب يقوم دائما بإرجاع مرجع, حتى لو كانت ضغطة الزر ليست داخل رسم .

البقية سهلة الفهم : مرجع الرسم تم حفظه في متغير مثيل, و يمكننا إستخدام أساليب أخرى للوحة الأساسية لتعديل خصائصه . و قي هذه الحالة, نحن نستخدم الأسلوب **itemconfig()** و **lift()** لزيادة رشاقة المخطط و نمرره للمقدمة .

يتم إستدعاء "ناقل" الرسم من خلال الأسلوب **mouseMove()**, والذي يتم إستدعائه في كل مرة بتم فيها تحريك الفأرة و الزر الأيسر مضغوط . الكائن **event** يحتوي هذه المرة أيضا على إحداثيات المؤشر الفأرة, في نهاية النقل . و نحن نستخدمها لحساب الفرق بين الإحداثيات الجديد و القديمة, من أجل تمرير للأسلوب **move()** لويدجت اللوحة, و التي سوف تنقل نفسها .

ونحن لا يمكننا إستدعاء هذا الأسلوب إذا كان هنالك تنفيذ كائن موجود (هذا دور متغير مثيل **selObject**) . و نحن نتأكد أيضا من أن الإحداثيات الجديدة المكتسبة تم حفظها .

الأسلوب **moveUp()** تنهي العمل . عندما يتم نقل الرسم إلى وجهته, يبقى فقط إلغاء تحديد و الإنتقال إلى سمك الأولي .

و هذا لا يمكن أن يعتبر إذا مان هنالك بالفعل إختيار, بطبيعة الحال .

---

[69] تذكير : معالج الأحداث لا ينقل الرسائل, إلا إذا وقعت أحداث محددة في اللوحة . نقرات الفأرة المضغوطة خارجا لا تقوم بأي تأثير .

في جسم البرنامج الإختبار, قمنا بتمثيل 15 رسم دون القلق حول حفظ مراجعه في متغيرات . يمكنك فعل هذا لأن tkinter تقوم بحفظ مرجع داخل لكس من هذه الكائنات (أنظر صفحة 98) .

> ملاحظة : إذا كنت تعمل مع مكتبات رسومية أخرى, فربما يجب عليك أن توفر ذاكرة من هذه المراجع .

في جسم البرنامج الإختبار, قمنا بتمثيل 15 رسم دون القلق حول حفظ مراجعه في متغيرات . يمكنك فعل هذا لأن tkinter تقوم بحفظ مرجع داخل لكس من هذه الكائنات (أنظر صفحة 98) .

ويدجات مكملة, ويدجات مركبة

إذا إستكشفت وثائق الكبير الموجود على الأنترنات ل tkinter, فسوف تعرف أنه يوجد توسعات ملحقة , في شكل مكتبات . هذه الملحقات تقدم أصناف ويدجت إضافية التي يمكن أن تكون لا تقدر بثمن لتطوير سريع للتطبيقات المعقدة . لا يمكننا بالطبع أن نتحدث عن كل هذه الويدجات في هذه الدورة . فإذا كنت مهتما, حاول زيارة مواقع ذات صلة لمكتبات Tix و Ttk (و الأخريات) . مكتبة Tix تقترح عليك أكثر من 40 ويدجت إضاف . مكتبة Ttk تهدف إلى "تلبيس" الويدجات مع ثيمات مختلفة (ستيل الأزرار و النوافذ ...إلخ) . و كتب بعض هذه المكتبات بالبايثون, مثل Pmw (Python Mega Widgets) .

و مع ذلك, يمكنك القيام بالكثير من الأشياء من دون البحث عن موارد أخرى لمكتبة tkinter القياسية . يمكنك في الواقع بناء بنفسك أصناف ويدجات جديدة مركبة وفقا لإحتياجاتك . وهذا قد يستغرق بعض العمل في البداية, و لكن عندما تقوم بذلك, فإنك تتحكم بدقة ما في تطبيقك الخاص, و أنك تضمن قابلية المحمولة لجميع الأنظمة التي تقبل البايثون, لأن tkinter توزع كجزء من البايثون القياسية . في الحقيقة, عندما تستخدم مكتبات طرف ثالث, يجب عليك دائما التحقق من توافرها و توافق للالات التي تستهدفها برامجك, و توفر تثبيتها, إذا لزم الأمر .

الصفحات التالي تشرح مبادئ العامة التي يتعين تنفيذها بنفسك للأصناف ويدجت المركبة, مع بعض الأمثلة الأكثر فائدة .

مكعبات كومبو مبسطة

التطبيق الصغير أدناه يوضح لك كيفية بناء صنف جديد من ويدجت من نوع مكعب كومبو . و يعرف أنه ويدجت الذي يربط بين حقل الإدخال و علبة القائمة : يمكن للمستخدم الدخول إلى نظام و الذي عو عناصر القائمة (بالضغط على إسمها), أو إذا كان العنصر غير موجود (يكتب إسم جديد من خلال حقل الإدخال) . نحن بسطنا المشكلة فقط من خلال ترك اللائحة واضحة دئما, لكن من الممكن تحسين هذا الويدجت بحيث يأخذ الشكل التقليدي لحقل الإدخال يرافقه زر صغير يتسبب في ظهور قائمة, و يتم إخفاء هذا في البداية (أمظر للتمرين 14.1 صفحة 326).

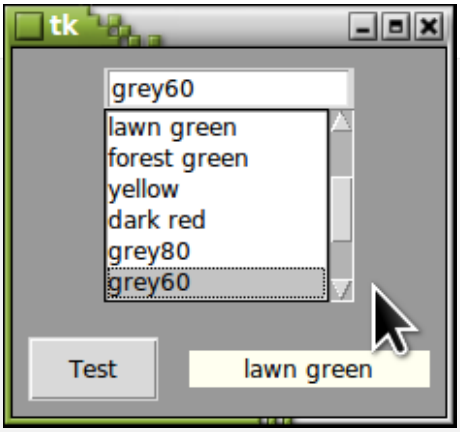
و كما نتصور, ويدجت الخاصة بنا **combo** سيتم تجميعها في كيان واحد من 3 ويدجات tkinter أساسية : حقل إدخال, مربع قائمة (**listbox**) و شريط تمرير .

مربع القائمة و شريط التمرير سيتم ربطهم, لأن شريط التمرير يسمح تجاوز قائمة علبته . و سيتم أيضا التأكد من أن الشريط التمرير سيكون دائما في نفس الإرتفاع العلبة, بصرف النظر عن حجم الذي إخترته لذلك .

سوف نضع علبة قائمة و شريط تمرير جنب إلى جنب في إطار (**Frame**), و وضعه مع مححتوياته في أسفل حقل الإدخال, في إطار أخر عام أكثر . و سوف تكون جميع الويدجات التي لدينا مركبة .

لإختبار الويدجت الخاص بنا, سوف نقوم بتضمين تطبيق بسيط جدا : عندما يقون المستخدم بإختيار لون القائمة (و يمكن أيضا إدخال إسم اللون مباشرة في حقل الإدخال), و هذا سيتسبب بتغير لون تلقائيا لخلفية النافذة الرئيسية .

في هذه نافذة الأساسية, سوف نضيف ملصق و زر, لنظهر لك كيفية يمكنك الوصول إلى الإختيار الذي تم إختياره في مكعب كومبو نفسها (الزر سيقوم بعرض إسم أخر لون تم إختياره) .




```
1#      from tkinter
        import *
2#
3#      class
        ComboBox(Frame):
4#        "Widget composite associant un champ d'entrée avec une boîte de liste"
5#        def __init__(self, boss, item='', items=[], command ='', width =10,
6#                listSize =5):
7#          Frame.__init__(self, boss)  #منشئ الصن الأصل
8#                                  #(<boss> هو مرجع الويدجت « السيد »)
9#          self.items =items          #عناصر لوضعها في علبة القائمة
10#         self.command =command      #دالة لإستدعائها عند النقر أو ضغط زر الإدخال
11#         self.item =item            #عناصر مدخلت أو محددة
12#
13#         #: حقل الإدخال
14#         self.entree =Entry(self, width =width)        #العرض بالعروف
15#         self.entree.insert(END, item)
16#         self.entree.bind("<Return>", self.sortieE)
17#         self.entree.pack(side =TOP)
18#
19#         #: علبة القائمة, شريط تمرير
20#         cadreLB =Frame(self)           #إطار لجمع الويدجتين
21#         self.bListe =Listbox(cadreLB, height =listSize, width =width-1)
22#         scrol =Scrollbar(cadreLB, command =self.bListe.yview)
23#         self.bListe.config(yscrollcommand =scrol.set)
24#         self.bListe.bind("<ButtonRelease-1>", self.sortieL)
25#         self.bListe.pack(side =LEFT)
26#         scrol.pack(expand =YES, fill =Y)
27#         cadreLB.pack()
28#
29#         #: تعبئة علبة القائمة مع العناصر
30#         for it in items:
31#           self.bListe.insert(END, it)
32#
33#       def sortieL(self, event =None):
34#         #إستخراج قائمة العناصر المحددة
35#         index =self.bListe.curselection()     #إرجاع نفق للمؤشر
36#         ind0 =int(index[0])            #نبقي الأول فقط
37#         self.item =self.items[ind0]
38#         #تحديث حقل الإدخال مع العنصر المختار
39#         self.entree.delete(0, END)
40#         self.entree.insert(END, self.item)
41#         #تشغيل الأمر المحدد مع العنصر المختار كبرامتر
42#         self.command(self.item)
43#
44#       def sortieE(self, event =None):
45#         #تشغيل الأمر المحدد مع برامتر تم ترميزه على هذا النحو
46#         self.command(self.entree.get())
```

```
47#
48#     def get(self):
49#         #إرجاع العنصر الأخير المحدد في علبة القائمة
50#         return self.item
51#
52#   if __name__ =="__main__":          #--- برنامج التجربة ---
53#     def changeCoul(col):
54#         fen.configure(background = col)
55#
56#     def changeLabel():
57#         lab.configure(text = combo.get())
58#
59#     couleurs = ('navy', 'royal blue', 'steelblue1', 'cadet blue',
60#             'lawn green', 'forest green', 'yellow', 'dark red',
61#             'grey80','grey60', 'grey40', 'grey20', 'pink')
62#     fen =Tk()
63#     combo =ComboBox(fen, item ="néant", items =couleurs, command
   =changeCoul,
64#                 width =15, listSize =6)
65#     combo.grid(row =1, columnspan =2, padx =10, pady =10)
66#     bou = Button(fen, text ="Test", command =changeLabel)
67#     bou.grid(row =2, column =0, padx =8, pady =8)
68#     lab = Label(fen, text ="Bonjour", bg ="ivory", width =15)
69#     lab.grid(row =2, column =1, padx =8)
70#     fen.mainloop()
```

تعليقات

* الأسطر من 5 إلى 8 : منشئ الويدجت الخاص بنا ينتظر مرجع الويدجت الأصل (**boss**) كبرامتر أول, ثم بقية الإتفاقية المعتادة . البرامترات الأخرى تسمح القدرة على توفير نص إفتراضي في حقل الإدخال (**item**), و يتم إضافة قائمة العناصر إلى علبة (**items**), و تعيين دالة لإستدعائها عندما يقوم المستخدم بالضغط في القائمة, أو من لوحة المفاتيح على زر الإدخال (**command**) . أبقينا الأسماء الإنجليزية لهذه البرامترات, بحيث يمكن لويدجتنا إستخدامه مع نفس الإتفاقيات الويدجات الأساسية التي يشتق منها .

* الأسطر 15و 39 و 40 : أساليب الويدجت الإدخال تم وصفها سابقا (أنظر للصفحة 218) . تذكر ببساطة الأسلوب **insert()** الذي يسمح بإضافة نص إلى حقل الإدخال, دون إزالة النص السابق . البرامتر الأول يسمح بتحديد مكان الإدراج النص الحالي لتأخذ مكانها . و يمكن أن يكون ذا عدد صحيح أو قيمة رمزية (عن طريق الإستدعاء لكامل لوحدة tkinter في السطر 1, سوف نقوم بجلب

مجموعة من المتغيرات العامة, مثل **End**, التي تحتوي على قيم الرمزية و **End** تشير إلى نهاية النص السابق) .

* الأسطر 16 و 24 : و يرتبط الحدثين مع أساليب محلية : يتم تحرير الزر الأيمن للفأرة إذا مؤشره موجود في علبة القائمة (الحدث **<ButtonRelease-1>**) و إذا ضغطنا على زر الإدخال (الحدث **<Return>**) .

* السطر 21 : إنشاء علبة القائمة (صنف الأصل **Listbox**) . و يعرب حجمه بعدد من الأحرف في السكر الحالي . و نقوم بطرح واحد أو إثنين, لتعويض تقريبا مكان الذي تشغله شريط التمرير (مجموعة من إثنين لديها ما يقرب نفس عرض حقل الإدخال) .

* السطر 22 : إنشاء شريط التمرير العمودي (صنف الأساس **Scrollbar**) . الأمر الذي سيرتبط به هو **command =self.bListe.yview** يشير لأسلوب الويدجت **Listbox** الذي سيتم إستدعائه و الذي سيتسبب في إنتقال قائمة في داخل العلبة, عندما نقوم بتحريك شريط التمرير .

* السطر 23 : بشكل متناظر, يجب علينا إعادة تكوين علبة القائمة للإشارة إلى أسلوب الويدجت **Scrollbar** الذي تم إستدعائه, بحيث موقع شريط التمرير يعبر بشكل صحيح على موقع للعنصر الذي تم إختياره في القائمة . و ليس من الممكن الإشارة إلى هذا الأمر في سطر تعليمات إنشاء صندوق القوائم, في السطر 21, لأن في هذه اللحظة الويدجت **Scrollbar** لم يوجد بعد . لذا تم إبعاد المرجع .

- السطر 33 : هذا الأسلوب يتم إستدعائه في كل مرة يقوم فيها المستخدم بتحديد عنصر في القائمة . فهي تقوم بإستدعاء الأسلوب **curselection()** لويدجت **Listbox** الأساس . و هذا يقوم بإرجاع نفق لمؤشرات, لأن هذا مقصود من مطوري tkinter التي تمكن المستخدم تحديد العديد من الاعناصر في القائمة (بمساعدة الزر **<Ctrl>**) . و مع ذلك نحن نفترض أن وحدا كان لافتا, و بالتالي إسترداد فقط عنصر واحد من هذا النفق . في السطر 47, يمكننا إذا إستخراج عنصر المقاب من القائمة و إستخدامه, في كل تحديث لحقل الإدخال (لاسطر 42) و كذلك مرجع تمثيل الويدجت (في حالة التطبيق الصغير لدينا, سيكون دالة **changeCoul()**) .

* السطور من 44 إلى 46 : يتم إستدعاء نفس لأمر عندما يقوم المستخدم بتفعيل زر الإدخال بعد ترميز سلسلة نصية في حقل الإدخال . البرامتر **event**, لا يتم إستخدامه هنا, و هو يسمح بإستعادة العنصر أو العناصر المرتبطة .

* السطران 48 و 49 : و لقد قمنا أيضا بتضمين الأسلوب **()get**, بعجد الإتفاقية التالية للويدجات الأخرى, للسماح لإستعادة بحرية أخر عنصر تم تحديده .

الويدجت نص يرافقه شريط تمرير

سوف نقوم بالشروع بنفس طريقة المثال السابق, سوف تقوم بربط ويدجات القياسية ل tkinter بطرق متعددة . و بالتلي نحن نقدم أدناه ويدجت مركب الذي يمكن إستخدامه لرسم نظام معالجة نصوص بدائي . عنصره الرئيسي هو الويدجت النص القياسي, الذي يمكنه عرض نصوص منسقة, هذا معناه دمج مع النصوص مختف سمتا الستيل (مثل تغميق, و مائل ...), و كذلك الخطوط المختلفة, و اللون و حتى الصور . سوف نقوم ببساطة بربط مع الشريط التمرير العمودي لنظهر لكم, مرة أخر, تمثيل الذي تستطيع صنعه بين هذه المركبات .

الويدجت نص قادر على تفسير نظام "العلامات" المدرجة في أي مكان في النص . معهم, يمكنك وضع معايير, عمل وصلة, و جعل العناصر قابلة للعرض (نصوص أو صور), بطريقة يمكن إستخدامه لتشغيل مجموعة متنوعة من الأليات .

على سبيل المثال, في التطبيق الموصوف أدناه, عندما نفوم بالضغط على إسم "Jean de la Fontaine'' بمساعدة زر الأيمن للفأرة, سيتسبب تلقائيا في تمرير التلقائي للنص (تمرير), حتى يصبح وصف هذا الكاتب مرئيا في الويدجت (أنظر للسكريبت المقابل في الصفحة التالية) . و من المميزات الأخر الموجودة, إمكانية تحديد بمساعدة الفأرة أي جزء من النص المعروض لتعديله, لكن نحن هنا نقدم الأساسيات . يرجى الرجوع إلى كتاب أو مواقع متخصصة لمزيد من المعلومات .

إدارة النص المعروض

يمكنك الوصول إلى أي جزء من نص يدعم ويدجت النص مع إثنين من المفاهيم التكميلية, الفهارس و المؤشرات :

* تتم لاإشارة إلى كل حرف من النص المعروض بمؤشر, الذي يجب أن يكون سلسلة نصة تحتوي على قيمتين متصلة بنقطة (على سبيل المثال " 5.2") . هتان القيمتان على التوالي تشيران إلى رقم السطر و رقم العمود أين يوجد الحرف .

* يمكن ربط أي جزء من النص مع واحدة أو أكثر من العلامات, التي أنت حر في إختيار إسمها و خصائصها . و هذه تسمح لك بتعريف الخط و الألوان و لون الخلفية و العناصر المرتبطة و إلخ ...

> لفهم النص أدناه, يرجي أن تعرف أنه سيتم التعامل مع ملف يسمى *CorbRenard.txt* تم ترميزه بواسطة *latin-1* .

```
1#     from tkinter import *
2#
3#     class ScrolledText(Frame):
4#        """Widget composite, associant un widget Text et une barre de défilement"""
5#        def __init__(self, boss, baseFont ="Times", width =50, height =25):
6#           Frame.__init__(self, boss, bd =2, relief =SUNKEN)
7#           self.text =Text(self, font =baseFont, bg ='ivory', bd =1,
8#                     width =width, height =height)
9#           scroll =Scrollbar(self, bd =1, command =self.text.yview)
10#          self.text.configure(yscrollcommand =scroll.set)
11#          self.text.pack(side =LEFT, expand =YES, fill =BOTH, padx =2, pady =2)
12#          scroll.pack(side =RIGHT, expand =NO, fill =Y, padx =2, pady =2)
13#
14#       def importFichier(self, fichier, encodage ="Utf8"):
15#          "insertion d'un texte dans le widget, à partir d'un fichier"
```

```
16#         of =open(fichier, "r", encoding =encodage)
17#         lignes =of.readlines()
18#         of.close()
19#         for li in lignes:
20#            self.text.insert(END, li)
21#
22#   def chercheCible(event=None):
23#      "défilement du texte jusqu'à la balise <cible>, grâce à la méthode see()"
24#      index = st.text.tag_nextrange('cible', '0.0', END)
25#      st.text.see(index[0])
26#
27#   ### البرنامج الرئيسي : نافذة مع ملصق و 'ScrolledText' ###
28#   fen =Tk()
29#   lib =Label(fen, text ="Widget composite : Text + Scrollbar",
30#          font ="Times 14 bold italic", fg ="navy")
31#   lib.pack(padx =10, pady =4)
32#   st =ScrolledText(fen, baseFont="Helvetica 12 normal", width =40, height =10)
33#   st.pack(expand =YES, fill =BOTH, padx =8, pady =8)
34#
35#   #: تعريف العلامات, و ربط حدث <نقر بالزر الأيمن> 
36#   st.text.tag_configure("titre", foreground ="brown",
37#                font ="Helvetica 11 bold italic")
38#   st.text.tag_configure("lien", foreground ="blue",
39#                font ="Helvetica 11 bold")
40#   st.text.tag_configure("cible", foreground ="forest green",
41#                font ="Times 11 bold")
42#   st.text.tag_bind("lien", "<Button-3>", chercheCible)
43#
44#   titre ="""Le Corbeau et le Renard
45#   par Jean de la Fontaine, auteur français
46#   \n"""
47#   auteur ="""
48#   Jean de la Fontaine
49#   écrivain français (1621-1695)
50#   célèbre pour ses Contes en vers,
51#   et surtout ses Fables, publiées
52#   de 1668 à 1694."""
53#
54#   #تعبئة الويدجت نص (طريقتين)
55#   st.importFichier("CorbRenard.txt", encodage ="Latin1")
56#   st.text.insert("0.0", titre, "titre")
57#   st.text.insert(END, auteur, "cible")
58#   #إضافة صورة
59#   photo =PhotoImage(file= "penguin.gif")
60#   st.text.image_create("6.14", image =photo)
61#   #إضافة علامات إضافية
62#   st.text.tag_add("lien", "2.4", "2.23")
63#
64#   fen.mainloop()
```

التعليقات

* الأسطر من 3 إلى 6 : الويدجت المركب الذي نعرفه في هذا الصنف سيتم إشتقاقه مرة أخرى من صنف **Frame()** . المنشئ ينتزر بع برامترا المثيل على سبيل المثال (الخط المستخدم, الطور و

العرض), مع قيم الإفتراضية . هذه البرامترات ستكون بسيطة التمرير لويدجت النص "الداخلي" (الأسطر 7 و 8) . و يمكنك بالطبع إضافة العديد من الأشياء الأخرى, لتحديد مظهر لون خلفية المؤشر, و لون الخلفية أو الحروف, و ما إذا كان يجب أن تقطع الأسطر الطويلة أو لا ...إلخ . كما يمكنك إرسال برامترات مختلفة إلى شريط التمرير .

* الأسطر من 7 إلى 10 : كما شرحنا سابقا (لويدجت مكعب بوكس), فإنه يأخذ ثلاثة أسطر من التعليمات لإنشاءتبادل لبن الويدجدان شريط التمرير و النص . بعد تمثيل الويدجت النص في السطران 7 و 8, نقوم بصنع شريط تمرير في السطر 9, مع تحديد في التعليمة بتمثيل أسلوب الويدجت النص الذي سيصبح تحت سيطرة شريط التمرير . ثم نقوم بإعادة تمثيل الويدجت نص في السطر 10, للإشارة إلى أسلوب العودة لشريط التمرير و الذي سيتم إستدعائه للحفاظ على الإرتفاع الصحيح . إعتمادا على شريط تمرير النص الأصلي . و ليس من الممكن الإشارة لهذا المرجع عند إنشاء ويدجت نص في السطرين 7 و 8, لأن في تلك اللحظة لم يكن شريط التمرير موجودا بعد .

* السطران 11 و 12 : الخيار **expand** للأسلوب **pack()** لا يقبل إلا قيم **YES** أو **NO** . و هي تحدد ما إذا كان يجب أن يمتدد الويدجت عند تغيير النافذة . خيار المكمل **fill** يمكنه أن يأخذ القيم الثلاثة التالية : **X** و **Y** أو **BOTH** . فهو يشير إذا كان الإمتداد يتم تنفيذه أفقيا (المحور **X**) أو عموديا (المحور **Y**) أو في الإتجاهين (**BOTH**) . عندما تطور تطبيق, فمن المهم أن تفكر في إعادة تحجيم الصفحة, خاصة إذا كان التطبيق يعمل في أنظمة تشغيل مختلفة (ويندوز, لينكس, ماك) .

* الأسطر من 22 إلى 25 : هذه الدالة هي معالج الأحداث, الذي يتم إستدعائه كلما ضغط المستخدم الزر الأسمن على إسم الكاتب (يتم ربط الحدث مع العلامة المقابلة, في السطر 42) .

في السطر 24, قمنا بإستخدام الأسلوب **tag_nextrange()** لويدجت النص للعثور على مؤشر لجزء معين من النص المرتبط مع العلامة "الهدف" . يقتصر البحث عن هذه المؤشرات على نطاق المعرف في البرامتر الثاني و الثالث (في مثالنا, نبحث من بداية إلى نهاية النص) . الأسلوب **()** **tag_nextrange** تقوم بإرجاع نفق به مؤشرين (وهي أول و أخر حروف من جزء النص المرتبط بعلامة "الهدف") . في السطر 25, نحن نستخدم واحدة من هذه المؤشرات (الأول) لتفعيل الأسلوب **see()** . و هذا يؤدي إلى تحريك التلقائي للنص (تمرير), بحيث الحرف المقابل للمؤشر الممرر يكون مرئيا في الويدجت (عادة مع عدد من الأحرف التي تتبعها) .

* الأسطر من 27 إلى 33 : البناء الكلاسيكي للنافذة يحتوي على ويدجدان .

* السطور من 35 إلى 42 : هذه الأسطر تعرف ثلاثة علامات : **titre** و **lien** و **cible** و هي تربط بتنسيق النص . السطر 42 يحدد أيضا أن النص المرتبط بعلامة **lien** سيكون قابل للنقر (الزر رقم 3 في الفأرة هو الزر الأيمن), مع الإشارة لمعالج الأحداث المطابق .

* السطر 55 : يمكنك إدخال أي مزة في تعريف النصف, كما فعلنا هنا من خلال توفير أسلوب لإستيراد الملف النصي في الويدجت نفسه, مع البرامتر لفك التشفير . مع هذا الأسلوب, النص الذي تم إستدعائه سيتم إضافته في نهاية النص الذي تم وضعه في الويدجت, لكن يمكنك بسهولة أن تحسنها بإضافة برامتر جديد لتحديد الموقع الدقيق حيث يتم إدراجه .

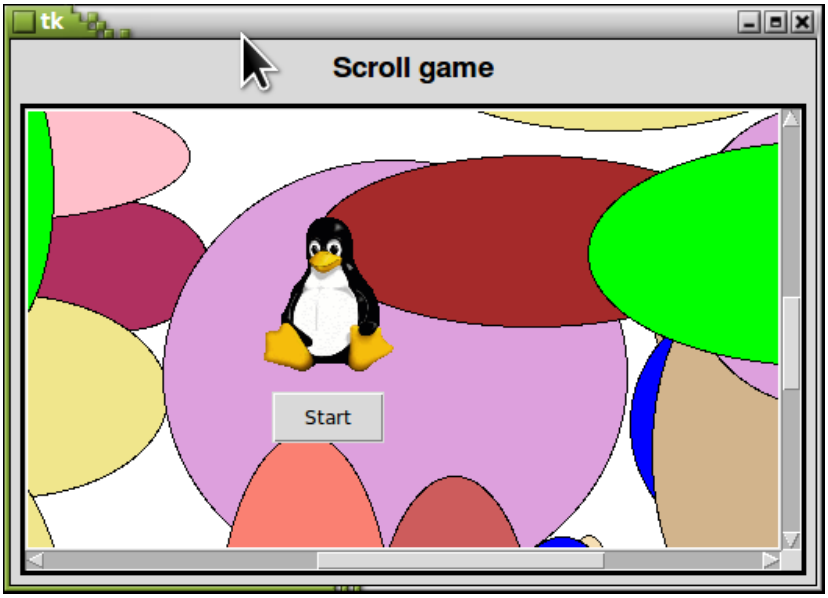
* السطران 56 و 57 : هذه التعليمات تقوم بإدراج أجزاء النص (في بداية و نهاية النص السابق), بربط علامة مع كل واحدة منهم .

* السطر 62 : ربط العلامات للنص ديناميكي . في أي وقت, يمكنك تفعيل ربط جدبد (كما فعلنا هنا من خلال ربك العلامة "**lien** " لجزء الحالي من النص) . ل"فصل" علامة, إستخدم الأسلوب **()** **tag_delete** .

لوحات مع أشرطة تمرير

لدينا العديد من إستغلالات الويدجت اللوحة, و الإمكانيات واسعة جدا . لقد رأينا كيفية إثراء هذا الصنف بالإشتقاق . و هذا ما سوف نفعله مرة أخرى في المثال أدناه, مع تعريف صنف جديد **ScrolledCanvas**, الذي ربطنا مع لوحة القياسية شريطي تمرير (أفقي و عمودي) .

لجعل التمرين أكثر جاذبية, سوف نستخدم صنف ويدجت جديد لجعل لعبة العنوان, و التي يجب على المستخدم أن ينجح بالضغط على زر الذي يتفاداه بإستمرار .



الويدجت لوحة متعدد جدا : يسمح بالجمع بين الرسومات و الصور النقطية (bmp), و أجزاء النص و العديد من الويدجات الأخرى, في مساحة ممتدة . إذا كنت ترغب في تطوير لعبة رسومية, فهذه القطعة الأولى التي يجب أن تتقنها .

تطبيقنا الصغير بقدم على أنه صنف جديد **FenPrinc()**, الذي تم الحصول عليه من خلال إشتقاق من صنف الأصل **Tk()** . و هي تحتوي على ويدجادان / ملصق بسيط و ويدجتنا المركب الجديد **ScrolledCanvas** . و هذا هو نظرة للوحة رسم كبيرة جدا, و يمكننا فيها "السفر" من خلال شريط التمرير .

المساحة المتاحة مرصودة جيدا, بدأ بزرع ديكود بسيط, مصنوع من 80 دائرة بيضوية من لون و الموقع و الحجم عشوائي . و لقد قمنا بإضافة غمزة عينة صغير في شطل صور نفطية, التي تهدف أن تذكركم كيفية التعامل مع هذا النوع من الموارد .

و أخيرا, لقد قمنا بتثبيت قطعة وظيفية حقيقية : و هي زر بسيط, لكن تقنية التنفيذ يمكن أن تطبف على أي نوع أخر من الويدجات, بما في ذلك ويدجات مركبة كبيرة مثل التي برمجناها سابقا . هذه المرونة في تطوير التطبيقات المعقدة هي واحدة من الفوائد الرئيسية من البرمجة الشيئية .

الزر يتحرك بأسرع من المرة الأولى, والرسوم المتحركة تتوقف إذا أتينا للضغط على زر جديد . في تحليل النص بالأسفل, إبلي إهتمام بالأساليب المستخدمة لتعديل خصائص كائن موجود .

```
1#     from tkinter import *
2#     from random import randrange
3#
4#     class ScrolledCanvas(Frame):
5#        """Canevas extensible avec barres de défilement"""
6#        def __init__(self, boss, width =100, height =100, bg="white", bd=2,
7#                 scrollregion =(0, 0, 300, 300), relief=SUNKEN):
8#           Frame.__init__(self, boss, bd =bd, relief=relief)
9#           self.can =Canvas(self, width=width-20, height=height-20, bg=bg,
10#                    scrollregion =scrollregion, bd =1)
11#           self.can.grid(row =0, column =0)
12#           bdv =Scrollbar(self, orient =VERTICAL, command =self.can.yview, bd =1)
13#           bdh =Scrollbar(self, orient =HORIZONTAL, command =self.can.xview, bd
      =1)
14#           self.can.configure(xscrollcommand =bdh.set, yscrollcommand =bdv.set)
15#           bdv.grid(row =0, column =1, sticky = NS)      #sticky =>
16#           bdh.grid(row =1, column =0, sticky = EW)      #رسك الشريط
17#           #ربط حدث <إعادة التحجيم> إلى المعالج المناسب
18#           self.bind("<Configure>", self.redim)
19#           self.started =False
20#
21#        def redim(self, event):
22#           "opérations à effectuer à chaque redimensionnement du widget"
23#           if not self.started:
24#              self.started =True      #لا تغير الحجم
25#              return                  #عند إنشاء الويدجت (و إلا => الحلقة)
26#           #من حج الإطار الجديد, غير حجم اللوحة
27#           #فرق 20 بيكسل للتعويض عن سمك أشرطة التمرير) :
28#           larg, haut = self.winfo_width()-20, self.winfo_height()-20
29#           self.can.config(width =larg, height =haut)
30#
31#     class FenPrinc(Tk):
32#        def __init__(self):
33#           Tk.__init__(self)
34#           self.libelle =Label(text ="Scroll game", font="Helvetica 14 bold")
35#           self.libelle.pack(pady =3)
36#           terrainJeu =ScrolledCanvas(self, width =500, height =300, relief=SOLID,
37#                        scrollregion =(-600,-600,600,600), bd =3)
38#           terrainJeu.pack(expand =YES, fill =BOTH, padx =6, pady =6)
39#           self.can =terrainJeu.can
40#           #ديكور : سلسلة من أشكار البيضاوية العشوائية
41#           coul =('sienna','maroon','brown','pink','tan','wheat','gold','orange',
42#                'plum','red','khaki','indian red','thistle','firebrick',
43#                'salmon','coral','yellow','cyan','blue','green')
44#           for r in range(80):
45#              x1, y1 = randrange(-600,450), randrange(-600,450)
46#              x2, y2 = x1 +randrange(40,300), y1 +randrange(40,300)
47#              couleur = coul[randrange(20)]
48#              self.can.create_oval(x1, y1, x2, y2, fill=couleur, outline='black')
49#           #إضافة صورة GIF صغيرة
50#           self.img = PhotoImage(file ='linux2.gif')
51#           self.can.create_image(50, 20, image =self.img)
52#           #زر للقبض عليه :
53#           self.x, self.y = 50, 100
54#           self.bou = Button(self.can, text ="Start", command =self.start)
55#           self.fb = self.can.create_window(self.x, self.y, window =self.bou)
56#
57#        def anim(self):
```

```
58#         if self.run ==0:
59#            return
60#         self.x += randrange(-60, 61)
61#         self.y += randrange(-60, 61)
62#         self.can.coords(self.fb, self.x, self.y)
63#         self.libelle.config(text = 'Cherchez en %s %s' % (self.x, self.y))
64#         self.after(250, self.anim)
65#
66#      def stop(self):
67#         self.run =0
68#         self.bou.configure(text ="Start", command =self.start)
69#
70#      def start(self):
71#         self.bou.configure(text ="Attrapez-moi !", command =self.stop)
72#         self.run =1
73#         self.anim()
74#
75#   if __name__ =="__main__":          #--- برنامج التجربة ---
76#      FenPrinc().mainloop()
```

تعليقات

* السطور من 6 إلى 10 : مثل الكثيري الأخرين, ويدجتنا مشتق من **Frame()** . منشئه يقبل عدد من البرامترات . لاحظ أن هذه البرامترات تمرر إلى جزء الإطار (البرامتر **bd** و **relief**), و لجزء اللوحة (البرامترات **width** و **height** و **bg** و **scrollregion**) . يمكنك إختيار إختيارات أخرى, الخيار **scrollregion** لويدجت اللوحة يستخدم لتحديد مساحة الرسم في عرض اللوحة التي يمكن أن تتحرك .

* الأسطر من 11 إلى 16 : نحن إستخدمنا هذه المرة الأسلوب **grid()** لوضع اللوحة و شراطي التمرير في أماكنهم (هذا الأسلوب قدم لك سابقا في صفحة 95) . الأسلوب **pack()** غير مناسب لوضع شريطي التمرير في أماكنهم, لأنها تتطلب إستخدام العديد من الإطارات متداخل (حاول, إعتبره تمرينا !) . التفاعلات بين شريط التمرير و الويدجت الذي يسيطر عليها (الأسطر 12 و 13 و 14) تم وصفهم بالتفصيل للويدجتان المركبان السابقان . الخيار **orient** لشريطي التمرين لم تستخدم حتى الأن, لأن قيمتها الإفتراضية (**VERTICAL**) تسبب حالات معالجات .

في الأسطر 15 و 16 الخيارات **sticky =NS** و **sticky =EW** ستسبب إمتداد شريطي العنوان إلى فوق كله (**NS** = معناها إتجاه شمال-جنوب) أو العرض (**EW** = معناه الإتجاه شرق-غرب) خلية في الشبكة . سيكون هنالك إعادة تحجيم تلقائية, كما هو الحال مع الأسلوب **pack()** (الخيارات **expand** و **fill** غير متوفرين) .

* السطر 18 : منذ أن الأسلوب **grid()** لا يتضمن الإعادة التحجيم التلقائية, يجب علينا أن نراقب الحدث الذي يتم إنشاؤه من قبل نظام التشغيل عندما يقوم المستخدم بتغيير حجم الويدجت, و ربطها مع أسلوب المناسب لجعلنا نعيد تحجيم مركبات الويدجت .

* الأسطر من 19 إلى 29 : أسلوب إعادة التحجيم سيقوم بإعادة تغيبر حجم اللوحة (أشرطة التمرير تتكيف بنفسها, بسبب الخيار **sticky** المطبق) . لاحظ أنه يمكنك إيجاد أبعاد الويدجات في سماته **winfo_width()** و **winfo_height()** .

متغير المثيل **self.started** هو مبدل بسيط, الذي يسمح بمنع ما يسمى إعادة التحجيم قبل الأوان, عند تمثيل ويدجت (هذا ينتج حلقة غريبة : حاول بدونه) .

* الأسطر من 31 إلى 55 : هذا الينف يعرف لعبتنا الصغيرة . منشئها يمثل ويدجتنا الجديد في متغير **terrainJeu** (السطر 36) . لاحظ أن نوع و سمك الحدود تطبق على إطار الويدجت المركب, في حين أن البرامترات الأخرى تختار أنها تطبق على لوحة . مع الخيار **scrollregion**, عرفنا مساحة اللعبة أكبر بكثير من مساحة سطح اللوحة نفسها, مما يضطر اللاعب لإنتقال (أو تغيير حجمها) .

* السطران 54 و 55 : الأسلوب **create_window()** لويدجت اللوحة هو الذي يسمح بإدخال أي ويدجت أخر (بما في ذلك الويدجات المركبة) . الويدجت الذي يتم إدخاله يجب أن يكون معرف على أنه عبيد اللوحة أو النافذة الرئيسية . الأسلوب **create_window()** ينتظر ثلاثة برامترات : الإحداثيات **X** و **Y** النقطة التي تريد إدراج الويدجت و مرجع هذا الويدجت .

* السطور من 57 إلى 64 : هذا الأسلوب يستخدم لجعل الزر رسم متخرك . بعد تغيير موقع زر تلقائيا عل مسافة معينة من الموقع السابق, و سوف تعيد هذا بعد توقف مؤقت لمدة 250 ميلي ثاني . هذا الإغلاق يحدث بإستمرار, مادام المتغير **self.run** يحتوي على قيمة ليست لا شيئ .

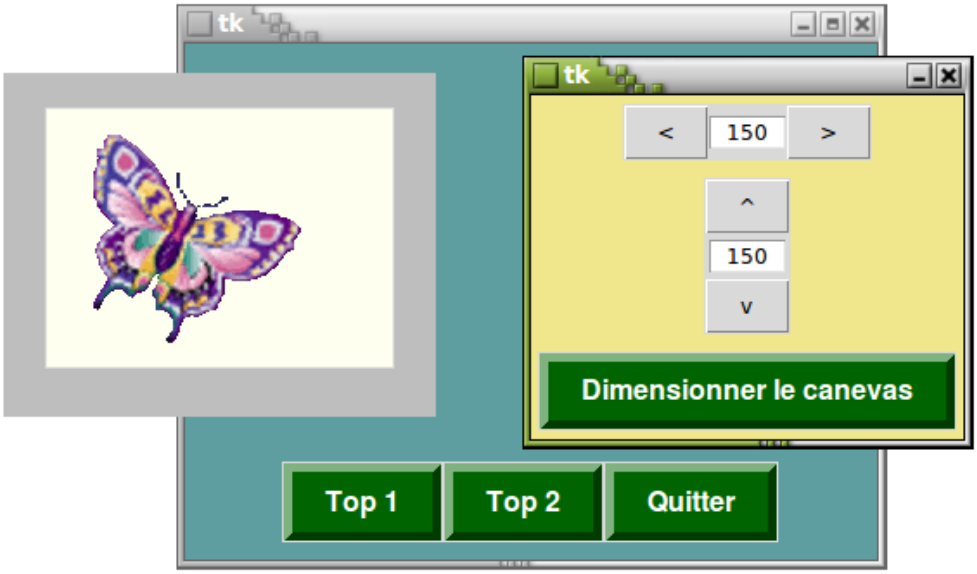
* الأسطر من 66 إلى 73 : هذان معالجان الأحداث مرتبطان بزر بالتناوب . فمن الواضح أنه لبدأ و إيقاف الرسوم المتحركة .

تطبيق نوافذ متعددة - تعيين ضمني

الصنف **Toplevel()** ل tkinter يسمح بصنع نوافذ "أقمار إصناعية" لتطبيقك الأصل . هذه النوافذ مستقلة, لكنها تغلق تلقائيا عند إغلاق النافذة الرئيسية . ضع هذا القيد على الجانب, و يتم التعامل معها بالطريقة المعتادة : يمكن وضع أي مجموعة من الويدجات .

التطبق الصغير أدناه يوضح لك بعض قدراتها . و هي تتألف من نافذة رئيسية عادية جدا, تحتوي على 3 أزرار . هذه الأزرار صنعت بصنغ مشتقاق من صنف **Button()** أساسا, و ذلك ليظهر لك مرة أخرى كم هو من السهل تكيف أنصاف الكائن الموجود لأجلك . سوف تلاحظ بعض خيارات "الديكور" المثيرة للإهتمام .

الزر **<Top1>** يظهر لك أول نافذة قمر صناعي تحتوي على لوحة مع سورة . لدينا مع هذه النافذة خصائصها الخاصة : و هي ليس لديها لا شعار-عنوان و لا حدود, و مستحيل إعادة تحجيمها بواسطة الفأرة . بالإضافة إلى ذلك, هذه النافذة مشروطة : الأمر يستحق كل نافذة لا تزال في المقدمة, أمام جميع النوافذ الأخرى لتطبيقات أخرى قد تكون موجودة على الشاشة .



الزر **<Top2>** يعرض نافذة قمر صناعي أكثر كلاسيكية, التي تحتوي على نسختين من ويدجت مركب صغير **SpinBox** الذي صنعناه وفقا للمبادئ المذكورة في الصفحات السابقة . هذا الويدجت

يتكون من زران و ملصق يشير إلى قيمة رقمية . الأزرار تسمح بزيادة أو تقليل القيمة المعروضة . بالإضافة إلى هذان **SpinBoxes**, النافذة تحتوي على زر كبير مزين . عند تشغيله, المستخدم يسبب بتغيير حجم اللوحة في نافذة قمر صنعي أخرى, بالإتفاق مع القيم الرقمية المعروضة في 2 **SpinBoxes** .

```
1#    from tkinter import *
2#
3#    class FunnyButton(Button):
4#      "Bouton de fantaisie : vert virant au rouge quand on l'actionne"
5#      def __init__(self, boss, **Arguments):
6#        Button.__init__(self, boss,  bg ="dark green", fg ="white", bd =5,
7#                  activebackground ="red", activeforeground ="yellow",
8#                  font =('Helvetica', 12, 'bold'), **Arguments)
9#
10#   class SpinBox(Frame):
11#     "widget composite comportant un Label et 2 boutons 'up' & 'down'"
12#     def __init__(self, boss, largC=5, largB =2, vList=[0], liInd=0, orient =Y):
13#       Frame.__init__(self, boss)
14#       self.vList =vList        #قائمة القيم المتوفرة
15#       self.liInd =liInd        #مؤشر القيمة العرض الإفتراضي
16#       if orient ==Y:
17#         s, augm, dimi = TOP, "^", "v"     #التوجيه " العمودس"
18#       else:
19#         s, augm, dimi = RIGHT, ">", "<"    #التوجيه "الأفقي"
20#       Button(self, text =augm, width =largB, command =self.up).pack(side =s)
21#       self.champ = Label(self, bg ='white', width =largC,
22#                  text =str(vList[liInd]), relief =SUNKEN)
23#       self.champ.pack(pady =3, side =s)
24#       Button(self, text=dimi, width=largB, command =self.down).pack(side =s)
25#
26#     def up(self):
27#       if self.liInd < len(self.vList) -1:
28#         self.liInd += 1
29#       else:
30#         self.bell()     #صوت تنبيه
31#       self.champ.configure(text =str(self.vList[self.liInd]))
32#
33#     def down(self):
34#       if self.liInd > 0:
35#         self.liInd -= 1
36#       else:
37#         self.bell()     #صوت تنبيه
38#       self.champ.configure(text =str(self.vList[self.liInd]))
39#
40#     def get(self):
41#       return self.vList[self.liInd]
42#
43#   class FenDessin(Toplevel):
44#     "Fenêtre satellite (modale) contenant un simple canevas"
45#     def __init__(self, **Arguments):
46#       Toplevel.__init__(self, **Arguments)
47#       self.geometry("250x200+100+240")
48#       self.overrideredirect(1)        #=> نافذة دون حدود
49#       self.transient(self.master)     #=> نافذة 'modale'
50#       self.can =Canvas(self, bg="ivory", width =200, height =150)
51#       self.img = PhotoImage(file ="papillon2.gif")
```

```
52#         self.can.create_image(90, 80, image =self.img)
53#         self.can.pack(padx =20, pady =20)
54#
55#   class FenControle(Toplevel):
56#     "Fenêtre satellite contenant des contrôles de redimensionnement"
57#     def __init__(self, boss, **Arguments):
58#       Toplevel.__init__(self, boss, **Arguments)
59#       self.geometry("250x200+400+230")
60#       self.resizable(width =0, height =0)   #=> منع تغيير الحجم
61#       p =(10, 30, 60, 90, 120, 150, 180, 210, 240, 270, 300)
62#       self.spX =SpinBox(self, largC=5,largB =1,vList =p,liInd=5,orient =X)
63#       self.spX.pack(pady =5)
64#       self.spY =SpinBox(self, largC=5,largB =1,vList =p,liInd=5,orient =Y)
65#       self.spY.pack(pady =5)
66#       FunnyButton(self, text ="Dimensionner le canevas",
67#             command =boss.redimF1).pack(pady =5)
68#
69#   class Demo(Frame):
70#     "Démo. de quelques caractéristiques du widget Toplevel"
71#     def __init__(self):
72#       Frame.__init__(self)
73#       self.master.geometry("400x300+200+200")
74#       self.master.config(bg ="cadet blue")
75#       FunnyButton(self, text ="Top 1", command =self.top1).pack(side =LEFT)
76#       FunnyButton(self, text ="Top 2", command =self.top2).pack(side =LEFT)
77#       FunnyButton(self, text ="Quitter", command =self.quit).pack()
78#       self.pack(side =BOTTOM, padx =10, pady =10)
79#
80#     def top1(self):
81#       self.fen1 =FenDessin(bg ="grey")
82#
83#     def top2(self):
84#       self.fen2 =FenControle(self, bg ="khaki")
85#
86#     def redimF1(self):
87#       dimX, dimY = self.fen2.spX.get(), self.fen2.spY.get()
88#       self.fen1.can.config(width =dimX, height =dimY)
89#
90#   if __name__ =="__main__":          #--- برنامج التجربة ---
91#     Demo().mainloop()
```

تعليقات

* الأسطر من 3 إلى 8 : إذا كنت ترغب في الحصول على نفس تصميم الأزرار المختلفة في الأجزاء المختلفة من مشروعك, لا تتردد في صنع فئة مشتقة, كما فعلنا هنا . و هذا سيوفر لك الحاجة لإعادة برمجة نفس الخيارات المحددة .

لاحظ النجمتين التي بدأنا بها إسم نهاية برامتر المنشئ : **Arguments** (**برامترات) . و هذا يعني أن هذا المتغير هو قاموس, قادر على تلقي تلقائيا أي مجموعة من البرامترات مع الملصق . هذه البرامترات يمكنها تمرير نفسها للمنشئ الصنف الأصل (في السطر 8) . و هذا يمنعنا من الإضطرار إلى

إعادةكتابة في قائمة البرامترات جميع الخيارات البرامترية للزر الأساس, و التي هي كثيرة جدا . و أيضا يمكنك تمثيل هذه الأزرار مع أي مجموعة من الخيارات, طالما أنه متاحة للأزرار الأساس . و نسمي هذا بالبرامترات الضمنية . يمكنك إستخدام هذا الشكل من البرامترات مع أي دالة أو أسلوب .

* الأسطر من 10 إلى 24 : منشئ الويدجت الخاص بنا **SpinBox** لا يحتاج إلا القليل من التعليقات . إعتمادا على التوجيه المطلوب, الأسلوب **pack()** يتيح أزرار و ملصقات من الأعلى للأسفل أو اليمين لليسار (البرامترات **TOP** أو **RIGHT** للخيار **side**) .

* الأسطر من 26 إلى 38 : هذان الأسلوبان لا تفعل شيئا أكثر من تعديل قيمة المعروضة في الملصق . لاحظ أن الصنف **Frame()** لديه أسلوب **bell()** ليتسبب في صوت "بيب" .

* الأسطر من 43 إلى 53 : تعريف نافذة القمر الصناعي الأولى هنا . لاحظ مرة أخرى إستخدام البرامتر الضمني للمنشئ, بمساعدة المتغير **Arguments**** . هذا هو الذي يسمح لنا بتمثل هذه النافذة (في السطر 81) عن طريق تحديد لون الخلفية (يمكن أن يطلب أيضا الحدود, إلخ ...) . أساليب الإستدعاء في الأسطر من 47 إلى 49 تم تعريف بعض خصائصه (ينطبق على أي نافذة) . الأسلوب **geometry()** يسمح لك بتعيين إحداثيات النافذة و مكانه في النافذة (+100+240 تعني أنه ينبغي نحول الزاوية العلوية اليسرى 100 بيكسل إلى اليمين و 240 بيكسل أسفل زاوية اليسرى العلوية من الشاشة) .

* الأسطر 45 و 57 : يرجى ملاحظة الفرق الصغير بين قوائم البرامترات لهذه الأسطر . في منشئ **FenDessin**, تم حذف البرامتر **boss**, الذي هو موجود في منشئ **FenControle** . هل تعلم أن هذا البرامتر يستخدم لتمرير مرجع الويدجت "السيد" ل "عبيده" . و هو عموما ضروري (في السطر 67, على سبيل المثال, فإننا إستخدما للمرجه أسلوب البرنامج الرئيسي), لكن لم يكن ضروريا للغاية : في **FenDessin** ليس لدينا به أي فائدة . سوف تجد الفرق واضح في تعليمات التمثيل لهذه النوافذ, في الأسطر 82 و 84 .

* الأسطر من 55 إلى 67 : بإستثناء الفرق المذكور أعلاه, منشئ الويدجت **FenControle** مشابه جدا ل **FenDessin** . لاحظ في السطر 60 الأسلوب الذي يسمح بمنع تغيير حجم النافذة (في حالة النافذة بدون حدود و بدون إعلان-العنوان, مثل **FenDessin**, سيكون هذا الأسلوب بدون جدوى) .

* الأسطر 73 و 74 (و 49) : جميع الأصناف المشتقة هي ويدجات tkinter ميزة تلقائيا بسمة **master**, الذي يحتوي على مرجع الصنف الأصل . الذي يسمح لنا بالوصول إلى أبعاد و لون خلفية من النافذة الرئيسية .

* الأسطر من 86 إلى 88 : هذا الأسلوب يسترد قيم الرقمية المعروضة في نافذة التحكم (الأسلوب **get()** لهذا الويدجت), لإعادة تحجيم لوحة نافذة الرسم . هذا المثال البسيط يبين لك, مرة أخرى, كيفية تحقيق تواصل بين مختلف مكونات البرنامج .

أشرطة الأدوات - تعبير لامدا (lambda)

العديد من البرامج لديها شريط أدوات (تولبار - **toolbar**) أو أكثر التي تتكون من أزرار صغيرة التي تمثل أيقونات . هذا النهج يتيح لك أن تقدم للمستخدم عدد كبير من الأوامر الخاصة, و ليس لأنها تحتل مساحة على الشاشة أيضا .

البرنامج الموضح أدناه لديه شريط أدوات و لوحة . عندما يضغط المستخدم على واحدة من 8 أزرار في الشريط, يتم نسخ الأيقونة على اللوحة, على موقع مختار عشوائيا . عندما يتم النقر على الزر الأخير, يتم حذف محتوى اللوحة .

```
1#    from tkinter import *
2#    from random import randrange
3#
4#    class ToolBar(Frame):
```

```
5#      "Barre d'outils (petits boutons avec icônes)"
6#      def __init__(self, boss, images =[], command =None, **Arguments):
7#         Frame.__init__(self, boss, bd =1, **Arguments)
8#         # <images> = قائمة أسماء الأيقونات لوضعها على الأزرار
9#         self.command =command          #أمر لتنفيذه عند النقر
10#        nBou =len(images)          #عدد الأزرار لبنائها
11#        #يجب وضع الأيقونات في متغيرات ثابة
12#        #قائمة تقوم بهذا :
13#        self.photoI =[None]*nBou
14#        for b in range(nBou):
15#           #إنشاء أيقونة (objet PhotoImage Tkinter) :
16#           self.photoI[b] =PhotoImage(file = images[b] +'.gif')
17#           #إنشاء زر . نقوم بإستدعاء دالة lambda
18#           #لتمرير برامتر إلى الأسلوب <action> :
19#           bou = Button(self, image =self.photoI[b], bd =2, relief =GROOVE,
20#                  command = lambda arg =b: self.action(arg))
21#           bou.pack(side =LEFT)
22#
23#     def action(self, index):
24#        #تنفيذ <command> مع مؤشر الزر كبرامتر
25#        self.command(index)
26#
27#  class Application(Frame):
28#     def __init__(self):
29#        Frame.__init__(self)
30#        #أسماء الملفات التي تحتوي على الأيقونات (نوع GIF):
31#        icones =('floppy_2', 'coleo', 'papi2', 'pion_1', 'pion_2', 'pion_3',
32#              'pion_4', 'help_4', 'clear')
33#        #إنشاء شريط أدوات
34#        self.barOut =ToolBar(self, images =icones, command =self.transfert)
35#        self.barOut.pack(expand =YES, fill =X)
36#        #إنشاء لوحة لإستقبال الصور
37#        self.ca = Canvas(self, width =400, height =200, bg ='orange')
38#        self.ca.pack()
39#        self.pack()
40#
41#     def transfert(self, b):
42#        if b ==8:
43#           self.ca.delete(ALL)      #مسح كل شيئ في اللوحة
44#        else:
45#           #قم بنسخ أيقونة الزر b (المستخرج من الشريط) => اللوحة) :
46#           x, y = randrange(25,375), randrange(25,175)
47#           self.ca.create_image(x, y, image =self.barOut.photoI[b])
48#
49#  Application().mainloop()
```

ميتابروغراميك - التعبير لامدا

أنت تعلم بصفة عامة, يرتبط مع كل زر أمر, و هو مرجع لأسلوب أو دالة معينة التي تهتم بالعمل عند تنشيط الزر . و مع ذلك, في هذا تطبيق يعرض, جميع أزرار التي تقريبا تقوم بنفس الشيئ (نسخ الرسم على لوحة), الفرق الوحيد بينها هو رسم concerné .

لتبسيط كودنا, نحن نريد ربط الخيار **command** لكل الأزرار مع واحد و نفس الأسلوب (سيكون الأسلوب **action()**), لكن في كل مرة نقوم فيها بتمرير المرجع للزر المستخدم بشكل خاص, بحيث يكون نشاطه مختلف عن كل واحد فيهم .

سوف تنشأ صعوبة, لأن الخيار **command** للويدجت الزر يقبل فقط قيمة أو تعبير, و ليس تعليمة . بل هو في الواقع يشير إلى مرجع الدالة, لكن ليس لإستدعاء دالة مع هذه البرامترات (الإستدعاء سيتم في الواقع عندما يقوم المستخدم بالضغط على الزر : و هذا هو متلقي أحداث ل tkinter) . هذا هو سبب وجود فقك إسم الدالة, بدون أقواس .

يمكننا حل هذه المشكلة بطريقتين :

* و نظر لطبيعته الديناميكية, البايثون يقبل البرنامج الذي يمكنه تغيير نفسه, على سبيل المثال من خلال تعريف وظائف جديد أثناء التنفيذ (و هذا هو مفهو الميتابروغراميك) .

و لذلك من الممكن تعرف دالة مع برامترات, يشير لك واحد منها قيمة إفتراضية, قم توفير مرجع لهذه الدالة للخيار **command** . منذ أن يتم تعريف دالة أثناء التشغيل, هذه القيم الإفتراضية يمكن أن تكون ضمن محتويات المتغير . عندما يمكن للحدث "الضغط على زر" إستدعاء دالة, لذلك سوف تستخدم القيم الإفتراضية لبرامتراته, كما لو كانت برامترات . الناتج من العملية هو بالتالي تمرير برامترات كلاسيكية .

لتوضيح هذه التقنية إستبدل السطر من 17 إلى 20 من السكريبت بهذا :

```
# إنشاء زر بتعريف دالة لحظيا مع برامتر ذا قيمة إفترضية و هي البرامتر الممرر . هذه الدالة
تقوم بإستدعاء الأسلوب الذي يتطلب برامتر
def agir(arg = b):
self.action(arg)from tkinter import *
50#   from random import randrange
51#
52#   class ToolBar(Frame):
53#      "Barre d'outils (petits boutons avec icônes)"
54#      def __init__(self, boss, images =[], command =None, **Arguments):
55#         Frame.__init__(self, boss, bd =1, **Arguments)
56#         # <images> = قائمة أسماء الأيقونات لوضعها على الأزرار
57#         self.command =command          # أمر لتنفيذه عند النقر
58#         nBou =len(images)          # عدد الأزرار لبنائها
59#         # يجب وضع الأيقونات في متغيرات ثابة
60#         # قائمة تقوم بهذا :
61#         self.photoI =[None]*nBou
62#         for b in range(nBou):
63#            # إنشاء أيقونة (objet PhotoImage Tkinter) :
```

```
64#         self.photoI[b] =PhotoImage(file = images[b] +'.gif')
65#         #إنشاء زر . نقوم بإستدعاء دالة lambda
66#         #لتمرير برامتر إلى الأسلوب <action> :
67#         bou = Button(self, image =self.photoI[b], bd =2, relief =GROOVE,
68#                 command = lambda arg =b: self.action(arg))
69#         bou.pack(side =LEFT)
70#
71#     def action(self, index):
72#         #تنفيذ <command> كبرامتر مع مؤشر الزر
73#         self.command(index)
74#
75#   class Application(Frame):
76#     def __init__(self):
77#         Frame.__init__(self)
78#         #أسماء الملفات التي تحتوي على الأيقونات (نوع GIF):
79#         icones =('floppy_2', 'coleo', 'papi2', 'pion_1', 'pion_2', 'pion_3',
80#               'pion_4', 'help_4', 'clear')
81#         #إنشاء شريط أدوات
82#         self.barOut =ToolBar(self, images =icones, command =self.transfert)
83#         self.barOut.pack(expand =YES, fill =X)
84#         #إنشاء لوحة لإستقبال الصور
85#         self.ca = Canvas(self, width =400, height =200, bg ='orange')
86#         self.ca.pack()
87#         self.pack()
88#
89#     def transfert(self, b):
90#         if b ==8:
91#             self.ca.delete(ALL)     #مسح كل شيئ في اللوحة
92#         else:
93#             #قم بنسخ أيقونة الزر b (المستخرج من الشريط) => اللوحة :
94#             x, y = randrange(25,375), randrange(25,175)
95#             self.ca.create_image(x, y, image =self.barOut.photoI[b])
96#
Application().mainloop()

# الأمر يرتبط مع زر يستدعي الدالة أدناه

bou = Button(self, image = self.photoI[b], relief = GROOVE,

command = agir)
```

* كل الذي سبق يمكن تبسيطه, بإستخدام تعبير لامدا . هذه الكلمة المحجوز في البايثون هي تعبير الذي يقوم بإرجاع كائن دالة, مماثلة لتلك التي تقوم بإنشاء مع التعليمة **def**, لكن مع إهتلاف ان لامدا هي تعبير و ليس تعليمة, يمكننا إستخدام كواجهة لإستدعاء دالة (مع تمرير برامترات)والذي عاديا غير ممكن . لاحظ أن مثل هذه الدالة مجهولة (أي ليس لها إسم) .

على سبيل المثال, التعبير :

```
lambda ar1=b, ar2=c : bidule(ar1,ar2)
```

يقوم بإرجاع مرجع دالة مجهولة تم صنعها, و التي تستطيع إستدعاء الدالة **bidule()** و تمرير برامترا **b** و **c**, و التي تستخدم الأخيرة قيم إفتراضية في تعريف البرامترات **ar1** و **ar2** للدالة .

هذه التقنية تستخدم نفس المبدأ السابق, لكن لديها ميزة كونها أكثر إيجازا, و الذي هو سبب في إننا قد إستخدمناها في سطريبتنا . و مع ذلك, فإنه من الصعب فهمها :

```
command = lambda arg =b: self.action(arg)
```

في هذا الجزء من التعليمات, الأمر يرتبط مع الزر الذي هو مرجع للدالة مجهولة مع تمرير للبرامتر **arg** قيمة أولية : قيمة البرامتر هي **b** .

يتم إستدعائها بدون برامتر من الأمر, هذه الدالة المجهولة يمكنها القيام بنفس إستخدام البرامتر **arg** (مع قيمة إفتراضية) بإستدعاء للأسلوب الهدف **self.action()**, و نحصل على نقل برامتر حقيقي لهذا الأسلوب .

نحن لا نريد تفاصيل هنا السؤال التعبير لامدا, لأنه خارج الإطار الذي وضعناه للتهيئة . إذا أردت المزيد, يمكنك سؤال أحدهم أو البحث في مراجع اللغة .

تمرير دالة (أو أسلوب) كبرامتر

لقد تحدثنا بالفعل على العديد من الويدجات المركبة و الكثير من خياراتها مثل الخيار **command** الذي يجب أن يرتبط مع إسم الدالة أو الأسلوب . بعبارات أعم, هذا معناه أن الدالة مع البرامتر يمكنها تلقي مرجع دالة أخرى كبرامتر, و الفائدة الشيئ واضح هنا .

و لقد قمنا ببرمجة وظيفة مميزة مثل هذه في صنفنا الجديد **ToolBar()** . و يمكنك أن ترى أننا قمنا أدرجنا إسم **command** في قائمة البرامترات لمنشئه, في السطر 6 . هذا البرامتر ينتظر مرجع دالة أو أسلوب كبرامتر . ثم يتم تخزين المرجع في متغير مثيل (في السطر 9), بحيث يكون في متناول بقية أساليب النصف . و يمكن لهذا أن تستدعي دالة أو أسلوب (إذا لزم الأمر عن طريق تمرير برامترات, بعد شرح التقنية في الجزء السابق) . و هذا ما يفعله أسلوبنا **action()**, في السطر 25 . في هذه الحال, الأسلوب التمرير هو أسلوب **transfert()** للصنف **Application** (أنظر للسطر 34) .

النتيجة لهذا هو أننا قادرين على نطوير صنف كائنات **ToolBar** قابلة لإعادة الإستخدام في سياقات أخرى . كما ترى تطبيقنا الصغير, يكفي إنشاء مثبل لهذه الكائنات التي تشير إلى مرجع أي دالة ببرامتر الخيار **command** . هذه الدالة سيتم إستدعائها تلقائيا مع رقم ترتيبها للزر عندما يضغطه المستخدم .

لا تتردد في تخيل ماذا تفعله هذه الدالة !

للإنتهاء من هذا المثال, لاحظ تفاصيل أخرى أيضا : يحيط بكل واحد من أزرارنا بأخدود (الخيار **relief =GROOVE**) . يمكنك بسهولة الحصول على مناطق أخرى عن طريق خيار **relief** و **bd** (الحافة في تعليمة المثيل لهذه الأزرار) . و خاصة, يمكن إختيار **relief =FLAT** و **bd =0** للحصول على أزرار صغيرة "مسطحة", بدون أي **relief** .

نوافذ مع قوائم

و لإنهاء زيارتنا الصغير لويدجات tkinter, سنقوم الأن بشرح منشئ نافذة لتطبيق مع أنواع مختلفة من القوائم "القاعدة", كل واحد من هذه القوائم قادر على "الفصل" من التطبيق الرئيسي لتصبح نافذة مستقلة, كما هو مبين على الجانب .

و لإنهاء زيارتنا الصغير لويدجات tkinter, سنقوم الأن بشرح منشئ نافذة لتطبيق مع أنواع مختلفة من القوائم "القاعدة", كل واحد من هذه القوائم قادر على "الفصل" من التطبيق الرئيسي لتصبح نافذة مستقلة, كما هو مبين على الجانب .

كما شرحنا سابقا[70], هذا الأسلوب لبدأ كتابة برنامج بمسودة, الذي يحتوي فقط على بضعة أسطر لكنها فعالة . سوف نجرب إذا هذه المسودة بعناية للقضاء على

[70] إنظر إلى صفحة Error: Reference source not found : البحث عن الأخطاء و التجريب.

أي علل . عندما تعمل المسودة بشكل جيد, سوف نضيف الوظائف الإضافية . نحن نختبر هذا الملحق حتى يعطي الإرتياح, ثم نضيف جزء أخر, و هكذا ...

هذا لا يعني أنه لا يمكنكالبدء فورا بالبرنامج دون إجراء تحليل دقيق للمشروع, على لأقل يجب أن يكون المخطط موصوف على نحو كافي و بوضوح في كراس المواصفات .

> و من الضروري أن تقوم بوضع تعليق بشكل صحيح على كود المنتج عند تطويره . و نحن نسعى لكتابة تعليقات جيد و الذي هو في الواقع ضروري, ليس فقط ليكون كودك أسهل للقراءة (و بالتالي للحفاظ على الوقت الاخرين و لك), و لكن أيضا إذا إضطررت للتعبير عن ما تريد حقا أن يفعله (أنظر إلى الأخطاء الدلائلية صفحة *Error: Reference source not found*.)

مواصفات التمرين

تطبيقنا يحتوي ببساطة على شريط قوائم و لوحة . عناوين مختلف و خيارات القوائم تؤدي إلى إظهار أجزاء نص في اللوحة أو في تعديل تفاصيل الديكور, و لكن سيكون في البداية مجموعة من الأمثلة, و تهدف إلى إعطائك العديد من الإحتمالات الذي يقدمها هذا النوع من الويدجت, الإكسسوارات الأساسية لأي تطبيق حديث الذي لديه بعض الأهمية .

نريد أيضا تنظيف كود البرنامج جيدا في هذا التمرين . للقيام بهذا, سوف نستخدم صنفين : صنف للتطبيق الرئيسي, و أخر لشريط العنوان . و نحن نريد أن نسلط الضوء على منشئ تطبيق يتصمن عدة أصناف كائن تفاعلي .

أول مسودة للبرنامج

عند بناء مسودة لبرنامج, يتعين علينا أن نحاول نجمع الهيكل, مع العلاقات بين الكتل الرئيسية التي تشكل التطبيق النهائي . هذا ما نحاول القيام به في المثال أدناه .

```
1#    from tkinter import *
2#
3#    class MenuBar(Frame):
4#        """Barre de menus déroulants"""
5#        def __init__(self, boss =None):
6#            Frame.__init__(self, borderwidth =2)
7#
8#            #####قائمو <ملف> #####
```

```
9#          fileMenu = Menubutton(self, text ='Fichier')
10#         fileMenu.pack(side =LEFT)
11#         #Partie "déroulante" :
12#         me1 = Menu(fileMenu)
13#         me1.add_command(label ='Effacer', underline =0,
14#                 command = boss.effacer)
15#         me1.add_command(label ='Terminer', underline =0,
16#                 command = boss.quit)
17#         #Intégration du menu :
18#         fileMenu.configure(menu = me1)
19#
20#   class Application(Frame):
21#       """Application principale"""
22#       def __init__(self, boss =None):
23#         Frame.__init__(self)
24#         self.master.title('Fenêtre avec menus')
25#         mBar = MenuBar(self)
26#         mBar.pack()
27#         self.can = Canvas(self, bg='light grey', height=190,
28#                   width=250, borderwidth =2)
29#         self.can.pack()
30#         self.pack()
31#
32#       def effacer(self):
33#         self.can.delete(ALL)
34#
35#   if __name__ == '__main__':
36#       app = Application()
37#       app.mainloop()
```

لذا يرجى الأن ترميز هذه الأسطر و تجربة تشغيلها . يجب عليك أن تحصل على نافذة مع لوحة رمادية فاتحة يعلوها شريط القوائم . في هذه المرحلة, شريط القوائم لا يحتوي سوى على قائمة وحيدة "الملف - Fichier" .

أنقر على "الملف" لإظهار القائمة : الخيار "مسح - Effacer" لا يعمل بعد (سوف يمحو محتويات اللوحة), لكن الخيار "إنهاء - Terminer" يجب أن يغلق بالفعل التطبيق بشكل صحيح .

مثل جميع القوائم التي صنعناها بواسطة tkinter, القائمة التي صنعتها يمكنها تحويل إلى قائمة "تعويم" : يكفي أن تضغط ببساطة على سطر المنقط على رأس القائمة . سوف تحصل على نافذة "قمر صناعي" صغيرة, التي يمكنك وضعها في أي مكان تريد على سطح المكتب .

تحليل السكريبت

هيكل هذا البرنامج الصغير يجب أن يظهر لك المعتاد : إن الفئات التي تم تعريفها في هذا السكريبت يمكن أن تستخدم مرة أخرى في مساريع أخرى عن طريق الإستيراد, كما شرحناه سابقا[71], جسم البرنامج الرئيسي (السطور من 35 إلى 37) تحتوي على البيان الكلاسيكي : **if __name__ == '__main__':**

التعليمتان التاليتين تتكون فقك بتمثيل كائن app و تشغيل أسلوبه mainloop() . كما تعلمون, فإننا يمكننا إختصار هتان التعليمتان إلى واحدة .

أساس البرنامج تجدها في تعريفات الصنف السابقة .

الصنف **MenuBar()** تحتوي على وصف لشريط القوائم . في هذه الحالة للسكريبت, تتلخص في مسودة المنشئ .

* السطر 5 : البرامتر **boss** يتلقى مرجع النافذة الرئيسية للويدجت في لحظة تمثيله . هذا المرجع يسمح لنا بإستدعاء الأساليب المرتبطة بهذه النافذة الأساسية, في الأسطر 14و 16 .

* السطر 6 : التفعيل الإلزامي لمنشئ الصنف الأصل .

* السطر 9 : تمثيل ويدجت صنف **Menubutton()**,ستم تعرفه على أنه "عبيد" ل self (هذا معناه كائن مركب "شريط القوائم" لذا نحن منشغلون بتحديد الصنف) . كما يوحي إسمه, هذا النوع من الويدجت يتصرف قليلا مثل الزر : الحركة تعمل عند الضغط على الزر .

---

[71] أنظر إلى صفحة 237 : وحدات تحتوي على مكتبات الأصناف.

* السطر 12 : هذا العمل يعمل على إظهار قائمة حقيقية, يبقى تعريفه : أنه ويدجت جديد, للصنف **Menu** هذه المرة. يتم تعريفه على أنه "عبيد" للويدجت **Menubutton** الذي تم تمثيله في السطر 9 .

* الأسطر من 13 إلى 16 : يمكننا تطبيق على الويدجات للصنف **Menu()** عدد من الأساليب المحددة, كل واحدة يقبل العديد من الخيارات . نحن نستخدم هنا الأسلوب **add_command()** لتثبيت في القائمة عنصرين "مسح - Effacer" و "إنهاء - Terminer" . سوف ندمج الأن, الخيار **underline**, الذي يستخدم لتعريف إختصار لوحة المفاتيح : هذا الخيار يشير إلى أي حروف للعنصر يجب أن تظهر مسطرة على الشاشة . المستخدم يعرف أنه إذا ضغط على هذا الحرف من لوحة المفاتيح ليتم تفعيل وظيفة هذا العنصر (كما لو نقرنا بالفأرة) .

سيعمل عندما يقوم المستخدم بتحديد عنصر من الخيار **command** . في سكريتنا الأوامر المعطاة هي أسلوبين للنافذة الأصل, وسوف يتم تمرير الويدجت في لحظة تمثيله عن طريق البرامتر **boss** . الأسلوب **effacer()** , الذي سنعرفه لاحقا, يعمل على مسح اللوحة . الأسلوب المعرف مسبقا **quit()** يعمل على الخروج من حلقة **mainloop()** و إيقاف إستقبال الأحداث المرتبطة مع نافذة التطبيق .

* السطر 18 : عندما يتم تعريف عنصر من القائمة, لايزال بحاجة إلى إعادة تكوين ويدجت الأصل **Menubutton** بحيث الخيار "menu - قائمة" هي في الواقع القائمة التي نريد بناها . في الواقع نحن لا نستطيع تحديد هذا الخيار عند التعريف الأصلي للويدجت **Menubutton** لأن في هذه المرحلة, القائمة لم تكن موجودة بعد . لا نستطيع أن لا نهرف الويدجت Menu في المكان الأول, لأن هذا سوف يتم تعريفه على أنه "عبيد" لويدجت **Menubutton** . يجب أن نفعل ذلك عن طريق 3 خطوات كما فعلنا سابق, يجب أن نستدعي الأسلوب **configure** . هذا الأسلوب يمكن تطبيقه على أي ويدجت موجودة لتعديل خيار أو أكثر .

الصنف **Application()** يحتوي على وصف للنافذة الرئيسية و أساليب معالجة الأحداث التي مرتبطة بها .

* السطر 20 : نحن نفضل إشتقاق تطبيقنا من الصنف **Frame()**, الذي لديه خيارات كثيرة, بدلا من الطبقة الأولية **Tk()** . بهذه الطريقة, التطبيق يتم تغليفه في ويدجت, و الذي قد يكون متكاملا في

وقت لاحق في تطبيق أهم . تذكر أن, في أي حال, tkinter يمثل تلقائيا نافذة الأصل من نوع **Tk()**, لإحتواء هذا الإطار .

* السطران 23 و 24 : بعد تفعيل اللازمة من المنشئ لصنف الأصل, نحن إستخدمنا سمات **master** التي tkinter مرتبطة بها تلقائيا في كل ويدجت, لمرجع الصنف لأصل (في الحالة السابقة, الكائن هو النافذة الرئيسية للتطبيق) و إعادة تعريف شعار-العنوان .

* الأسطر من 25 إلى 29 : تمثيل لويجدتان "عبيد" لإطارنا (**Frame**) الرئيسي . من الواضح أن شريط القوائم هو ويدجت معرف في صنف أخر .

* السطر 30 : مثل أي قطعة من أي ويدجت أخر, إطارنا الرئيسي يجب أن يكون معهود لأسلوب لتنفيذ إظهار الحقيقية .

* الأسطر 32 و 33 : الأسلوب المستخدم لمسح اللوحة يتم تعرفه في النصف (لأن الكائن لوحة في الحقيقة جزء), لكن يتم إستدعائها من خلال الخيار **command** للويدجت العبيد الذي تم تعريفه في صنف أخر .

كما شرحنا أعلاه, هذا الويدجت يتلقى مرجع لويدجته السيد عن طريق البرامتر **boss** . كل هذه المراجع تحدد أولويات بمساعدة إشارات أسماء بالنقاط .

إضافة القائمة Musiciens (الموسيقيين)

واصل تطوير هذا البرنامج الصغير, بإضافة أسطر التالية في منشئ الصنف **MenuBar()** (بعد السطر 18) :

```
##### قائمة <الموسيقيين> #####
self.musi = Menubutton(self, text ='Musiciens')
self.musi.pack(side =LEFT, padx ='3')
#: جزء "أسفل" قائمة الموسيقيين
me1 = Menu(self.musi)
me1.add_command(label ='17e siècle', underline =1,
        foreground ='red', background ='yellow',
        font =('Comic Sans MS', 11),
```

```
            command = boss.showMusi17)
    me1.add_command(label ='18e siècle', underline =1,
            foreground='royal blue', background ='white',
            font =('Comic Sans MS', 11, 'bold'),
            command = boss.showMusi18)
    #: تكامل القائمة
    self.musi.configure(menu = me1)
```

... و تعاريف الأساليب التالية للصنف **Application()** (بعد السطر 33) :

```
    def showMusi17(self):
        self.can.create_text(10, 10, anchor =NW, text ='H. Purcell',
                    font=('Times', 20, 'bold'), fill ='yellow')

    def showMusi18(self):
        self.can.create_text(245, 40, anchor =NE, text ="W. A. Mozart",
                    font =('Times', 20, 'italic'), fill ='dark green')
```

عندما تقوم بإضافة كل هذه الأسطر, قم بحفظ السكريبت و قم بتشغيله .

شريط القوائم يضم الأن قائمة جديدة : قائمة "Musiciens" (الموسيقيين) .

القائمة المقابلة تقدم عنصرين التي تظهر مع ألوان و خطوط شخصية . يمكنك أن تتعلم هذه التقنيات الزخرفية لمشاريعك الشخصية .

الأوامر التي قمت بربطها مع هذه العناصر هي مبسطة حتى لا نتعب في التمرين : لأنها تسبب عرض نصوص صغيرة على اللوحة .

تحليل السكريبت

التغيرات الوحيدة التي أدخلت في هذه الأسطر هي إستخدام حطوط للحروف المحددة (الخيار **font**), و لون المقدمة (الخيار **foreground**) و لون الخلفية (الخيار **background**) للنصوص المعروضة .

يرجى ملاحظة مرة أخرى أن إستخدام الخيار **underline** لتعيين حرف ليكون إختصا للوحة المفاتيح (لا ننسى أن ترقيم الأحرف يبدأ من الصفر), و خاصة الخيار **command** لهذه الويدجات تصل إلى أساليب للصنف الأخر, من خلال مرجع مخزن في سمة **boss** .

الأسلوب **create_text()** للوحة يجب إستخدامه مع برامترين رقميين, و الذان هما الإحداثيات **X** و **Y** للنقطة في اللوحة . النص الممرر سيتم وضعه في هذه النقطة, في دالة للقيمة المخترة للخيار **anchor** : هنا تم تحديدها كيفية يجب ان يكون جزء النص "راسية" في النقطة المختارة في اللوحة, في وسطه أو في الجانب أقصى اليسار أو إلخ ..., في دالة بناء جملة التي تستخدم قياسا على نقاط الأساسية الجغرافية (**NW** = لأقصى اليسار و **SE** = لليسار العلوي, و **CENTER** = للوسط, إلخ ...) .

إضافة قائمة Peintres (الرسامين)

هذه القائمة الجديدة تم صنعاها بطريقة مماثلة تماما لسابقتها, لكن نحن أضفنا وظيفة إضافية : القوائم "الشلالات" . يرجى إذا إضافة الأسطر التالية في منشئ الصنف **MenuBar()** :

```
##### قائمة <الرسامين > #####
self.pein = Menubutton(self, text ='Peintres')
self.pein.pack(side =LEFT, padx='3')
# جزء الأسفل :
me1 = Menu(self.pein)
me1.add_command(label ='classiques', state=DISABLED)
me1.add_command(label ='romantiques', underline =0,
```

```
        command = boss.showRomanti)
#: قائمة فرعية للرسامين الإنطباعيين
me2 = Menu(me1)
me2.add_command(label ='Claude Monet', underline =7,
        command = boss.tabMonet)
me2.add_command(label ='Auguste Renoir', underline =8,
        command = boss.tabRenoir)
me2.add_command(label ='Edgar Degas', underline =6,
        command = boss.tabDegas)
#: تكامل القائمة الفرعية
me1.add_cascade(label ='impressionistes', underline=0, menu =me2)
#تكامل القائمة :
self.pein.configure(menu =me1)
```

... و تعاريف الأساليب التالية للصنف **Application()** :

```
def showRomanti(self):
    self.can.create_text(245, 70, anchor =NE, text = "E. Delacroix",
                font =('Times', 20, 'bold italic'), fill ='blue')

def tabMonet(self):
    self.can.create_text(10, 100, anchor =NW, text = 'Nymphéas à Giverny',
                font =('Technical', 20), fill ='red')

def tabRenoir(self):
    self.can.create_text(10, 130, anchor =NW,
                text = 'Le moulin de la galette',
                font =('Dom Casual BT', 20), fill ='maroon')

def tabDegas(self):
    self.can.create_text(10, 160, anchor =NW, text = 'Danseuses au repos',
                font =('President', 20), fill ='purple')
```

تحليل السكريبت

يمكنك بسهولة صنع القوائم الشلالات, من خلال ربط القوائم الفرعية مع بعضها البعض في مستوى معين (و لكن ينصح أن لا تتجاوز خمسة مستويات متتالية, لأنك ستخسر مستخدمك).

يتم تعريف القائمة الفرعية على أنها قائمة عبيد لقائمة في المستوى السابق (في مثالنا, **me2** تم تعريفها على أنها قائمة "عبيد" ل **me1**) . و سيتم دمجها بمساعدة الأسلوب **add_cascade()** .

واحدة من العناصر معطلة (الخيار **state = DISABLED**) . المثال التالي يظهر لك كيف يمكنك تعطيل أو تفعيل عناصر, من خلال البرنامج .

إضافة القائمة Option (خيارات)

تعريف هذه القائمة هو قليلا أكثر تعقيدا, لأننا صوف تصمن إستخدام المتغيرات الداخلية ل tkinter .

وظيفة هذه القائمة هي أكثر تفصيلا : الخيارات المضافة تجعل من الممكن تعطيل أو تفعيل القوائم " Musiciens" و "Peintres" و يمكنك تغيير مظهر شريط القوائم نفسه .

يرجى إذا إضافة الأسطر التالية في منشئ الصنف **MenuBar()** :

```
##### قائمة <خيارات #####
optMenu = Menubutton(self, text ='Options')
optMenu.pack(side =LEFT, padx ='3')
# متغيرات tkinter :
self.relief = IntVar()
self.actPein = IntVar()
self.actMusi = IntVar()
# جزء "أسفل" القائمة
```

```
    self.mo = Menu(optMenu)
    self.mo.add_command(label = 'Activer :', foreground ='blue')
    self.mo.add_checkbutton(label ='musiciens',
          command = self.choixActifs, variable =self.actMusi)
    self.mo.add_checkbutton(label ='peintres',
          command = self.choixActifs, variable =self.actPein)
    self.mo.add_separator()
    self.mo.add_command(label = 'Relief :', foreground ='blue')
    for (v, lab) in [(0,'aucun'), (1,'sorti'), (2,'rentré'),
              (3,'sillon'), (4,'crête'), (5,'bordure')]:
      self.mo.add_radiobutton(label =lab, variable =self.relief,
                    value =v, command =self.reliefBarre)
    #: تكامل القائمة
    optMenu.configure(menu = self.mo)
```

... و تعاريف الأساليب التالية دائما للصنف **Application()** :

```
    def reliefBarre(self):
        choix = self.relief.get()
        self.configure(relief =[FLAT,RAISED,SUNKEN,GROOVE,RIDGE,SOLID][choix])

    def choixActifs(self):
        p = self.actPein.get()
        m = self.actMusi.get()
        self.pein.configure(state =[DISABLED, NORMAL][p])
        self.musi.configure(state =[DISABLED, NORMAL][m])
```

قائمة مع خانات إختيار

القائمة الجديدة تتكون من جزئين . و لتسليط الضوء, لقد قمنا بإدراج سطر فاصل و 2 من العناصر الكاذبة (« Activer : » و « Relief : ») التي تخدم العناوين . لقد قمنا بإظهار هذه بلون لا يمكن للمستخدم الخلط مع الأوامر الحقيقية .

تم تجهيز العناصر التالية في الجزء لأول من خانات الإختيار . عندما يقوم المستخدم بالضغط من خلال الفأرة على عنصر أو أكثر من هذه العناصر, يتم تفعيل أو تعطيل الخيارات, و هذه الحالة « actif / inactif » يتم عرضها في شكل خانات . التعليمات التي تخدم تنفيذ هذا النوع من القائمة هي ذاتية التوضيح . لديها في الحقيقية هذه العناصر كويدجات من نوع **chekbutton** :

```
self.mo.add_checkbutton(label = 'musiciens', command = choixActifs,
                        variable = mbu.me1.music)
```

من المهم أن نفهم هنا أن هذا النوع من الويدجت يحتوي بالضرورة متغيرات داخلية, لتخزين حالة « actif / inactif » للويدجت . كما قمنا بتفسير هذا أعلاه, هذا المتغير لا يمكن أن يكون متغير بايثون عاديا, لأن أصناف مكتبة tkinter تم كتابتهم بلغات أخرى . و بالتالي لا يمكننا الوصول إلى هذا المتغير الداخلي إلا من خلال كائن-الواجهة, و الذي نسميه متغير tkinter لتبسيطه[72].

حتى في مثالنا, إستخدمنا صنف **IntVar()** tkinter لصنع كائنات تعادل متغيرات نوع صحيح .

* لقد قمنا بتمثيل هذه كائنات-المتغيرات, التي قمنا بتخزينها كسمات مثيل : **self.actMusi=IntVar()** .

بعد هذه المهمة, كائن مرجع في **self.actMusi** يحتوي على ما يعادل متغير من نوع صحيح, بشكل خاص ل tkinter .

* ثم يجب ربط خيار المتغير كائن **checkbutton** لمتغير tkinter الذي تم تعريفه :

```
self.mo.add_checkbutton(label =’musiciens’, variable =self.actMusi)
```

* من الضروري أن تفعل ذلك في خطوتين, لأن tkinter لا يقبل تعيين قيم لمتغيرات بايثون . لسبب بسيط, لا يمكن للبايثون قراءة مباشرة مختوى متغير tkinter . يجب أن تستخدم أساليب خاصة لهذا صنف الكائن : الأسلوب **get()** للقراءة, و الأسلوب **set()** للكتابة :

```
m = self.actMusi.get()
```

72 أنظر أيضا إلى صفحة Error: Reference source not found.

في هذه التعليمات, قمنا بتعيين ل **m** (متغير عادي لبايثون) محتوى المتغير tkinter **self.actMusi** (الذي هو في حد ذاته يرتبط مع تطبيق واضح و محدد) .

كل ما سبق قد يبدو معقدا قليلا . أنظر ببساطة إلى أول مواجهة مع مشاكل التواصل بين لغتي برمجة مختلفتين, المستخدمة معا في مشروع مركب .

قائمة مع خيارات حصرية

الجزء الثاني من قائمة "Options" يسمح للمستخدك بإختيار شكل من شأنه أن يجعل شريط القوائم, بستة خيارات . و غني عن القول أنه يمكنك تفعيل إحدى هذه الخيارات في كل مرة . لتنفيذ هذا النوع من الوظائف, نقوم كلاسكيا بإستدعاء الويدجات من نوع "أزرار راديو" . و الميزة الأساسية من هذه الويدجات هي أنه يجب أن تكون مرتبطة مع متغير tkinter . كل زر راديو يتوافق مع قيمة معينة, و هذه القيمة تم تعيينها عندما يقوم المستخدم بتحديد الزر .

و هكذا, التعليمة :

```
    self.mo.add_radiobutton(label ='sillon', variable =self.relief,
                            value =3, command =self.reliefBarre)
```

إن تكوين عنصر من قائمة "Options" بحيث يتصرق مثل زر راديو .

عندما يقوم المستخدم بتحديد هذا العنصر, يتم تعيين القيمة 3 إلى المتغير **self.relief** tkinter (و يشار إلى هذا الأخير بمساعدة خيار variable للويدجت), و يتم إجراء الإستدعاء نحو أسلوب **()reliefBarre** . و هذا يسترد القيمة المخزنة في المتغير tkinter للقيام بعمله .

في سياق معين من هذه القائمة, أردنا أن نقدم 6 خيارات مختلفة للمستخدم . لذا نحن بحاجة إلى 6 "أزرار راديو", التي تمكننا من ترميز 6 تعليمات متماثلة التي كنت تستنسخها في الأعلى, كل واحدة منها لا تختلف عن الخمسة الأخرى إلا بخياراتها **value** و **label** . في مثل هذا الوضع, الممارسة الجيدةهو وضع قيم نت هذه الخيارات في قائمة, ثم قم بإستعراض القائمة بإستخدام حلقة for, ثم قم بإنشاء مثيل ويدجات مع تعليمة مشتركة :

```
    for (v, lab) in [(0,'aucun'), (1,'sorti'), (2,'rentré'),
                     (3,'sillon'), (4,'crête'), (5,'bordure')]:
        self.mo.add_radiobutton(label =lab, variable =self.relief,
                                value =v, command =self.reliefBarre)
```

القائمة تستخدم قائمة متكونة من 6 أنفاق (قيم, ملصقات) . في كل واحدة من 6 تكرارات الحلقة, يتم إنشاء مثيل زر راديو جديد, و يتم إستخراج الخيارات **label** و **value** من قائمة من خلال المتغيرات **lab** و **v** .

في مشاريعك الخاصة, سوف تجد في كثير من الأحيان أنه يمكنك إستبدال تعليمات متماثلة ب هيكل برمجي أكثر إحكاما (عادة مزيج من قائمة و حلقة, مثل في المثال أعلاه) .

سوف تكتشف شيئا فشيئا تقنيات أخرى لتخفيف كودك : سوف نقدم مثال في الفقرة القادمة . و مع ذلك حالو أن تأخذ في الإعتبار هذه القاعدة : البرنامج الجيد يجب أن يكون دائما قابل للقرائة و معلق أيضا .

التحكم في تدفق التنفيذ بمساعدة قائمة

يرجى النظر الأن إلى تعريف الأسلوب **reliefBarre()** .

في السطر الأول, الأسلوب **get()** يسمح لنا بإسترداد حالة المتغير tkinter الذي يحتوي على رقم الإختيار الذي أدلى به المستخدم في القائمة الفرعية " Relief : ".

في السطر الثاني, إستخدمنا محتوى المتغير **choix** لإستخراج قائمة بها 6 عناصر التي نحن مهتمين بها . على سبيل المثال, إذا كان **choix** يحتوي على القيمة 2, سيكون المتغير **SUNKEN** هو المستخدم لإعادة تكوين الويدجت .

المتغير **choix** هو المستخدم هنا كمؤشر, و يستخدم لتعيين عنصر قائمة . بدلا من هذا البناء المدمج, و يمكننا برمجة سلسلة من الإختبارات الشرطية, مثل :

```
if choix ==0:
    self.configure(relief =FLAT)
elif choix ==1:
    self.configure(relief =RAISED)
elif choix ==2:
    self.configure(relief =SUNKEN)
...
etc.
```

من وجهة نظر وظيفية, ستكون النتيجة نفسها . يجب الإعتراف أن البناء الذي إخترناه هو أكثر فعالية حيث أن عدد من الخيارات تم إزالتاه . تخيل على سبيل المثال, أن أحد برامجك الشخصية

يجب الإختيار بين عدد كبير من العناصر و هي : بناء من نوع أعلاه, قد تكون هنالك حاجة لترميز صفحات متعدد من elif !

و نحن لا نزال نستخدم نفس التقنية في الأسلوب **choixActifs()** . التعليمة :

```
self.pein.configure(state =[DISABLED, NORMAL][p])
```

يستخدم محتوى المتغير **p** كمؤشر للإشارة إلى أي حالة **DISABLED** و **NORMAL** يجب أن يتم تحديدها لإعادة تكوين القائمة "Peintres" .

عندما يتم إستدعاء الأسلوب **choixActifs()** يتم إعادة تكوين قائمتين **Peintres** و **Musiciens** لشريط القائمة, لجعلها تبدو "طبيعية" أو "معطلة" وفقا لحالة المتغيرات **m** و **p**, و التي هي في حد ذاتها تعبير عن متغيرات tkinter .

هذه المتغيرات **m** و **p** خب في الواقع لتوضيح السكريبت . سكون من الممكن حذفها, و جعل السكريبت أكثر إحكاما, بإستخدام تركيبة من التعليمات . و يمكننا على سبيل المثال إستبدال التعليمتان :

```
m = self.actMusi.get()
self.musi.configure(state =[DISABLED, NORMAL][m])
```

بتعليمة واحدة فقط, مثل هذه :

```
self.musi.configure(state =[DISABLED, NORMAL][self.actMusi.get()])
```

نلاحظ أننا يمكننا أن نربح, لكن عندما نختصره نفقد بعض الوضوح .

الضبط المسبق لقائمة

لإنهاء هذا التمرين, يمكنك أن ترى أنك يمكنك أن تضع تحديدات مسبقا, أو تعديل البرنامج .

لذا يرجى قبل إضافة التعليمة التالية في منشئ للصنف **Application()** (فقط قبل التعليمة **()self.pack**, على سبيل المثال) :

```
mBar.mo.invoke(2)
```

عند تشغيل السكريبت المعدل, يمكنك أن ترى في بداية أن قائمة **Musiciens** في شريط القوائم مفعلة, في حين أن **Peintres** ليست كذلك . برمج كما هي, و ينبغي أن يكون هذان القائمتين مفعلتين بشكل إفتراضي . و هذا في الواقع ما يحدث إذا قمنا بحذف التعليمة

```
mBar.mo.invoke(2)
```

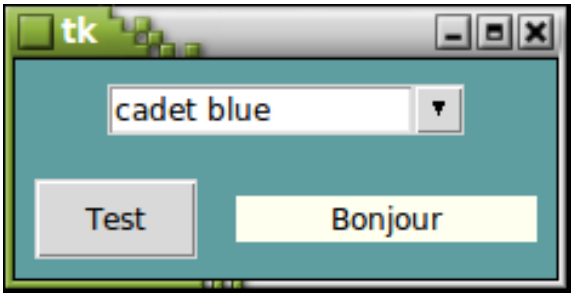
لقد إقترحنا إضافة هذه التعليمة إلى السكريبت لنبين لكم كيف يمكنك تنفيذ نفس العملية من قبل البرنامج مثل التي عادة الضغط من الفأرة .

التعليمة أعلاه تستدعي الويدجت **mBar.mo en** من خلال تشغيل الأمر المرتبط للعنصر لثاني لهذا الويدجت . بالرجوع إلى القائمة, يمكنك التحقق من أن العنصر الثاني هو كائن من نوع **checkbutton** الذي يقوم بتفعيل\تعطيل قائمة **Peintres** (تذكر مرة أخرى أن الترقيم يبدأ دائما من الصفر) .

في بداية البرنامج, كل شيئ يحدث كما لو كان المستخدم قام على الفور بالضغط على قائمة **Peintres** لقائمة **Options**, مما يؤدي إلى تعطيل القائمة .

تمرين

14.1 أكمل الويدجت "مكعب كومبو المبسط" الذي تم وصفه في الصفحة 225, بحيث يتم إخفاء قائمة البداية, و الزر الصغير على اليمين حقل الإدخال لا يظهر . ما عليك القيام به هو وضع قائمة و شريط تمرير في تافذة "قمر صناعي" بدون حدود (أنظر إلى الويدجت **Toplevel**, الصفحة 235), وضعه بشكل صحيح (قد تحتاج إلى الرجوع إلى مواقع التعامل مع tkinter للعثور على المعلومات الضرورية, لكن سيكون هذا من جزء تعلمك !), و تأكد من أن هذه النافذة تختفي بعد أن يقوم المستخدم بتحديد عنصر في قائمة .



15

تحليل برنامج محدد

في هذا الفصل, سنحاول توضيح عملية تصميم برنامج رسومي, من المسودة الأولى غلى مرحلة متقدمة نسبيا من التطوير . نريد أن نظر مدى البرمجة الشيئية تمكن أن تسهل و تأمن إستراتيجية تطوير إضافية التي نريدها[73].

مطلوب إستخدام الأصناف, عندما سكون هنالك مشروع جاري نثبت أنه أكثر تعقيدا مما كنا نتخيل أصلا . بالتأكيد سوف تعيش حياة بمسارات مماثل لتكل التي وصفناها .

لعبة القصف

هذا المشروع[74] أستلهم من همل مماثل أنتجه طلاب السنة النهائية .

من المستحسن البدء بمشروع مثل هذا المشروع من خلال سلسلة من الرسومات الصغيرة و الرسوم البيانية, التي تم وصفها في عناصر بناء رسم مختلفة, و التي تسخدم بكثرة . إذا كنت لا ترغب في إستخدام التكنولوجيا القديمة ورقة\قلم (المبرهنة حتى الان), يمكنك الإستفادة من برامج الرسم التقني, مثل إستخدام Draw من المجموعة المكتبية أوبن أوفيس[75] التي كنا قد جعلنا الرسم البياني في الصفحة التالية .

---

[73] أنظر إلى صفحة Error: Reference source not found : البحث عن الأخطاء و التجريب, و أيضا إلى صفحة Error: Reference source not found : نوافذ مع قوائم

[74] نحن لا نتردد هنا في مناقشة تطوير لعبة, لأنه مجال متاح للجميع, و هي أهداف محددة يسهل التعرف إليها . و يمكن تطبيق نفس الأساليب المنتمية إلى تطبيقات أحرى أكثر "جدية" .

الفكرة بسيطة : لاعبين يتنافسان في برميل . يجب على كل واحد منهم ضبط زاوية إطلاق النار ليحاول الوصول إلى خصمه, و القذائف البالستية تصف المسارات .

و يتم تعريف عرف مكان وجود المدفع في بداية اللعبة بشكل عشوائي (على الأقل في الأعلى) . بعد كل طلقة, يتم إستبدال البنادق (لزيادة الإهتمام بالعبة, و بالتالي يتم تعديل الطلقات أكثر صعوبة) . يتم تسجيل التسديدات على الهدف .

التصميم الأولي الذي قمنا بإستنساخه أعلاه هو واحد من النماذج التي يمكن ان تستغرق عملك التحليلي . قبل البدء في تطوير مشروع برمجة, يجب أن نسعى دائما تفصيل المواصفات . و هذه الدراسة الأولية مهمة للغاية . معظم المبتدئين بدأوا بسرعة بكتابة اسطر عديد من الكود مع فكرة غامضة, لكنه يتجاهل البحث عن هيكل العام . و قد تصبح برمجتهم خطيرة ثم تصبح فوضى, لأنها ستنفذ هذا الهيكل عاجل ام اجلا . و في كثير من الأحيان يبدأ بالحذف و إعادة كتابة قطاعات بأكملها من المشروع لأنها صممت بطريقة متجانسة جدا\او تكوينها بشكل غير صحيح .

* متجانسة جدا : هذا يعني قد فشل في كسر مشكلة معقدة إلى عدة مشاكل فرعية صغيرة أكثر بساطة . على سبيل المثال, تداخل العديد من المستويات من تعليمات مركبة, بدلا من إستخدام دالات أو أصناف .

* سوء تكوينه : هذا يعني انه يتعامل فقط مع حالة معينة, بدلا من دراسة الحالة العامة . على سبيل المثال, أعطينا لكائن رسومي أبعاد ثابة, بدلا من توفير متغيرات للسماح بتغيير حجمه .

عندما تريد أن تطور مشروع يجب أن تبدأ دائما بمرحلة التحليل, و تنفيذ نتائج هذا التحليل في مجموعة من الوثائق (رسومات و خطط و وصف ...إلخ) التي تشكل المواصفات . للمشاريع الكبيرة, هنالك أيضا أساليب متطورة للتحليل (UML, Merise...إلخ) و هذه لا نستطيع شرحها هنا لأنها تحتاج كتب بأكملها .

يقال, و يجب أن أعترف أنه صعب جدا (و ربما مستحيل) تحليل مشروع برمجي في البداية بأكمله . لأننا عند تشغيله سوف نعرف نقاط ضعفه . و تبقى هنالك حالات إستخدام او قيود لم نقصدها أصلا .

---

[75] MS-Office, و هو مجموعة مكتبية كاملة, حرة و مجانية, و هي متوافقة على نطاق واسع مع متاحة للينكس و ويندوز و ماك و يولاريس ... و لقد كتب هذا الكتاب بواسطة معالج النصوص الخاص بع . يمكنك الحصول عليه عن طريق تحميله من موقع : http://www.openoffice.org

من جهة أخرى, مشروع برمجي هو دائما يحتاج إلى التطوير : سوف يحتاج في كثير من الأحيان إلى تعديل المواصفات أثناء التطوير, و ليس بالضرورة أنك أخطأت في التحليل الأولي, لكن ببساطة أنك تريد إضافة مميزات إضافية .

و في الختام, حاول دئما التعامل مع مشروع برمجة جديد بإحترام النقطتين التاليتين :

* صف مشروعك بالتفصيل قبل البدء بكتابة السطور الأولى من التعليمات البرمجية, لتسليط الضوء على المكونات الرئيسية و العلاقة بينها (أعتقد وصف حالات الإستخدام لمستخدم برنامجك) .

* عند البدء في العمل الحقيقي, لا تسترسل في كتابة كتل كبيرة من التعليمات . تأكد من تقطيع برنامجك لعدد من المكونات القابلة للتكوين مغلفة بشكل جيد, بحيث يمكنك بسهولة تعديل أس واحد منهم دون المساس بتشغيل الأخرين, و حتى إعادة إستخدامهم في سياقات مختلفة إذا دعت الحاجة إلى ذلك .

و لتلبية هذا الطلب تم إختراع البرمجة الشيئية .

على سبيل المثال أنظر إلى المشروع المرسوم في الصفحة السابقة .

قد يميل المبتدأ بإجراء هذه اللعبة بإستخدام البرمجة الإجرائية فقط (هذا معناه عدم تحديد أصناف جديدة) . و هذا هو أيضا كيف قمنا بمعالجة خلال الفصل الأول في الواجهات الرسومية, من الفصل 8 . و هذا النهج لا يبرر أن البرامج الصغيرة (تمارين أو إختبارات أولية) . عند معالجة مشروع من حجم معين, و تعقيد المشاكل التي نشأة بسرعة سوف تصبح كبيرة جدا, و سيصبح من الضروري تفكيكه و تجزئته .

الأداة البرمجة التي تسمح بهذه التجزئة هي الصنف .

ربما سوف تفهم أفضل فائدته بمساعدته بالقياس .

جميع الأجهزة الالكترونية تتكون من عدد قليل من المكونات الأساسية, و هي الترانزستورات و الثنائيات و المقاومات و المكثفات و إلخ . بنيت الحواسيب الأولى مباشرة من هذه المكونات . و كانت ضخمة و مكلفة و كانت لديهم وظائف قليل جدا و دائما ما تتعطل .

ثم قامو بتطوير تقنيات جديدة لتغليف في علبة مجموعة كبيرة من المكونات الالكترونية الأساسية . لإستخدام هذه الدوائر الكهربائية المدمجة, لم يعد من الضروري معرفة محتوياته بالضبط : وظيفة واحدة مهمة فقط . كانت الوظائف الأولة المتكاملة لا تزال نسبيا بسيطة : كانت على سبيل المثال, بوابات منطقية, مزازيات ... إلخ . من خلال الجمع بينهذه الدوائر معا, سوف نحصل على المزيد من المميزات المتقدمة, مثل السجلات أو أجهزة فك تشفير, و الذي بدوره يجب أن يكون متكامل و هكذا, إلى المعالجات الحالية . و هي تتكون من الملايين من المكونات, و مع ذلك لديها موثوقية عالية .

وفقا لذلك, للإلكترونيات الحديثة التي تريد بناء على سبيل المثال عداد ثنائي (الئارة تتطلب عدد من المقاييس), فمن الواضح أنه أسهل بكثير و أسرع و أكثر أمانا لإسخدام مقاييس متكاملة, بدلا من خطأ في الجمع بين مئات الترانزستورات و المقاومات .

بطريقة مماثلة, يمكن للمبرمج الحديث الإستفادة من العمل المتراكم من سابقيه في إستخدام الوظائف الدالات المدمجة في أصناف وجود في البايثون . و الأفضل من ذلك, فإنه يمكن بسهولة إنشاء أصناف جديد لتغليف المكونات الرئيسية للتطبيق, و خاصة تلك التي تظهر في نسخ متعددة . و ذك أبسط و أسرع و أكثر أمانا من كتل تعليمات تتضاعف في هيئة برامج متماثلة متجانس, ضخمة أكثر و مفهوم أقل .

لدينا الأن مسودة المرسومة . أهم مكونات هذه اللعبة هي البنادق الصغيرة, سوف تكون قادرة على الرسم في مواقع مختلفة و في إتجاهات مختلفة, و نحن بحاجة على الأقل على نسختين منها .

بدلا من الإستفادة من الرسم قطعة قطعة و لو في أثناء المباراة, نحن مهتمون في بها ككائنات البرامج في حد ذاتها, مع خصائص متعددة و سلوك معين (ما كنا نريد قوله هنا هو أن تكون مجهزة مع آليات مختلفة, يمكننا فعل هذا برمجيا بإستخدام أساليب معينة) . Il est donc certainement judi cieux de leur consacrer une classe spécifique.

نماذج لصنف مدفع (Canon)

من خلال تعريف هذا الصنف, سوف نفوز في عدة جبهات . ليس فقط جمع كافة التعليمات البرمجية لتصميم و تشغيل المدفع في "كبسولة" واحدة,بعيدا عن بقية البرنامج, لكن بالإضافة إلى ذلك نقدم إماكنية تمثيل بأي عدد من المدافع في اللعبة, و الذي يتيح لنا المزيد من الفرص للتطوير .

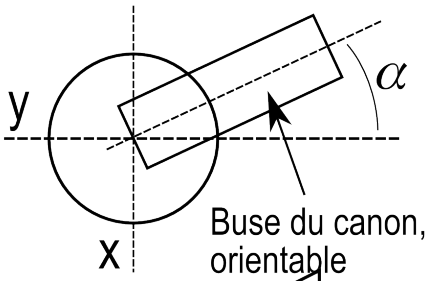
عندما يتم بناء و إختبار النموذج الأولي لصنف **Canon()**, سيكون من الممكن أيضا تحسينه من خلال منح مميزات إضافية دون تغيير(أو تقليل) واجهته, و هذا يعني على نحو ما "الإرشادات" : بمعرفة التعليمات اللازمة لإنشاء مثيل و إستخدامه في مختلف التطبيقات .

سوف ندخل الأن في صلب الموضوع .

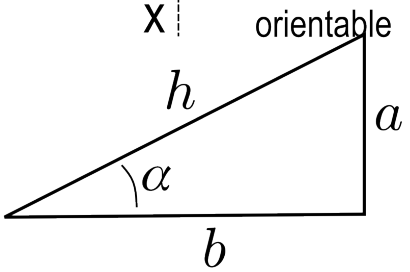
رسم المدفع يمكن تبسيطه إلى أقصى الحدود . شعرنا أننا يمكننا تلخيصها في دائرة إلى جانب مستطيل, و إنه يمكن أيضا أن تعتبر نفسها جزء بسيط خط مستقيم سميك بشكل خاص .

إذا تم تعبئتها جميعا بلون واحد (أسود مثلا), نحصل على نوع من القنابل ذات مصداقية بما فيه الكافية .

في الوسيطة التالية, نحن نفترض أن موقع المدفع هو ف الواقع موقع مركز الدائرة (الإحداثيات **x** و **y** في الرسم جانبا) . هذه النقطة الأساسية تدل أيضا على محور دوران فوهة المدفع, و الخط السميك في الذي يمثل الفوهة .



لإنهاء تصميمنا, فلم يتبقى سوى تحديد إحداثيات الطرف الأخر من الخط . هذه الإحداثيات يمكن حسابها بدون صعوبة كبيرة, تذكر مفهوميت أساسين للمثلث (الجيب و الجيب التمام) الذي يجب أن يكون على دراية بها بالتأكيد :



في المثلث لقائم, نسبة الجانب المقابل في زاوية و وتر المثلث هي خاصية معينة لهذا الزاوية تسمى جيب الزاوية . جيب الزاوية التمام هي النسبة بين الجانب المجاور للزاوية و الوتر .

Ainsi, dans le schéma ci-contre : $\sin\alpha=\frac{a}{h}$ et $\cos\alpha=\frac{b}{h}$.

لتمثيل طرف المدفع, على إفتراض أن نعرف طول و زاوية الإطلاق α, لذا يجب علينا رسم خط مستقيم سميك, من إحداثيات مركز الدائرة (x,y) إلى أخر نقطة إلى اليمين فوق, والمسافة الأفقية Δx تساوي l.cos α, و المسافة العمودية Δy تساوي l.sin α .

ملخص كل ما سبق, يجب علينا رسم مدفع في النقطة **x**,**y** تتكون ببساطة :

* رسم دائرة سوداء في وسط **x**,**y** .

* رسم خط أسود سميك من النقطة **x**,**y** إلى النقطة x + l.cos α, y + l.sin α.

يمكننا الأن أن نبدأ في النظر إلى مسودة البرمجة لصنف "Canon" . إننا لا نتحدث بعد عن برمجة اللعبة . نريد فقط معرفة ما إذا كان التحليل الذي قمنا به الأن "يأخذ طريقه" عن طريق إجراء نموذج أولي للوظيفة .

النموذج الأولي هو برنامج صغير لإختبار فكرة, التي نقترح دمجها في تطبيق أكبر . بسبب البساطة و الإبجاز, البايثون يفسح لك المجال يشكل جيد في تطوير النماذج الأولية, و العديد من المبرمجين يستخدمونها لبرمجة برامج مختلفة ثم سيقومون ربما بإعادة برمجتها بلغات أخرى "ثقيلة", على سبيل المثال لغة السي .

في نموذجنا الأولي, الصنف **Canon()** لا يملك سوى أسلوبين : المنشئ الذي يقوم بصنع عناصر الأساسية للرسم, و الأسلوب الذي يسمح بتعديل ضبط زاوية الإطلاق (الصندوق الخلفي للفوهة) . كما فعلنا في كثير من الأحيان في أمثلة أخرى, و سوف نقوك بتضمين بضعة أسطر برمجية في نهاية السكريبت من أجل إختبار على الفور الصنف :

```
1#    from tkinter import *
2#    from math import pi, sin, cos
3#
4#    class Canon(object):
5#      """Petit canon graphique"""
6#      def __init__(self, boss, x, y):
7#        self.boss = boss          #مرجع اللوحة
8#        self.x1, self.y1 = x, y    #محور دوران المدفع
9#        #: رسم فوهة البندقية, أفقيا للبدء
10#       self.lbu = 50             #عرض الفوهة
11#       self.x2, self.y2 = x + self.lbu, y
12#       self.buse = boss.create_line(self.x1, self.y1, self.x2, self.y2,
13#                                    width =10)
14#       #: رسم جسم المدفع أدناه
15#       r = 15                    #نصف قطر الدائرة
16#       boss.create_oval(x-r, y-r, x+r, y+r, fill='blue', width =3)
17#
```

```
18#     def orienter(self, angle):
19#        "choisir l'angle de tir du canon"
20#        #يتلقى سلسة نصية <angle> البرامتر.
21#        #: يجب تحويلها إلى عدد حقيقي, ثم إلى راديان
22#        self.angle = float(angle)*2*pi/360
23#        self.x2 = self.x1 + self.lbu*cos(self.angle)
24#        self.y2 = self.y1 - self.lbu*sin(self.angle)
25#        self.boss.coords(self.buse, self.x1, self.y1, self.x2, self.y2)
26#
27#  if __name__ == '__main__':
28#     #: كود لتجربة بإختصار الصنف Canon
29#     f = Tk()
30#     can = Canvas(f,width =250, height =250, bg ='ivory')
31#     can.pack(padx =10, pady =10)
32#     c1 = Canon(can, 50, 200)
33#
34#     s1 =Scale(f, label='hausse', from_=90, to=0, command=c1.orienter)
35#     s1.pack(side=LEFT, pady =5, padx =20)
36#     s1.set(25)                    #زاوية إطلاق النار الأولية
37#
38#     f.mainloop()
```

تعليقات

* السطر 6 : في قائمة البرامترات التي سنمررها للمنشئ عند التمثيل, نتوقع أن الإحداثيات **x** و **y**, التي تشير إلى مكان وجود المدفع في اللوحة, و لكن الإشارة إلى اللوحة نفسها (المتغير **boss**) . هذا المرجع هو لا غنى عنه : سوف يستخدم لإستدعاء أساليب اللوحة .

يمكن أن نشمل أيضا برامتر لإختيار زاوية الإطلاق الأولية, لكن بما أننا نعتزم تنفيذ طريقة محددة لحل هذا التوجه, سيكون من الحكمة إستخدامه عند الحاجة .

* الأسطر 7-8 : سوف نستخدم هذه المراجع في جميع الأساليب المختلفة التي نحن نطورها في الصنف . و لذلك يجب علينا أن نصنع سمات مثيل .

* الأسطر من 9 إلى 16 : سوف نصمم الفوهة أولا, ثم جسم المدفع . و هكذا, جزء من الفوهة الظاهرة تبقى مختبئة . و هذا يسمح لنا بتلوين جسم المدفع .

* الأسطر من 18 إلى 25 : هذه الأساليب سيتم إستدعائها مع البرامتر **angle**, و التي سيتم توفير درجتها(درجة الزاوية) (العد من الأفقي) . و هذا يتم عمله بمساعدة الويدجات مثل **Entry** أو **Scale**, و هي سوف تمررها على شكل سلسلة نصية, و نحن سنقوم بتحويلها أولا إلى عدد حقيقي قبل إستخدامه في حساباتنا (و قد وصفنها في الصفحة السابقة) .

* الأسطر من 27 إلى 38 : لإختبار صنف جديد, سوف نستخدم ويدجت **Scale** . لتعريف الموقع الأولي لمؤشره, ثم تعيين زاوية إرتفاع أولية من المدفع, جب علينا أن نستدعي الأسلوب **set()** (السطر 36) .

إضافة الأساليب للنموذج

نموذجنا هو وظيفي, و لكنها بدائية إلى حد كبير . و نحن الأن بحاجة إلى تطوير القدرة على إضافته إلى إطلاق النار .

سيتك التعامل مع هذه كرصاص التي من شأنها أن تكون دوائر بسيطة من فوهة المدفع مع سرعة أولية مماثلة للفوهة . لجعل تتبعها واقعي, يجب أن نتذكر بع العناصر الفيزيائية .

مثل الكائن يطرح و يترك نفسه ليتطور في الفضاء, إذا أهمنا الظاهرة الثانوية مثل مقاومة الهواء .

هذه المشكلة قد تبدو معقدة, و لكنها في الواقع بسيطة جدا : إن مجرد الإعتراف بأن الكرة تتحرك أفقيا و عموديا, و أن هتان الحركتان على الرغم من أنها متزامنة, فهي مستقة عن بعضها البعض .

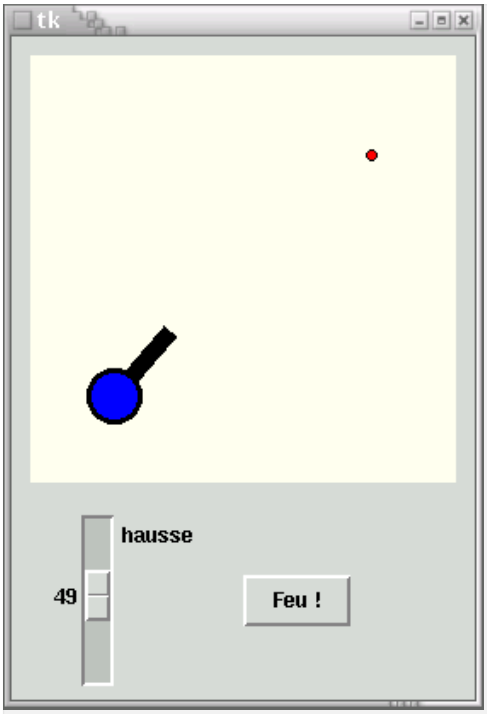
سوف نقوم الأن بإنشاء حلقة الرسوم المتحركو, التي من خلالها يمكنك إعادة حساب الإحداثيات **x** و **y** الجديدة و للكرة على فترات منتضمة من الزمن, مع العلم أن :

* الحركة أفقية موحدة . في كل تكرار, فقط زيادة التدريجية في إحداثية **x** للكرة, و أن نقوم دائما بإضافة نفس مكان **Δx** .

* التسارع بشكل موحد و الحركة الأفقية, هذا يعني ببساطة أن كل تكرار, يجب إضافة الإحداثيات **y** بإزاحة **Δy** نفسها مما يزيد تدريجيا, و دائما في نفس المقدار .

أنظر الأن إلى هذا السكريبت :

لنبدأ, يجب أن تضيف الأسطر التالية إلى نهاية أسلوب المنشئ . و سوف يتم إستخدامه لصنع الكائن "القذيفة" . و إعداد متغير المثيل من شأنه أن يسخدم رسوم متحركة . يتم صنع القذيفة مع الحد الأدنى للأبعاد (دائرة بكسل واحدة) و أن تظل تقريبا غير مرئية :




```
#: رسم قذيفة (تم إختصارها إلى نقطة واحدة, قبل الرسوم المتحركة
self.obus =boss.create_oval(x, y, x, y, fill='red')
self.anim =False            #التبديل إلى الرسوم المتحركة
#: إيجاد عرض و إرتفاع اللوحة
self.xMax =int(boss.cget('width'))
self.yMax =int(boss.cget('height'))
```

و السطران الأخيران يتخدمان الأسلوب **cget()** للويدجت "السيد" (اللوحة, هنا), من أجل العثور بعض من خصائصه . نريد أن يكون صنفنا **Canon** عام, و هذا يعني قابل لإعادة الإستخدام في أي سياق, و نحن بالتالي لا يمكننا الإعتماد على الأبعاد المحددة للوحة الذي سيتم إستخدامه للمدفع .

> يقوم *tkinter* بإرجاع هذه القيم كسلاسل نصية . و لذلك من الضروري تحويلها إلى نوع رقمي إذا كنا نريد إستخدامها في عملية حسابية ..

ثم نحن بحاجة إلى إضافة أسلوبين جديدين : واحد لإطلاق النار, و الأخر لإدارة رسوم المتحركة للكرة بعد إطلاقها :

```
1#        def feu(self):
2#          "déclencher le tir d'un obus"
3#          if not self.anim:
4#             self.anim =True
5#             #: مكان إطلاق القذيفة (الفوهة)
6#             self.boss.coords(self.obus, self.x2 -3, self.y2 -3,
7#                              self.x2 +3, self.y2 +3)
8#             v =15          #السرعة الأولية
9#             #: المكونات الأفقية و العمودية لهذه السرعة
10#            self.vy = -v *sin(self.angle)
11#            self.vx = v *cos(self.angle)
12#            self.animer_obus()
13#
14#      def animer_obus(self):
15#         "animation de l'obus (trajectoire balistique)"
16#         if self.anim:
17#            self.boss.move(self.obus, int(self.vx), int(self.vy))
18#            c = tuple(self.boss.coords(self.obus))    #coord. résultantes
19#            xo, yo = c[0] +3, c[1] +3   #coord. du centre de l'obus
20#            if yo > self.yMax or xo > self.xMax:
21#               self.anim =False       #إيقاف التحريك
22#            self.vy += .5
23#            self.boss.after(30, self.animer_obus)
```

تعليقات

* السطور من 1 إلى 4 : هذا الأسلوب سيتم إستدعائه بالضغط على الزر . فإنه سوف يتسبي في تحريك القذيفة, و تعيين قيمة "الحقيقية" إلى "رسوم متحركة" (المتغير **self.anim** : أنظر أدناه) ؟ و مع ذلك, يجب علينا أن نضمن مدة لهذه الرسوم المتحركة, ضغطة جديدة على زر لا يمكنها تنشيط حلقت أخرى للتحريك الرسوم . و هذا هو دور السطر الثالث : كتلة التعليملت الت تعمل إذا كان المتغير **self.anim** قيمته "faux", مما يعني أن الرسوم المتحركة لن تبدأ بعد .

* الأسطر من 5 إلى 7 : لوحة tkinter لديه أسلوبان لنقل الكائنات الرسومية :

- الأسلوب **coords()** (المستخدم في السطر السادس) ينفذ موضع المطبق, و مع ذلك يجب أن يتم توفير جميع المعلومات للكائن (كما لو أننا أعدنا رسمه) .

- الأسلوب **move()** (المستخدم في وقت لاحق, السطر 17), يؤدي إلى النزوح نسبيا, و هي تستخدم مع برامترين فقط, هما المكونان الأفقي و العمودي للحركة المطلوبة .

* الأسطر من 8 إلى 12 : يتم إختيار السرعة الأولية للطلقة في السطر 9 . كما قمنا نحن بشرحه في الصفحة السابقة, حركة الكرة هي نتيجة لحركة أفقية و لحركة عمودية . نحن نعرف قيمة السرعة الأولية و الميل (و هذا يعني زاوية إطلاق النار) . لتحديد المكونات الأفقية و العمودية لهذه السرعة,

نحن سنقوم فقط بإستخدام العلاقات المثلثية المماثلة لتلم التي إستخدمت بالفعل في ريم فوهة المدفع . توقيع - المستخدم في السطر 10 يات من إحداثيات العمودية من الأعلى إلى الأسفل .

السطر 12 يفعل (ينشط) الحركة نفسها .

* الأسطر من 14 إلى 23 : هذا الإجراء يقوم بإستدعاء نقسه كل 30 ميلي ثانية عن طريق الأسلوب **after()** الذي تم إستدعائها في السطر 23 . و هذه تكمل لطالما المتغير **self.anim** (محرك الروسم المتحركة الخاص بنا) يبقى "صحيحا - vraie",و حالة تتغير عند إحداثيات القذيفة تخرج من إختبار حدود (إختبار في السطر 20) .

* الأسطر 18-19 : لمعرفة إحداثيات بعد كل إزاحة, نقوم بإستدعاء في كل مرة الأسلوب **()** **coords** للوحة : يستخدم هذه المرة مع مرجع الكائن الرسومي كبرامتر واحد, سوف تقوم بإرجاع أربعة إحداثيات في كائن itérable التي يمكن تحويلها إلى قائمة أو إلى نفق بمساعدة الدالات المدمجة **list()** و **tuple()** .

8 السطور 17 - 22 : الإحداثيات الأفقية للطلقة تزيد دائما بمقدار(حركة موحدة), في حين أن زيادات التنسيق العمودي من قبل المقدار الذي هو في حد ذاته في كل مرة في السطر 24 (الحركة الموحدة بشكل تسارعي) . و النتيجة هي مسار مكافئ .

> المعامل += سيمح بزيادة إلى المتغير :
>
> لاحظ أن إستخدام هذا المعامل الخاص هو الأكثر كفائة من . a = a +3 تعادل a += 3 و إعادة التخصيص التي إستخدمناها حتى الأن .
>
> لتمثيل False و True بداية من الإصدار 2.3, يقوم بالبايثون بتهيئة متغيرين تلقائيا إسمهم الصحيح و الخطأ تعبير (لاحظ أن الإسمين يبدأن بحرف كبير) . كما فعلنا في السكريبت أعلاه, يمكنك إستخدام هذه المتغيرات في تعابير شرطية لزيادة سهولة قراءة كودك . فإذا كنت تفضل, يمكنك إستخدام القيم الرقمية' كما فعلنا سابقا (أنظر إلى "صحة\خطأ تعبير" صفحة) .

و أخيرا, فإنه لا يزال إضافة زر الزناد في النافذة الرئيسية . سطر مثل هذا (التي سوف يتم وضعها في تعليمات الإختبار البرمجية) تقوم بشكل جيد :

```
Button(f, text='Feu !', command =c1.feu).pack(side=LEFT)
```

تطوير تطبيق

الأن لدينا صنف الكائنات "canon" assez bien dégrossie و نرى أن تطوير التطبيق نفسه . و منذ ان قررنا إستخدام منهجية البرمجة الشيئية, يجب علينا تطوير هذا التطبيق على أنه مجموعة من الكائنات التي تتفاعل من خلال أساليبها .

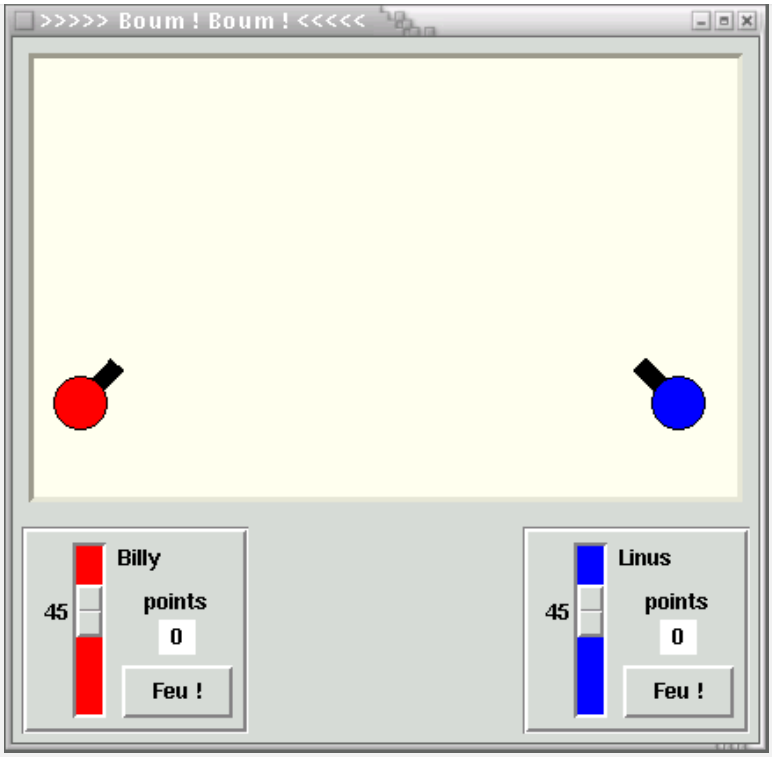
العديد من هذه الكائنات تأتي من أصناف قائمة, بطبيعة الحال : لوحة, أزرار ...إلخ . لكن رأينا فب الصفحات السابقة لدينا إهتمام بتجميع مجموعات محددة جيدا من هذه الكائنات الأساليب في أصناف جديدة, كل مرة يمكننا تحديد هذه المجموعات لميزة معينة . على سبيل المثال هذه مجموعة من الدوائر و الخطوط المتحركة, من إستدعاء "**canon** - المدفع" .

هل يمكننا أن نجعل لمشروعنا الأولي مكونات أخرى التي ينبغي أن يتم تغليفها في أصناف جديدة ؟ بالتأكيد نعم, يوجد على سبيل المثال le pupitre de contrôle التي يمكننا أن نربطها مع كل مدفع : يمكننا جمع ما يصل إلى الأعلى (زاوية إطلاق النار), زر إطلاق النار, نقاط المتحصل عليها, و مؤشرات أخرة ربما لاتزل مثل إسم اللاعب . و من المثير للإهتمام بشكل خاص لصنف معينة, و نحن نعلم منذ البداية أننا نأخذ مثيلين .

و هنالك أيضا التطبيق نفسه, بطبيعة الحال . عن طريق التغليف في صنف, و سوف نبذل هدفنا الرئيسي, الذي يؤدي لكل الأخرين .

يرجى الأن تحليل السكريبت أدناه . سوف تجد الصنف **Canon()** مطور : أضفنا بعض السمات الإضافية و ثلاثة أساليب إضافية, من أجل إدارة حركة المدفع, فضلا عن الأهداف .

الصنف **Application()** يقوم بإستبدال كود ختبار النماذج أعلاه . نحن وسف نقوم بتمثيل كائنين **Canon()**, و كائنين لصنف جديد **Pupitre()**, الذي نحن وضعناه في قاموس التنبؤ بالتطورات المستقبلية (بمكننا أن نتصور فعلا زيادة عدد المدافع و بالتالي عدد مناضد) . اللعب الأن وظيفية : المدافع تتحرك بعد كل حلقة, و يتم تسجبل الأهداف .




```
1#      from tkinter import *
2#      from math import sin, cos, pi
3#      from random import randrange
4#
5#      class Canon(object):
6#        """Petit canon graphique"""
7#        def __init__(self, boss, id, x, y, sens, coul):
8#          self.boss = boss          #مرجع اللوحة
9#          self.appli = boss.master    #مرجع نافذة التطبيق
10#         self.id = id              #تعريف مدفع (سلسلة)
11#         self.coul = coul          #اللون المرتبط بالمدفع
12#         self.x1, self.y1 = x, y     #محور دوران المدفع
13#         self.sens = sens          #إتجاه الإطلاق (-1:يسار, +1:يمين)
14#         self.lbu = 30             #طول الفوهة
15#         self.angle = 0            #الزيادة الإفتراضية (زاوية الإطلاق)
16#         #إيجاد عرض و إرتفاع اللوحة
17#         self.xMax = int(boss.cget('width'))
18#         self.yMax = int(boss.cget('height'))
19#         #: رسم فوهة المدفع (أفقي)
20#         self.x2, self.y2 = x + self.lbu * sens, y
21#         self.buse = boss.create_line(self.x1, self.y1,
22#                                      self.x2, self.y2, width =10)
23#         #: رسم جسم المدفع(دائرة ملونة)
24#         self.rc = 15              #نصف قطر الدائرة
25#         self.corps = boss.create_oval(x -self.rc, y -self.rc, x +self.rc,
26#                                       y +self.rc, fill =coul)
27#         #«: رسم قذيفة مخفية (نقطة خارج اللوحة)
28#         self.obus = boss.create_oval(-10, -10, -10, -10, fill='red')
29#         self.anim = False         #مؤشرات الرسوم المتحركة
30#         self.explo = False        #  و الإنفجار
31#
```

```
32#     def orienter(self, angle):
33#       "régler la hausse du canon"
34#       #البرامتر <angle> يتلقى كسلسلة
35#       #: يجب تحويلها إلى عدد حقيقي, ثم إلى راديان
36#       self.angle = float(angle)*pi/180
37#       #الذي يفضل مع أعداد صحيحة coords إستخدم الأسلوب :
38#       self.x2 = int(self.x1 + self.lbu * cos(self.angle) * self.sens)
39#       self.y2 = int(self.y1 - self.lbu * sin(self.angle))
40#       self.boss.coords(self.buse, self.x1, self.y1, self.x2, self.y2)
41#
42#     def deplacer(self, x, y):
43#       "amener le canon dans une nouvelle position x, y"
44#       dx, dy = x -self.x1, y -self.y1    #قيمة الإزاحة
45#       self.boss.move(self.buse, dx, dy)
46#       self.boss.move(self.corps, dx, dy)
47#       self.x1 += dx
48#       self.y1 += dy
49#       self.x2 += dx
50#       self.y2 += dy
51#
52#     def feu(self):
53#       "tir d'un obus - seulement si le précédent a fini son vol"
54#       if not (self.anim or self.explo):
55#         self.anim =True
56#         #: جلب وصف جميع المدافع الحالية
57#         self.guns = self.appli.dictionnaireCanons()
58#         #: موقع بدء القذيفة (فوهة المدفع)
59#         self.boss.coords(self.obus, self.x2 -3, self.y2 -3,
60#                          self.x2 +3, self.y2 +3)
61#         v = 17          #السرعة الأولية
62#         #: المكونات الأفقية و العمودية لهذه السرعة
63#         self.vy = -v *sin(self.angle)
64#         self.vx = v *cos(self.angle) *self.sens
65#         self.animer_obus()
66#         return True    #=> إشارة إلى أن القذيفة أطلقت
67#       else:
68#         return False    #=> لم يتم إطلاق القذيفة
69#
70#     def animer_obus(self):
71#       "animer l'obus (trajectoire balistique)"
72#       if self.anim:
73#         self.boss.move(self.obus, int(self.vx), int(self.vy))
74#         c = tuple(self.boss.coords(self.obus))    #coord. résultantes
75#         xo, yo = c[0] +3, c[1] +3     #coord. du centre de l'obus
76#         self.test_obstacle(xo, yo)    #a-t-on atteint un obstacle ?
77#         self.vy += .4              #التسارع العمودي
78#         self.boss.after(20, self.animer_obus)
79#       else:
80#         #: الرسوم المتحركة إنتهت – إخفاء القذائف و نقل المدافع
81#         self.fin_animation()
82#
83#     def test_obstacle(self, xo, yo):
84#       "évaluer si l'obus a atteint une cible ou les limites du jeu"
85#       if yo >self.yMax or xo <0 or xo >self.xMax:
86#         self.anim =False
87#         return
88#       #analyser le dictionnaire des canons pour voir si les coord.
89#       #de l'un d'entre eux sont proches de celles de l'obus :
90#       for id in self.guns:          #id = المفتاح في القاموس
91#         gun = self.guns[id]        #القيم المطابقة
```

```
92#          if xo < gun.x1 +self.rc and xo > gun.x1 -self.rc \
93#          and yo < gun.y1 +self.rc and yo > gun.y1 -self.rc :
94#            self.anim =False
95#            #: رسم إنفجار القذيفة (دائرة صفراء)
96#            self.explo = self.boss.create_oval(xo -12, yo -12,
97#                    xo +12, yo +12, fill ='yellow', width =0)
98#            self.hit =id      #مرجع إصابة الهدف
99#            self.boss.after(150, self.fin_explosion)
100#            break
101#
102#    def fin_explosion(self):
103#      "effacer l'explosion ; réinitaliser l'obus ; gérer le score"
104#      self.boss.delete(self.explo)   #مسح الإنفجار
105#      self.explo =False            #إذن لإطلاق نار جديد
106#      #: إشارة نجاح إلى النافذة الرئيسية
107#      self.appli.goal(self.id, self.hit)
108#
109#    def fin_animation(self):
110#      "actions à accomplir lorsque l'obus a terminé sa trajectoire"
111#      self.appli.disperser()        #نقل المدافع
112#      #: إخفاء القذيفة (عن طريق إرسالها خارج اللوحة)
113#      self.boss.coords(self.obus, -10, -10, -10, -10)
114#
115#  class Pupitre(Frame):
116#    """Pupitre de pointage associé à un canon"""
117#    def __init__(self, boss, canon):
118#      Frame.__init__(self, bd =3, relief =GROOVE)
119#      self.score =0
120#      self.appli =boss            #مرجع التطبيق
121#      self.canon =canon           #مرجع المدفع المرتبط
122#      #: نظام تحكم في زاوية الإطلاق
123#      self.regl =Scale(self, from_ =85, to =-15, troughcolor=canon.coul,
124#                 command =self.orienter)
125#      self.regl.set(45)           #الزاوية الأولية للإطلاق
126#      self.regl.pack(side =LEFT)
127#      #: علامة تعريف المدفع
128#      Label(self, text =canon.id).pack(side =TOP, anchor =W, pady =5)
129#      #: زر الإطلاق
130#      self.bTir =Button(self, text ='Feu !', command =self.tirer)
131#      self.bTir.pack(side =BOTTOM, padx =5, pady =5)
132#      Label(self, text ="points").pack()
133#      self.points =Label(self, text=' 0 ', bg ='white')
134#      self.points.pack()
135#      #وضع على يمين أو اليسار إعتمادا على إتجاه المدفع
136#      if canon.sens == -1:
137#        self.pack(padx =5, pady =5, side =RIGHT)
138#      else:
139#        self.pack(padx =5, pady =5, side =LEFT)
140#
141#    def tirer(self):
142#      "déclencher le tir du canon associé"
143#      self.canon.feu()
144#
145#    def orienter(self, angle):
146#      "ajuster la hausse du canon associé"
147#      self.canon.orienter(angle)
148#
149#    def attribuerPoint(self, p):
150#      "incrémenter ou décrémenter le score, de <p> points"
151#      self.score += p
```

```
152#        self.points.config(text = ' %s ' % self.score)
153#
154#  class Application(Frame):
155#     '''Fenêtre principale de l'application'''
156#     def __init__(self):
157#        Frame.__init__(self)
158#        self.master.title('>>>>> Boum ! Boum ! <<<<<')
159#        self.pack()
160#        self.jeu = Canvas(self, width =400, height =250, bg ='ivory',
161#                    bd =3, relief =SUNKEN)
162#        self.jeu.pack(padx =8, pady =8, side =TOP)
163#
164#        self.guns ={}          #قاموس المدفع الموجودة
165#        self.pupi ={}          #قاموس طاولات اللعب
166#        #تمثيل كائني مدفع (+1, -1 = معاكس) :
167#        self.guns["Billy"] = Canon(self.jeu, "Billy", 30, 200, 1, "red")
168#        self.guns["Linus"] = Canon(self.jeu, "Linus", 370,200,-1, "blue")
169#        #تمثيل طاولتي تسجيل مرتبطة بهذه المدافع :
170#        self.pupi["Billy"] = Pupitre(self, self.guns["Billy"])
171#        self.pupi["Linus"] = Pupitre(self, self.guns["Linus"])
172#
173#     def disperser(self):
174#        "déplacer aléatoirement les canons"
175#        for id in self.guns:
176#           gun =self.guns[id]
177#           #وضع على يمين أو اليسار إعتمادا على إتجاه المدفع
178#           if gun.sens == -1 :
179#              x = randrange(320,380)
180#           else:
181#              x = randrange(20,80)
182#           #: النزوح
183#           gun.deplacer(x, randrange(150,240))
184#
185#     def goal(self, i, j):
186#        "le canon <i> signale qu'il a atteint l'adversaire <j>"
187#        if i != j:
188#           self.pupi[i].attribuerPoint(1)
189#        else:
190#           self.pupi[i].attribuerPoint(-1)
191#
192#     def dictionnaireCanons(self):
193#        "renvoyer le dictionnaire décrivant les canons présents"
194#        return self.guns
195#
196#  if __name__ =='__main__':
197#     Application().mainloop()
```

تعليقات

* السطر 7 : و بالمقارنة مع النموذج, تم إضافة 3 برامترات إلى الأسلوب المنشئ . البرامتر id يسمح لما بتعريف كل مثيل في الصنف **Canon()** بمساعدة أي إسم . و البرامتر **sens** تشير إلى إتجاه البندقية فإذا كانت على اليمين (**sens = 1**) و إذا كانت على اليسار (**sens = -1**) . البرامتر **coul** الخاص بلون المرتبط بالمدفع .

* السطر 9 : جميع ويدجات tkinter يمكنها الوصول إلى سمة **master** التي تحتوي على مرجع ودجتهم "السيد" المحتمل (الحاوي) . هذا المرجع هو بالنسبة لنا التطبيق الرئيسي . لقد قمنا بتنفيذ تقنية مماثلة لمرجع اللوحة, ذلك بإستخدام سمة **boss** .

* الأسطر من 42 إلى 50 : هذا الأسلوب يسمح لنا بوضع المدفع في مكان جديد . إستخدمه لإعادة وضع المدفع بشكل عشوائي بعد كل طلقة, مما يزيد من الإهتمام في اللعبة .

* الأسطر 56 و 57 : نحن نحاول بناء صنف المدفع بحيث يمكن إستخدامه في مشاريع أكبر, تشمل على أي عدد من أصناف المدافع التي يمكنها الظهور و الإختفاء في وسط الفتال . في هذا المنظور, لابد أن لدينا وصف لجميع المدافع الحالية, قبل كل طلقة, بحيث يمكن تحديد ما إذا كان قد تم إصابة الهدف أو لا . و هذا الوصف يتم صنعه بواسطة التطبيق الرئيسي, في قاموس, و التي يمكن طلب نسخة من خلال الأسلوب **dictionnaireCanons()** .

* الأسطر من 66 إلى 68 : في هذا المنظور العام نفسه, قد يكون من المفيد أن أبلغ أن البرنامج الإستدعاء أطلق النار فعلا أو لا .

* السطر 76 : يتم التعامل مع رسوم المتحركة للطلقة (أو القذيفة) من خلال أسلوبين متكاملة . لتوضيح الكود, وضعنا في أسلوب منفصل تعليمات تحدد طريقة ما إذا تم التوصل إلى الهدف (الأسلوب **test_obstacle()**) .

* الأسطر من 79 إلى 81 : لقد رأينا سابقا أن الرسوم المتحركة للقذيفة تتوقف عند تعيين القيمة "سالب - fausse" للمتغير **self.anim** . الأسلوب **animer_obus()** يوقف الحلقة قم ينفذ المود في السطر 81 .

* الأسطر 83 إلى 100 : هذا الأسلوب يقيم ما إذا كانت الإحداثيات الحالية ببطلقة خرجة من حدود النافذة, أو إنها تقترب من قذيفة أخرى . في كلا الحالتين, يتم تفعيل مبدل الرسوم المتحركة, لكن في الحالة الثانية, نرسم "إنفجار" أصفر, و يتم تخزين مرجع المدفع . ينم إستدعاء أسلوب المرفق **fin_explosion()** بعد وقت قصر لإنهاء العمل, و هذا معناه حذف دائرة الإنفجار و إرسال رسالة إلى نافذة الرئيسية للإشارة إلى أنه ضرب .

* الأسطر 115 إلى 152 : ستم تعريف الصنف **Pupitre()** في ويدجت جديد مشتق من الصنف **Frame()**, و هي تقنية أصبحت الأن مألوفة و هذا الويدجت الجديد يجمع أوامر الإرتفاع و الإطلاق النار , ثم يعرض النقاط المرتبطة بالمدفع المحددة جيدا . و يتم توفير مراسلات بصرية بين الإثنين من خلال إعتماد لون مشترط . الأساليب **tirer()** و **orienter()** تتواصل مع الكائن **Canon()** المرتبط بها, عن طريق أساليبها .

* الأسطر من 154 إلى 171 : نافذة التطبيق هي أيضا ويدجت مشتق من **Frame()** . منشئه يمثل مدفعين و يشير إلى مواقع الإطلاق, و تم وضع هذه الكائنات في قاموسين **self.guns** و **self.pupi** . هذا يسمح بتنفيذ معالجات مختلفة على منهجية كل واحد منهم (على سبيل المثال الأسلوب التالي) . بالقيام بذلك, فإنه تحتفظ أيضا إمكانية زيادة عدد المدافع إذا لزم الأمر, في تطويرات لاحقة من البرنامج .

* الأسطر من 173 إلى 183 : يتم إستدعاء هذه الأسليب بعد كل طلقة لنقل المدفعن بشكل عشوائي, مما يزيد من صعوبة اللعبة .

تطويرات إضافية

كما موضح أعلاه, برنامجنا هو أكثر أو أقل من الموصفات الأصلية, لكن من الواضح أننا نتمكن من الإستمرار في تحسينه .

أ) ينبغي لنا أن نضع مثال أفضل . ما معنا هذا . في شكله الحالي, لعبتنا لديها حجم لوحة محدد سابقا (400 × 250 بيكسل, أنظر للسطر 161) . فإذا أردنا تغيير هذه القيم, فإننا نحتاج أيضا إلى ضمان تعديل أسطر أخرى من السكريبت حيث الأبعاد المعنية (على سبيل المثال هذه الأسطر 168-169 أو 179-184) . قد تصبح هذه الأسطر مترابطة إذا أضفنا العديد من الميزات الأخرى . سيكون من الخكمة تغيير حجم اللوحة بمساعدة متغيرات, قيمته تم تعريفها في مكان واحد . هذه المتغيرات سيتم إستخدامهم في جميع الأسطر التعليمات التي تشترك فيها أبعاد اللوحة .

لقد قمنا بالفعل بجزء من هذا العمل : في الصنف **Canon()**, في الحقيقة أبعاد اللوحة سيتم إستردادها بإستخدام أسلوب محدد مسبقا (أنظر للسطور 17-18), و وضعها في سمات المثيل التي يمكن إستخدامها في أي في صنف .

ب) بعد كل طلقة, سوف نقوم بنقل عشوائيا المدافع, و إعادة تعريف إحداثياتها . ربما يكون أكثر واقعيا نسبيا و السبب الحقيقي في النزوح, بدلا من إعادة تعريف عشوائيا المواقع المطلقة . للقيام بذلك, يكفي أن تعيد عمل الأسلوب **deplacer()** للصنف **Canon()** . في الحقيقة, سيكون أكثر إثارة للإهتمام جعل هذه الطريقة يمكن أن تنتج ذلك, فضلا عن نزوح تحديد المواقع النسبية المطبقة, بناءا على القيمة الممررة كبرامتر .

ج) ينبغي تحسين نظام التحكم في إطلاق النار : لأن لدينا فقط نظام واحد و هو الفأرة, إسل اللاعبين بالتناوب, و ليس لدينا الية لإجبارهم على القيام بذلك . لذا إعتمد على النهج الذي يوفر أوامر الإرتفاع و إطلاق النار حتى بإستخدام بعض مفاتيح لوحة المفاتيح التي يجب أن تختلف بين كلا اللاعبين .

د) و لكن الأكثر إثارة للإهتمام في تطوير برنامجنا لجعله برنامج يعمل على الشبكة . اللعبة سيتم تثبيتها على مجموعة من الأجهزة المتعددة التي تتواصل مع كل لاعب للتحكم على مدفع واحد . سيكون أكثر جاذبية السماح بتنفيذ أكثر من مدفقعي, للسماح بالقتال التي تشمل على الكثير من اللاعبين .

و هذا النوع من التطوير, يتطلب منا إتقان مجالين من المجالات التي هي خارج إطار الدورة :

* تقنية sockets, التي تسمح بالإتصال بين جازي حاسوب .

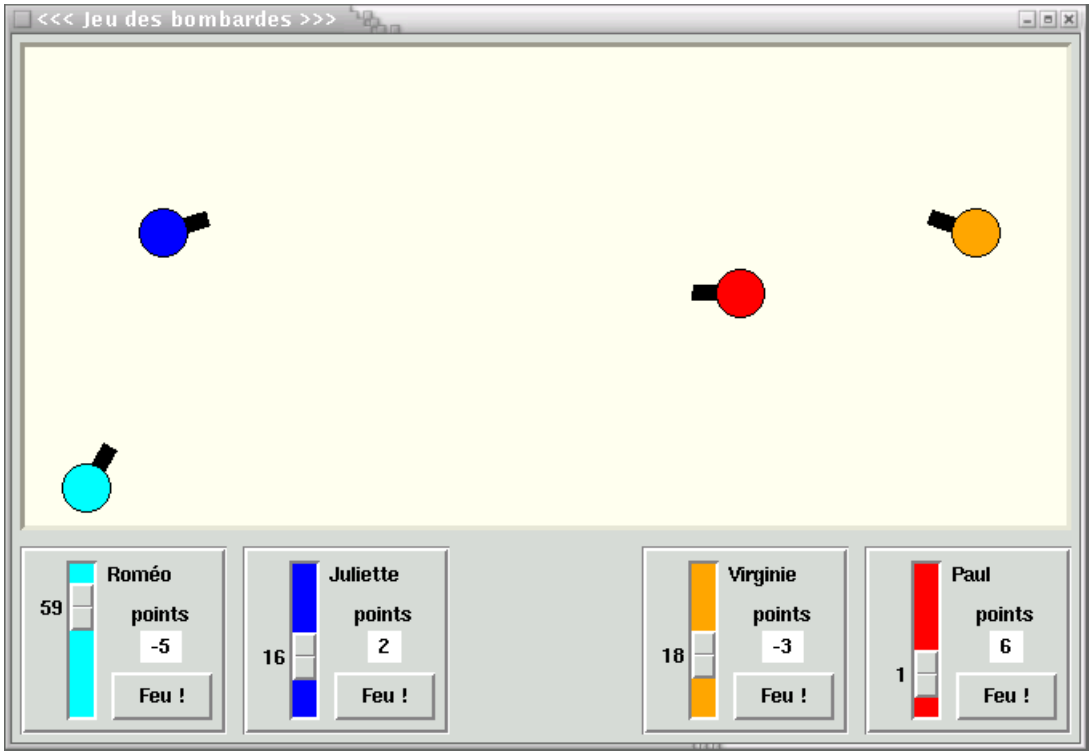
* تقنية threads, التي تسمح لبرنامج واحد بتنفيذ عدة مهما في وقت واحد (و هذا ضروري, إذا كنت تريد بناء تطبيق يمكنه التواصل مع عدة شركاء) .

هذه المواد ليست جزء من كائنات التي وضعناها لهذه الدورة, و التي تشمل معالجة وحده فصلا كاملا . نحن لا نناقش هذه المسألة هنا . المهتمين بهذا الموضوع : هذا الفصل موجودا, و لكنه كتكملة لنهاية الكتاب (الفصل 19) : و سوف تجد نسخة من لعبتنا تعمل على الشبكة .

و في الوقت نفسه, لا تزال ترى كيف يمكننا إحراز المزيد من التقدم في تحقيق بعض التحسينات في مشروعنا التي من شأتها أن تجعل اللعبة لأربعة لاعبين . و سوف نقوم أيضا بوضع برمجتنا مقسمة بشكل جيد, بحيث أن أساليب الأصنافنا قابلة لإعادة الإستخدام . و سنرى أيضا كيف يمكننا تغيير

الطريقة, دون المساس (التغيير) بالتعليمات البرمجية الموجودة, و سنقومب بهذا عن طريق الميراث لصنع أصناف جديدة من تلك المكتوبة .

نبدأ بحفظ عملنا السابق في ملف, (و الذي نفترض أن له بقية) و إسم الملف هو : **canon03.py** .



لدينا الأن وحدة بايثون حقيقية, التي يمكننا إستدعائها في سكريبت جدبد بمساعدة تعليمة واحدة (في سطر واحد) . من خلال إستغلال هذه التقنية, سوف نواصل تحسين طلبنا, عن طريق الحفاظ على أعيننا الجديد :

```
1#      from tkinter import *
2#      from math import sin, cos, pi
3#      from random import randrange
4#      import canon03
5#
6#      class Canon(canon03.Canon):
7#        """Canon amélioré"""
8#        def __init__(self, boss, id, x, y, sens, coul):
9#          canon03.Canon.__init__(self, boss, id, x, y, sens, coul)
10#
11#       def deplacer(self, x, y, rel =False):
12#         "déplacement, relatif si <rel> est vrai, absolu si <rel> est faux"
13#         if rel:
14#           dx, dy = x, y
15#         else:
```

```
16#         dx, dy = x -self.x1, y -self.y1
17#       #الحدود الأفقية :
18#       if self.sens ==1:
19#         xa, xb = 20, int(self.xMax *.33)
20#       else:
21#         xa, xb = int(self.xMax *.66), self.xMax -20
22#       #: لا تتحرك إلا داخل الحدود
23#       if self.x1 +dx < xa:
24#         dx = xa -self.x1
25#       elif self.x1 +dx > xb:
26#         dx = xb -self.x1
27#       #: الحدود العمودية
28#       ya, yb = int(self.yMax *.4), self.yMax -20
29#       #: لا تتحرك إلا داخل الحدود
30#       if self.y1 +dy < ya:
31#         dy = ya -self.y1
32#       elif self.y1 +dy > yb:
33#         dy = yb -self.y1
34#       #: نحريك فوهة و جسم المدفع
35#       self.boss.move(self.buse, dx, dy)
36#       self.boss.move(self.corps, dx, dy)
37#       # renvoyer les nouvelles coord. au programme appelant :
38#       self.x1 += dx
39#       self.y1 += dy
40#       self.x2 += dx
41#       self.y2 += dy
42#       return self.x1, self.y1
43#
44#     def fin_animation(self):
45#       "actions à accomplir lorsque l'obus a terminé sa trajectoire"
46#       #: تحريك المدفع الذي سيطلق النار
47#       self.appli.depl_aleat_canon(self.id)
48#       #: إخفاء القذيفة (عن طريق إرسالها خارج اللوحة)
49#       self.boss.coords(self.obus, -10, -10, -10, -10)
50#
51#     def effacer(self):
52#       "faire disparaître le canon du canevas"
53#       self.boss.delete(self.buse)
54#       self.boss.delete(self.corps)
55#       self.boss.delete(self.obus)
56#
57#   class AppBombardes(Frame):
58#     '''Fenêtre principale de l'application'''
59#     def __init__(self, larg_c, haut_c):
60#       Frame.__init__(self)
61#       self.pack()
62#       self.xm, self.ym = larg_c, haut_c
63#       self.jeu = Canvas(self, width =self.xm, height =self.ym,
64#                         bg ='ivory', bd =3, relief =SUNKEN)
65#       self.jeu.pack(padx =4, pady =4, side =TOP)
66#
67#       self.guns ={}          #قاموس المدافع الموجودة
68#       self.pupi ={}          #قاموس الطاولات الموجودة
69#       self.specificites()    #كائنات مختلفة في أصناف مشتقة
70#
71#     def specificites(self):
72#       "instanciation des canons et des pupitres de pointage"
73#       self.master.title('<<< Jeu des bombardes >>>')
74#       id_list =[("Paul","red"),("Roméo","cyan"),
75#                 ("Virginie","orange"),("Juliette","blue")]
```

```
76#         s = False
77#         for id, coul in id_list:
78#           if s:
79#             sens =1
80#           else:
81#             sens =-1
82#           x, y = self.coord_aleat(sens)
83#           self.guns[id] = Canon(self.jeu, id, x, y, sens, coul)
84#           self.pupi[id] = canon03.Pupitre(self, self.guns[id])
85#           s = not s        #تغيير الجانب في كل تكرار
86#
87#       def depl_aleat_canon(self, id):
88#         "déplacer aléatoirement le canon <id>"
89#         gun =self.guns[id]
90#         dx, dy = randrange(-60, 61), randrange(-60, 61)
91#         #: تحريك (مع تحديث الإحداثيات الجديدة)
92#         x, y = gun.deplacer(dx, dy, True)
93#         return x, y
94#
95#       def coord_aleat(self, s):
96#         "coordonnées aléatoires, à gauche (s =1) ou à droite (s =-1)"
97#         y =randrange(int(self.ym /2), self.ym -20)
98#         if s == -1:
99#           x =randrange(int(self.xm *.7), self.xm -20)
100#        else:
101#          x =randrange(20, int(self.xm *.3))
102#        return x, y
103#
104#      def goal(self, i, j):
105#        "le canon n°i signale qu'il a atteint l'adversaire n°j"
106#        # de quel camp font-ils partie chacun ?
107#        ti, tj = self.guns[i].sens, self.guns[j].sens
108#        if ti != tj :          #كانوا في إتجاهين متعاكسين
109#          p = 1               #يربح نقطة واحدة
110#        else:                 #إذا كانوا في نفس الإتجاه
111#          p = -2              #نضرب حليف !!!
112#        self.pupi[i].attribuerPoint(p)
113#        #الذي أصيب سوف يخسر نقطة على أي حال
114#        self.pupi[j].attribuerPoint(-1)
115#
116#      def dictionnaireCanons(self):
117#        "renvoyer le dictionnaire décrivant les canons présents"
118#        return self.guns
119#
120#  if __name__ =='__main__':
121#      AppBombardes(650,300).mainloop()
```

تعليقات

السطر 6 : شكل الإستدعاء المستخدم في السطر 4 يسمح لنا بإعادة تعريف صنف جديد و هو Canon() المشتق من سابقه, مع الإحتفاظ بنفس الإسم . و بهذه الطريقة, ينبغي أن أجزاء التعليمات البرمجية التي تستخدم هذا النصف لا يمكن تغييرها (لا يمكنك إذا إستخدمت على سبيل المثال :

from canon03 import *

* السطور من 11 إلى 16 : الأسلوب المعرف هنا الذي يحمل نفس الإسم هو أسلوب للسنف الأصل . و سيتم إستبداله في صنف جديد (يمكننا القول أن الأسلوب **deplacer()** مثقل) عند تنفيذ هذا النوع من التغيير, فإنه يهدف بشكل عام للتأكد من أن الأسلوب الجدبد يوقم بنفس العمل كما في السابق عندما يتم إستدعائه بنفس الطريقة الأخيرة . و هذ يضمن أن التطبيقات تستخدم الصنف الأصل تستطيع أيضا إستخدام الصنف البنت, دون تعديل نفسها .

نحصل على هذه النتيجة عن طريق إضافة برامتر واحد أو أكثر, و القيم الإفتراضية تجبر السلوك القديم . لذلك, عندما لا نقد أي برامتر للبرامتر **rel**, البرامترات **x** و **y** يستخدمون كإحداثيات مطلقة (السلوك القديم للأسلوب) . من جانب أخر, إذا كنت تقدم ل **rel** برامتر صحيح, يتم التعامل مع البرامترات **x** و **y** كنزوح نسبي (سلوك جديد) .

* السطور من 17 إلى 33 : سيتم إنشاء التنقلات المطلوبة بشكل عشوائي . لذلك نحن بحاجة لتوفير نظام حاجز, بحيث ينتقل الكائن و لا يخرج من اللوحة .

* السطر 42 : نحن نشير إلى الإحداثيات الجديدة الناتجة للبرنامج المستدعي , قد يكون جيدا أن المدفع يتنقل دون معرفة موقعه الأولي .

* السطور 44 إلى 49 : و هذه مرة أخرى نتجاوز فيها أسلوب موجود في صنف الأصل, و ذلك للحصول على سلوكيات مختلفة : بعد كل طلقة, نحن لا نقوم بتشتيت كل المدافع الحالية, لكن فقط الذي أطلق النار .

* السطور من 51 إلى 55 : أسلوب تم إضافته تحسبا من التطبيقات التي ترغب في تثبيت أو إزالة مدافع على مدار اللعبة .

* السطر 57 و الذي يليه : هذا الصنف الجديد تم تصميمه من البداية بحيث يمكن بسهولة أن يشتق , و هذا هو سبب في أننا قسمنا المنشئ إلى جزئين : الأسلوب**__init__()** التي تحتوي على التعليمات البرمجية المشتركة بين جميع الكائنات و التي سيتم تمثيل من هذا الصنف الذي يحب تمثيلهم من

الصنف المشتق الممكن . الأسلوب **specificites()** يحتوي على أجزاء من الكود أكثر تحديدا : الهدف من هذا الأسلوب هو واضح و هو أن يتم تجاوز الأصناف المشتقة الممكنة .

لعبة البينغ

في الصفحات التالية, سوف تجد سكريبت لبرنامج صغير كامل . تم توفير هذا البرنامج كمثال على ما يمكن أن يتم النظر إلى تطوير نفسك كمشروع شخصي . و هذا يريك مرة أخرى كيفية يمكنك إستخدام أصناف متعددة التي يمكن لمنشئ السكريبت تنظيمه . لكنه يظهر لكم كيف يمكن إنشاء تطبيق واجهة مستخدم رسومية بحيث يمكن تغيير حجم كل شيئ فيها .

المبدأ

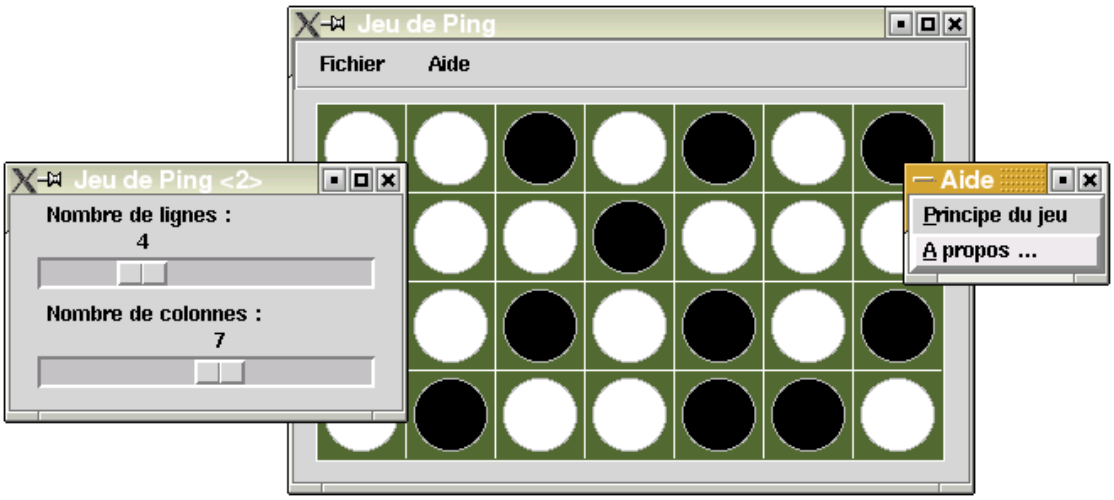
اللعبة التي تنفذ هنا هي أشبه بتمرين رياضيات . إنها تلعب على شبكة من الأحجام متغيرة, و كل خانة بها بيدق . و هذه البيادق بها وجهين الأبيض و الأسود (مثل بيادق لعبة Othello/Reversi) في بداية التمرين تكون كلها بالوجه الأبيض .

عند النقر على بيدق. البيادق الأربعة المجاورة تعود . اللعبة هي أن يتم إعادة جمبع البيادق, بالضغط على بعض منها .

التمرين سيكون سهلا جدا م شبكة ب 2 × 2 مربعات (يكفي الضغط على كل واحد من القطع الأربعة) . سيكون أكثر صعوبة مع شبكة أكبر, و مستحيل مع البعض منهم . يجب أن تحدد أيها .

لا تهمل النظر على شبكة 1 × n .

> يمكنك العثور على المناقشة الكاملة للعبة البينغ, و نظريتها و إمتداداتها, في مجلة *la science no298* لشهر أوت 2002, الصفحات من 98 إلى 102, أو على الموقع الألكتروني لجامعة *Lille* :
> *http://www2.lifl.fr/~delahaye/dnalor/JeuAEpisodes.pdf*<



برمجة

عند برمجة مشروع برمجي, حاول دئاما وصف نهجك (طريقك) بأكبر قدر من الوضوح . أبدأ مع مواصفات مفصلة و لا تهمل التعليق على تعليمات البرمجية, عند البرمجة (و ليس بعد) .

يمكنك أن تفرض على نفسك أن تعبر عما يريد الجهاز القيام به, و الذي يساعدك على تحلي المشاكل و هيكل التعليمات البرمجية لكودك بشكل صحيح .

مواصفات البرنامج الذي تريد تطويره

* سيتم البناء على أساس نافذة رئيسية مع لوحة اللعب و شريط القوائم .

* يجب أن توسغ الإدارة من قبل المستخدم, و خانات اللوحة تبقى مربعة .

* يجب على خيارات القائمة أن تسمح ب :

- تحديد حجك الشبكة (عدد المربعات) .

- إعادة ضبط اللعبة (و هذا بعني أن تصبح جميع البيادق بالوجه الأبيض) .

- إظهار مبدأ اللعبة في نافذة مساعدة .

- إنهاء (إغلاق التطبيق) .

* سوف تقوم بإنشاء ثلاثة أصناف :

- صنف الرئيسي .

- صنف شريط القوائم .

- صنف للوحة اللعبة .

* ستم وضع لوحة اللعبة على لوحة, و اللوحة مثبة على إطار (frame) . نعتبر أن تغيير الحجم بتم التحكم به من خلال المستخدم, و الإطار يحتل كل مرة كل المساحة المتاحة : أي ستقدم إلى المبرمج كأي مستطيل, أبعاده تسكون أساس حساب أبعاد شبكة التي سترسم .

* و بما أن هذه المربعات في الشبكة يجب أن تبقى مربعة, فمن السهل أن تبدأ بحساب حجمها الأقصى, ثم عين أبعاد اللوحة وفقا لذلك .

* إدارة نقرة الفأرة : فنقوم بربط اللوحة مع أسلوب-معالجة للخدق "ضغطة بالزر الأيسر" . إحداثيات الأحداث سيتم إستخدامها لتحديد في أي خانة في الشبكة (رقم السطر و رقم العمود) الذي تم البضغط عليه, بغ النظر على أبعاد الشبكة . في الخانات 8 المجاورة, البيادق سيتم "إرجاعهم" (التبديل الألوان الأسود و الأبيض) .

```
#########################################
```

```
# لعبة بينغ                    #
# Références : Voir article de la revue  #
# <Pour la science>, Aout 2002         #
#                                      #
# (C) Gérard Swinnen (Verviers, Belgique) #
# http://www.ulg.ac.be/cifen/inforef/swi  #
#                                      #
# Version du 29/09/2002 - Licence : GPL  #
##########################################

from tkinter import *

class MenuBar(Frame):
  """Barre de menus déroulants"""
  def __init__(self, boss =None):
    Frame.__init__(self, borderwidth =2, relief =GROOVE)
    ##### قائمة <الملف> #####
    fileMenu = Menubutton(self, text ='Fichier')
    fileMenu.pack(side =LEFT, padx =5)
    me1 = Menu(fileMenu)
    me1.add_command(label ='Options', underline =0,
            command = boss.options)
    me1.add_command(label ='Restart', underline =0,
            command = boss.reset)
    me1.add_command(label ='Terminer', underline =0,
            command = boss.quit)
    fileMenu.configure(menu = me1)

    ##### قائمة <مساعدة> #####
    helpMenu = Menubutton(self, text ='Aide')
    helpMenu.pack(side =LEFT, padx =5)
```

```
    me1 = Menu(helpMenu)
    me1.add_command(label ='Principe du jeu', underline =0,
            command = boss.principe)
    me1.add_command(label ='A propos ...', underline =0,
            command = boss.aPropos)
    helpMenu.configure(menu = me1)

class Panneau(Frame):
  """Panneau de jeu (grille de n x m cases)"""
  def __init__(self, boss =None):
    # هذه لوحة اللعبى تتكون من إطار يمكن تغيير حجمه يحتوي على لوحة . عند كل تغيير في حجم الإطار, نحن
نحسب أكبر حجم ممكن لمربعات الشبكة, و تكييف أبعاد اللوحة وفقا لذلك .
    Frame.__init__(self)
    self.nlig, self.ncol = 4, 4       #4 = شبكة أولية x 4
    #ربط الحدث <resize> لمعالج مناسب :
    self.bind("<Configure>", self.redim)
    #اللوحات :
    self.can =Canvas(self, bg ="dark olive green", borderwidth =0,
            highlightthickness =1, highlightbackground ="white")
    #ربط الحدث <clic de souris> لمعالجه :
    self.can.bind("<Button-1>", self.clic)
    self.can.pack()
    self.initJeu()

  def initJeu(self):
    "Initialisation de la liste mémorisant l'état du jeu"
    self.etat =[]            #صنع قائمة من قوائم
    for i in range(12):        #(يعادل جدول من
      self.etat.append([0]*12)  #  12 خط x 12 عمود)

  def redim(self, event):
```

```
    "Opérations effectuées à chaque redimensionnement"
    #الخصائص المرتبطة مع حدث إعادة التكوين يحتوي على أبعاد الجديدة للإطار :
    self.width, self.height = event.width -4, event.height -4
    #و يستخدم فرق 4 بيكسلات للتعويض عن سمك "الحدود" المحيط باللوحة .
    self.traceGrille()

  def traceGrille(self):
    "Dessin de la grille, en fonction des options & dimensions"
    #العرض و الإرتفاع الأقصى للمربعات
    lmax = self.width/self.ncol
    hmax = self.height/self.nlig
    #: جانب لمربع يساوي أصغر هذه الأبعاد
    self.cote = min(lmax, hmax)
    #-> إنشاء أبعاد جديدة للوحة :
    larg, haut = self.cote*self.ncol, self.cote*self.nlig
    self.can.configure(width =larg, height =haut)
    #: تخطيط الشبكة
    self.can.delete(ALL)          #محو الرسوم السابقة
    s =self.cote
    for l in range(self.nlig -1):     #خطوط أفقية
      self.can.create_line(0, s, larg, s, fill="white")
      s +=self.cote
    s =self.cote
    for c in range(self.ncol -1):     #خطوط عمودية
      self.can.create_line(s, 0, s, haut, fill ="white")
      s +=self.cote
    #: تتبع جميع القطع, بيضاء أو سوداء حسب حالة اللعبة
    for l in range(self.nlig):
      for c in range(self.ncol):
        x1 = c *self.cote +5        #حجم القطع(البيدق) =
        x2 = (c +1)*self.cote -5      #حجم المربع -10
```

```
            y1 = l *self.cote +5          #
            y2 = (l +1)*self.cote -5
            coul =["white","black"][self.etat[l][c]]
            self.can.create_oval(x1, y1, x2, y2, outline ="grey",
                        width =1, fill =coul)

    def clic(self, event):
      "Gestion du clic de souris : retournement des pions"
      #نبدء بتحديد السطر و العمود
      lig, col = int(event.y/self.cote), int(event.x/self.cote)
      #: بعد ذلك نقوم بمعالجة 8 المربعات المجاورة
      for l in range(lig -1, lig+2):
        if l <0 or l >= self.nlig:
          continue
        for c in range(col -1, col +2):
          if c <0 or c >= self.ncol:
            continue
          if l ==lig and c ==col:
            continue
          #: عكس البيدق بعكس منطقي
          self.etat[l][c] = not (self.etat[l][c])
      self.traceGrille()


class Ping(Frame):
    """corps principal du programme"""
    def __init__(self):
      Frame.__init__(self)
      self.master.geometry("400x300")
      self.master.title(" Jeu de Ping")
```

```
    self.mbar = MenuBar(self)
    self.mbar.pack(side =TOP, expand =NO, fill =X)

    self.jeu =Panneau(self)
    self.jeu.pack(expand =YES, fill=BOTH, padx =8, pady =8)

    self.pack()

  def options(self):
    "Choix du nombre de lignes et de colonnes pour la grille"
    opt =Toplevel(self)
    curL =Scale(opt, length =200, label ="Nombre de lignes :",
        orient =HORIZONTAL,
        from_ =1, to =12, command =self.majLignes)
    curL.set(self.jeu.nlig)    #الموقع الأولي للمؤشر
    curL.pack()
    curH =Scale(opt, length =200, label ="Nombre de colonnes :",
        orient =HORIZONTAL,
        from_ =1, to =12, command =self.majColonnes)
    curH.set(self.jeu.ncol)
    curH.pack()

  def majColonnes(self, n):
    self.jeu.ncol = int(n)
    self.jeu.traceGrille()

  def majLignes(self, n):
    self.jeu.nlig = int(n)
    self.jeu.traceGrille()

  def reset(self):
```

```
    self.jeu.initJeu()
    self.jeu.traceGrille()

  def principe(self):
    "Fenêtre-message contenant la description sommaire du principe du jeu"
    msg =Toplevel(self)
    Message(msg, bg ="navy", fg ="ivory", width =400,
      font ="Helvetica 10 bold",
      text ="Les pions de ce jeu possèdent chacun une face blanche et "\
      "une face noire. Lorsque l'on clique sur un pion, les 8 "\
      "pions adjacents se retournent.\nLe jeu consiste a essayer "\
      "de les retouner tous.\n\nSi l'exercice se révèle très facile "\
      "avec une grille de 2 x 2 cases. Il devient plus difficile avec "\
      "des grilles plus grandes. Il est même tout à fait impossible "\
      "avec certaines grilles.\nA vous de déterminer lesquelles !\n\n"\
      "Réf : revue 'Pour la Science' - Aout 2002")\
      .pack(padx =10, pady =10)

  def aPropos(self):
    "Fenêtre-message indiquant l'auteur et le type de licence"
    msg =Toplevel(self)
    Message(msg, width =200, aspect =100, justify =CENTER,
      text ="Jeu de Ping\n\n(C) Gérard Swinnen, Aout 2002.\n"\
      "Licence = GPL").pack(padx =10, pady =10)

if __name__ == '__main__':
  Ping().mainloop()
```

> تذكير
>
> إذا كنت ترغب في تجربة هذه البرامج بدون إعادة كتابتها, يمكنك العثور على كودها على :
> *http://www.inforef.be/swi/python.htm*.

16

إدارة قواعد البيانات

قواعد البيانات هي أدوات تستخدم بشكل متزايد . يتك إستخدامها لتخزين البيانات العديدة في حزمة واحدة منظمة بشكل جيد . عندما يتعلق الأمر بقواعد البيانات العلائقية, يمكن تمام تجنب "جحيم التكرار" . ربما قد واجهة بالفعل هذه المشكلة : تم تخزين نفس البيانات في ملفات مختلفة . و عندما تريد تعديل أو حذف أي من هذه البيانات, يجب عليك فتحها و تعديل أو حذفها في جميع الملفات التي تحتويها ! و إحتمال الخطأ كبير جدا, و هذا الأمر يؤدي إلى التضارب, ناهيك عم ضياع الوقت . و الحل لهذه المشكلة هي قواعد البيانات . البايثون يتيح لك طرق مختلفة لإستخدام موارد الكثير من الأنظمة, لكننا لن نختبر سوى مثالين : SQLite و PostgreSQL .

قواعد البيانات

هنالك العديد من قواعد البيانات . يمكننا على سبيل المثال تعتبر قاعدة البيانات ملف يحتوي على قائمة من الأسماء و العناوين .

إذا كانت القائمة ليست طويلة جدا, و إذا كنت لا تريد أن تكون قادر على تنفيذ عمليات البحث على أساس معايير معقدة, فمن النافلة انه يمكن الوصول إلى هذا النوع من البيانات بإستخدام تعليمات بسيطة مثل تلك التي نافشناها في الصفحة 107 .

و سيتعقد الوضع بسرعة كبيرة إذا كنا نريد تحديد و فرز البيانات, خاصة إذا إزداد عددهم . و ستزداد الصعوبة إذا تم سرد البيانات في مجموعات مختلفة متصلة بواسط عدد من العلاقات, و إذا كان هنالك العديد من المستخدمين بحاجة إلى الوصول إليها في نفس الوقت .

على سبيل المثال, تخيل أن إتجاه مدرستك يتعهد لكم بتطوير نظام نشرة محوسبة . حان وقت التفكير قليلا. كنت قد أردتكت بسرعة أنه يجب تنفيذ مجموعة من الجداول المختلفة : جدول أسماء الطلاب (و التي قد تحتوي بطبيعة الحال معلومات خاصة بهؤلاء الطلاب : العنوان, تاريخ الميلاد, إلخ ...) و جدول يحتوي على قائمة الدروس (مع إسم الإستاذ, و عدد ساعات التعليم في الأسبوع و إلخ .) . و جدول لتخزين أعمال pris en compte pour l'évaluation (مع أهميتخا, و تاريخها, و محتواها إلخ .) . و جدول يصف كيف يمكن جمع الطلاب في مجموعات حسب الفصول أو الخيارات, و الدروس التي أخذها كل واحد, إلخ .

يجب أن تعرف أن هذه الجداول ليسا مستقلة . بل ترتبط بالعمل الذي قام به الطالب مع دروسه المختلفة . لتحديد مقدار نشرة الطالب, غذا لابد من إستخراج من جداول العمل, بالطبع, لكن مع المعلومات الموجود في الجداول الأخرى (هذه الدورات و الصفوف و الخيارات و إلخ .) .

سوف نرى لاحقا كيفية تمثيل جداول و العلاقات بينها .

SGBDR – RGB - نموذج عميل\خادم (سيرفر)

البرامج الحاسوبية قادرة على إدارة مجموعات من البيانات المعقدة (معقدة كثيرا أيضا) , و ندعو هذه البرامج ب SGBDR (و هي إختصار لكلمة فرنسية معناها إدارة أنظمة قواعد البيانات العلائقية) . و هذه التطبيقات المعلوماتية مهمة جدا للشركات . بعض من هذه الشركات المتخصصة في صناعتها : IBM و أوركل و ميكروسوفت و informix و Sybase ...) و عادة ما تباع بأسعار مرتفعة . و قد تم تطوير برامج الاخر في مراكز أبحاث و تدريس جامعي (PostgreSQL, SQLite, MySQL...) . و عادة ما تكون مجانية .

هذه الأنظمة لدى كل واحدة منها خصائصها و أدائها, و لكن معظمها تعمل على نموذج عميل\سيرفر : و هذا يعني الجزء الأكبر من البرنامج (يتم إدارته من خلال قواعد البيانات) يتم تثبيته في مكان واحد, من حيث المبدأ على الة قوية (و هذا يشكل الخادم(السيرفر) بأكمله), في حين أن الأخر أكثر بساطة من ذلك بكثير, فهو يتم تثبيته على أي عدد من محطات العمل , تدعى العملاء (clients) .

و يتم ربط العملاء بالخادم (سيرفر), بشكل دائم أو لا, عن طريق طرق مختلفة و بروتوكولات (ربما عن طريق الإنترنات) . كل واحد منهم يمكنه الوصول إلى جزء مهم جدا أو أقل أهمية, مع موافقة أو

لا لتعديل بعضها, إضافة أو حذف, إعتمادا على قواعد محددة جدا, تم تعريفها عن طريق مدير قاعدة البيانات .

الخادم و العملاء هي في الواقع تطبيقات منفصلة التي تتبادل المعلومات . تخيل على سبيل المثال أنك أحد مستخدمي النظام .
للوصول إلى البيانات, يجب تشغيل تطبيق عميل في أي محطة عمل . عند تشغيله, يبدأ تطبيق العميل عن طريق تأسيس إتصال مع الخادم و قاعدة البيانات[76]. عندما يتم تأسيس الإتصال, يمكن للتطبيق العميل الإستعلام عن الخادم عن طريق إرسال طلب في شكل المتفق عليه . فعلى سبيل المثال, عند البحث عن معلومة دقيقة . يتم تنفيذ الخادم عن طريق البحث في البيانات المناظرة في قواعد البيانات, ثم يرجع الإجابة للعميل .

و يعتبر هذا رد عن المعلومات المطلوبة, أو رسالة خطأ في حالة الفشل .
الإتصالات بين العميل و الخادم هي عبارة على طلبات و ردود . الطلبات هي تعليمات حقيقية يتم إرسالها من العميل إلى الخادم, و ليس فقط لإستخراج من قواعد البيانات, لكن أيضا إضافة أو حذف أو تعديل و إلخ .

لغة SQL

بعد قراءة ما سبق, سوف تفهم أننا لن نشرح في هذه الصفحات كيفية صنع برنامج خادم . و هذا فعل عمل المتخصصين (على سبيل المثال,كأنك ستطور لغة برمجة جديدة) . و أما تطوير برنامج عميل, هو شيئ في متناول اليد, و يمكنك تحقيق فائدة كبيرة . و يجب أن تعرف أن معظم التطبيقات "الجدية" تعمل على قواعد بيانات بمختلف التعقيد : حتى الألعاب يجب عليها تخزين الكثير من البيانات و الحفاظ على العلاقات بينها .

إعتمادا على إحتياجات تطبيقك, سيكون لديك الإختيار, إما أن تتصل بخادم بعيد يتمكن الإتصال به العديد من المستخدمين, و إما أن تصنع خادم محلي أقل أو أكثر كفاءة . في حالة تطبيق منفرد, يمكنك إستخدام برنامج خادم مثبت على نفس الجهاز الذي يوجد به تطبيقك, أو ببساطة أكثر,

---

76قد تحتاج إلى إدخال بعض المعلومات للوصول : عنوان سيرفر على الشبكة, إسم قاعدة البيانات, إسم المستخدم, كلمة المرور ...

إستخدام مكتبة خادم متوافقة مع لغة البرمجة الخاصة بك . سوف ترى أنه في جميع الحالات, الاليات التنفيذ ستبقى في الأساس نفسها .

يمكن للمرء أن يخاف (يخشى), في الواقع , إنه بالنظر إلى التنوع الكبير من الخوادم الموجوة, فمن الضروري إستخدام لغات مختلفة و برتوكولات لإرسال الطلبات إلى كل واحد منهم . لكن لحسن الحظ, بذلت جهود كبيرة لتوحيد تطوير لغة الإستعلام المشترط, و التي ما تسمى SQL (Structured Query Language - لغة الإستعلام الهيكلية) . فيما يتعلق بالبايثون, تم تقديم جهود إضافية لتوحيد الإجراءات للوصول إلى الملقمات نفسها, و على واجهة مشتركة (DBAPI[77]).

لذا يجب عليك حفظ بعض أساسيات اللغة للإستمرار, و لكن لا ينبغي أن تخاف . بالتأكيد سوف تجد فرصة للإلتقاء ب SQL في مجالات أخرى (على سبيل المثال, المكتبية) . في إطار محدود من هذه الدورة, يجب عليك مرفة بعض تعليمات لغة SQL بسيطة لفهم الاليات الأساسية و ربما جعل بعض المشاريع مثيرة للإهتمام .

SQLite

مكتبة بايثون القياسية تشمل محرك قاعدة بيانات علائقية يدعى SQLite[78], التي تم تطويرها بشكل مستقل بلغة السي, و تنفذ العديد من معايير SQL-92 .

---

[77] بايثون DataBase Application Programming Interface Specification يعرف مجموعة من قواعد السلوكية لمطوري الوحدات للوصول إلى SGBDR المختلفة, حيث أن هذه الوحدات قابلة للتبادل . و بالتالي, نفس تطبيق بايثون سوف يمكون قادرا على إستخدام SGBDR أو اخر, بسعر تبادل بسيط من الوحدات .

[78] ا SQLite (http://www.sqlite.org/) هو محرك قواعد بيانات الأكثر إستخداما في العالم . يتم إستخدامه في العديد من الأدوات مثل Firefox, Skype, Google Gears,, و في بعض منتجات أبل و أدوب و مكافي و في المكتبات القياسية في العديد من اللغات البرمج مثل PHP و البايثون . و هو أيضا الأكثر شعبية في النظم المضمتة, بما في ذلك الهواتف الذكية الحديثة . و هو مجاني و خالي من الحقوق .

و هذا يعني أنه يمكنك كتابة بالبايثون تطبيق يحتوي على SGBDR مدمج, دون الحاجة إلى تثبيت أي شيئ آخر, و هذا سيحسن الأداء .

سوف ترى في نهاية الفصل كيف تسير الأمور إذا كان التطبيق الخاص بك يجب أن يستخدم بدلا من ذلك خادم قواعد البيانات التي تم إستضافتها في جهاز آخر, و لكن المبادئ تبقى نفسها . كل هذا سوف تتعلمه مع SQLite و سيكون قابل للنقل دون تعديل, إذا كنت تريد لاحقا العمل مع SGDBR أكثر "فرض" مثل PostgreSQL أو MySQL أو Oracle .

لنبدأ على الفور لإستكشاف أساسيات هذا النظام, على سطر الأوامر . سوف نكتب فيما بعد سكريبت صغير لإدارة قاعدة بيانات بسيطة مع جدولين .

إنشاء قاعدة بيانات - كائنات "إتصال" و "مؤشر"

و كما كنت تتوقع, يجب إستدعاء وحدة للوصول إلى مميزاتها :

```
>>> import sqlite3
```

> الرقم في نهاية الإسم هو رقم الإصدار الحالي من وحدة واجهة في وقت كتابة هذه السطور . فمن الممكن أن يتم تغيير هذا في الإصدارات المستقبلية من البايثون .

ثم يجب عليك أن تقرر إسم الملف الذي تريد تعيين إلى قاعدة البيانات . SQLite يقوم بحفظ جميع جداول قاعدة البيانات في ملف واحد متعدد المنصات الذي يمكنك أن تقوم بحفظ أي شيئ تريد (و هذا يجب أن يبسط إلى حد كبير حياتك للأرشيفات !) :(

```
>>> fichierDonnees ="E:/python3/essais/bd_test.sq3"
```

إسم الملف يمكن أن يتضمن أي إسم المسار و أي إمتداد . و من الممكن إستخدام إسم خاص **:memory:**, الذي يشير إلى أن يتم معالجة قواعد البيانات في الذاكرة العشوائية (رام) فقط . و بذلك يمكنك إختصار الوقت و الوصول إلى البيانات, و التطبيق سيكون سريعة جدا, و الذي قد يكون ذا فائدة في سياق برنامج لعبة على سبيل المثال, شرط أن تكون هنالك آلية خاصة للحفظ على القرص .

سوف تقوم إذا بصنع كائن-إتصال, بمساعد دالة-صنع **connect()** . هذا الكائن يتفاعل بين البرنامج و قاعدة البيانات . العملية مماثلة تماما لإفتح ملف نصي, و ممثيل كائن سيصنع ملف التخزين (إذا كان الملف غير موجود) :

```
>>> conn =sqlite3.connect(fichierDonnees)
```

تم الأن وضع كائن الإتصال في مكانه, و سوف تكون قادر على التفاعل معه بإستخدام SQL . سيكون هذا ممكن مباشرة عن طريق إستخدام بعض أساليب هذا الكائن[79], لكن من المفضل أت تضع في مكانه لتحاور مع كائن-واجهة أخر الذي يسمى المؤشر . بل هو نوع من الذاكرة العازلة, لتخزين البيانات في الذاكرة بشكل مؤقت عند القيام بمعالجتها, فضلا عن عمليات التي تقوم لها عليها, قبل نقلها إلى قاعدة البيانات النهائية . هذه التقنية يجل من الممكن إلغاء إذا لزم الأمر عملية أو أكثر التي مهي غير كافية, و إعادتها إلى المعالجتها, دون أت تتأثر قاعدة البيانات (يمكنك معرفة المزيد عم هذا المفهوم من خلال إحدى وثائق التي تتعامل مع لغة SQL) .

```
>>> cur =conn.cursor()
```

قاعدة البيانات تتكون دائما من جدول أو أكثر, التي تحتوي على السجلات (أو المحفوظات), و هي تحتوي على أنواع مختلفة من المجالات . و هذه المفاهيم ربما كنت على دراية بها إذا كنت قد عملت مع أي جدول : السجلات يتم حفظها في أسطر الجدول, و المجالات في خلايا السطر . سوف نكتب أول إستعلام SQL لنطلب منه إنشاء جدول جديد :

```
>>> cur.execute("CREATE TABLE membres (age INTEGER, nom TEXT, taille REAL)")
```

يتم التعبير عن الإستعلام في سلسلة نصية كلاسيكية, و التي نريد تمريرها للمؤشر عبر أسلوبه **()excute** . لاحظ جيدا أن SQL يتجاهل حالة الأحرف, بحيث يمكن ترميز إستعلامات SQL بحروف كبيرة أو صغيرة (أو معا) . إخترنا شخصيا الكتابة بحروف كبيرة تعليمات هذه اللغة, و ذلك لتفرقة بين تعليمات البايثون المحيطة بها, و لكن بالطبع يمكنك أن تتبع عادات أخرى .

---

[79] وحدة SQLite توفر بعض الأساليب المختصرة للوصول إلى البيانات دون إستخدام المؤشر (أو على نحو أدق, وذلك بإستخدام مؤشر ضمني) . هذه الأساليب لا تتوافق مع التقنيات القياسية, و نحن نفضل تجاهل ذلك هنا .

كما يرجى ملاحظة أن أنواع البيانات لا تحمل نفس الأسماء في البايثون و في SQL . لا ينبغي على الترجمة أن تزعجك كثيرا . ملاحظة بسيطة و هي أن السلاسل النصية يتم ترميزها إفتراضيا ب Utf-8, حسب الإتفاقيية ذاتها مع الملفات النصية (أنظر للصفحة 114) .

يمكننا الأن إدخال السجلات :

```
>>> cur.execute("INSERT INTO membres(age,nom,taille) VALUES(21,'Dupont',1.83)")
>>> cur.execute("INSERT INTO membres(age,nom,taille) VALUES(15,'Blumâr',1.57)")
>>> cur.execute("INSERT Into membres(age,nom,taille) VALUES(18,'Özémir',1.69)")
```

إنتبه, في هذه المرحلة من العمليات, ستم حفظ السجلات في مؤشر عازل, لكنا لم تنقل بعد إلى قاعدة البيانات . لذا يمكنك إلغائها تماما, إذا لزم الأمر, كما سنرى بعد قليل . و سيتم تشغيل نقل البيانات من خلال الأسلوب **commit()** لكائن الإتصال :

```
>>> conn.commit()
```

و بعد الإنتهاء من العمل يمكنك إغلاق المؤشر, و كذلك الإتصال[80].

```
>>> cur.close()
>>> conn.close()
```

الإتصال بقاعدة بيانات موجودة

بعد العمليات أعلاه, تم إنشاء ملف يسمى **bd_test.sq3** في موقع محدد في جهازك . لقد قمت الأن بالخروج من البايثون و ربما قد أغلقت حاسوبك : البيانات التي تم حفظها, كيف يمكننا الوصول إليها مرة أخرى ؟ الأمر في غاية البساطة : يكفي أن تستخدم بالضبط هذه التعليمات :

```
>>> import sqlite3
>>> conn =sqlite3.connect("E:/python3/essais/bd_test.sq3")
>>> cur =conn.cursor()
```

---

[80] تطبيقات التي تستخدم قواعد بيانات كبيرة غالبا ما تكون تطبيقات متعددة المستخدمين . و سوف نرى لاحقا (صفحة) أن مثل هذه التطبيقات تنفذ عدة "أبناء" لتنفيذ متزامن للبرنامج, و تدعى المواضيع, من أجل التعامل مع الطلبات الموازية من عدة متسخدمين مختلفين . و بالتالي سوف يكون لكل واحد كائنات إتصالات و مؤشر داخل البرنامج نفسه, و أنه لن يكون هنالك تضارب . في حالة SQLite, و هو نظام مستخدم منفرد, إغلاق الإتصالات يتسبب أيضا بإغلاق الملف الذي يحتو على قاعدة البيانات, و التي سوف يختلف عن النظام الكبير .

يتم تنفيذ الإستعلام بالطبع بمساعدة إستعلامات SQL, الذي يعكي للأسلوب **execute()** للمؤشر, دائما في شكل سلسلة نصية :

```
>>> cur.execute("SELECT * FROM membres")
```

هذا الإستعلام يقوم بطلب تحديد مجموعة معينة من السجلات, التي سيتم تحويلها من قاعدة البيانات إلى المؤشر . في هذه الحالة, التحديد ليس عنصر واحد, لأننا طلبنا أن يتك إسترداد جميع سجلات الجدول **membres** .

> تذكر أن الرمز * يستخدم كثير في المعلوماتية ك"جوكر" بمعني "كل" .

السجلات المحددة هي الأن في المؤشر . فإذا أردنا أن نراها, يجب علينا إستخراجها . و هذا يتم بطريقتين, و قد تبدو للوهلة الأولى مختلفة, لكن في الواقع أن الطريقتين لكائن-المؤشر يتم صنعها من البايثون هي مكررة, و هذا يعني, جهاز توليد المتسلسلات .[81]

يمكنك الذهاب مباشرة إلى التسلسل المنتج, بمساعدة حلقة for الكلاسيكية, و سوف تحصل على مجموعة من الأنفاق :

```
>>> for l in cur:
...     print(l)
...
(21, 'Dupont', 1.83)
(15, 'Blumâr', 1.57)
(18, 'Özémir', 1.69)
```

أو يتم جمعها في قائمة أو نفق لمزيد من المعالجة (بمساعدة الدالات المدمجة **list()** أو **tuple()**) :

```
>>> cur.execute("SELECT * FROM membres")
>>> list(cur)
[(21, 'Dupont', 1.83), (15, 'Blumâr', 1.57), (18, 'Özémir', 1.69)]
```

بطريقة أكثر كلاسيكية, يمكنك أيضا إستدعاء الدالة **fetchall()** للمؤشر, التي تقوم بإرجاع قائمة أنفاق :

---

[81] التكرارات هي جزء من مميزات المتقدمة للبايثون . نحن لن ندرسها في هذا الكتاب, و كذلك العديد من الأدوات الأخرى المثيرة للإهتمام, مثل تعريف الوظيفي للقوائم, و الديكورات, إلخ . و سوف تظل أشياء كثيرة لا تزال لإستكشافها' إذا كنت تريد إستكشاف هذه اللغة !

```
>>> cur.execute("SELECT * FROM membres")
>>> cur.fetchall()
[(21, 'Dupont', 1.83), (15, 'Blumâr', 1.57), (18, 'Özémir', 1.69)]
```

كما أن المؤشر لا يزال مفتوحا, يمكنك بالطبع إضافة سجلات إضافية :

```
>>> cur.execute("INSERT INTO membres(age,nom,taille) VALUES(19,'Ricard',1.75)")
```

في برنامج عملي, البيانات التي تريد تسجيلها يتم حفظها في متغيرات تنشأ في الغالب في متغيرات البايثون . و سوف تحتاج أيضا إلى إنشاء سلسلة نصية تحتوي على طلب إستعالم SQL, لتسمل قيم من هذه المتغيرات . فمن المستحسن إستخدامها لهذا الغرض في التقنيات الهادية لتنسيق السلاسل, لأن هذه قد تفتح ثغرة أمنية في برامجها, و تسمح لإقتحامها عن طريق طريقة تدعى SQL Injection (حقن SQL)[82]. ولذلك يجب التأكد من تنسق إستعلاماتك للوحدة الواجهة نفسها . و التقنية السليمة أدناه : سلسلة "رئيس" تستخدم علامة إستفهام كعلامات التحويل, و تنسيق نفسه معتمد من قبل الأسلوب **execute()** للمؤشر :

```
>>> data =[(17,"Durand",1.74),(22,"Berger",1.71),(20,"Weber",1.65)]
>>> for tu in data:
...     cur.execute("INSERT INTO membres(age,nom,taille) VALUES(?,?,?)", tu)
...
>>> conn.commit()
```

في هذا المثال, سلسلة الإستعلام تحمل 3 علامات إستفهام, و التي هي علاماتنا . سوف يتم إستبدالهم ب 3 عناصر من نوع أنفاق في كل تكرار للحلقة, وحدة الواجهة مع SQLite يتم تحميلها مع كل متغير وفقا لنوعه .

في هذه المرحلة من العمليات, قد تعتقد أن كل ما رأيناه هو معقد للغاية لكتابة و قراءة المعلومات في ملف . لن يكون أكثر بساطة إذا إستخدمت معالجات ملفات التي نعرفها ؟ نعم و لا . هذا صحيح بالنسبة للكميات الصغيرة من المعلومات التي لا تحتاج إلى تغيير كبير مع مرور الوقت . لككنا لا يمكننا الدفاع عنه إذا أخذنا مشكلة بسيطة للتعديل أو حذف أو إضافة أي سجل . في قاعدة البيانات, هذا بسيط جدا :

```
>>> cur.execute("UPDATE membres SET nom ='Gerart' WHERE nom='Ricard'")
```

---

[82] هذه المشكلة الأمنية تنشأ عن طريق تطبيقات الويب, الهجوم يتم بإستخدام حقل نموذج HTML و يردي إلى حقن التعليمات SQL بالبيانات الخبيثة حيث يتوقع البرنامج أن السلاسل غير مؤذية . و مع ذلك, فمن المستحسن إستخدام تقنيات برمجة أكثر أمنا, حتى تطبيق بسيط لشخص واحد .

لحذف سجل أو أكثر, إستخدم إستعلام مثل هذه :

```
>>> cur.execute("DELETE FROM membres WHERE nom='Gerart'")
```

مع ما نعرفه من الملفات النصية, يجب علينا بالتأكيد كتابة العديد من الأسطر(كود) للحصول على نفس الشيئ ! و لكن هنالك الكثير مثير للإهتمام .

> إنتبه
>
> لا تنسى أن تغييرات المؤشر تجدث في الرام, و بالتالي لن يتم حفظ أي شيئ بشكل دائما ما لم تقم بتشغيل التعليمة ***conn.commit***() . يمكنك إلغاء جميع التغييرات متذ ***commit***() السابق, و إغلاق الإتصال بإستخدام الأمر ***conn.close***() .

البحث التحديدي في قاعدة بيانات

تمرين

16.1 قبل المضي قدما, و بوصفها تمرين تجميعي, سوف أطلب منك إنشاء قاعدة بيانات " **Musique**" التي تحتوي على الجدولين التاليين (هذه بعض الأعمال, لكن يجب أن تكون قادرا على تتعامل مع عدد من البيانات بشكل صحيح لإختبار دالات البحص و الفرز المعتمد من قبل SGBDR) :

Compositeurs
comp (chaîne)
a_naiss (entier)
a_mort (entier)

Oeuvres
comp (chaîne)
titre (chaîne)
duree (entier)
interpr (chaîne)

أبدأ بملئ جدول **Compositeurs** مع البيانات التالية (إغتنم هذه الفرصة لإظهار مهاراتك التي تعلمتها من خلال كتابة سكريبت صغير لتسهيل إدخال المعلومات : تحتاج إلى حلقة !) :

```
comp            a_naiss       a_mort

Mozart          1756          1791
Beethoven       1770          1827
```

```
Haendel         1685          1759
Schubert        1797          1828
Vivaldi         1678          1741
Monteverdi      1567          1643
Chopin          1810          1849
Bach            1685          1750
Shostakovich    1906          1975
```

في جدول **oeuvres**, أدخل البيانات التالية :

```
comp          titre                        duree    interpr

Vivaldi       Les quatre saisons            20      T. Pinnock
Mozart        Concerto piano N°12           25      M. Perahia
Brahms        Concerto violon N°2           40      A. Grumiaux
Beethoven     Sonate "au clair de lune"     14      W. Kempf
Beethoven     Sonate "pathétique"           17      W. Kempf
Schubert      Quintette "la truite"         39      SE of London
Haydn         La création                  109      H. Von Karajan
Chopin        Concerto piano N°1            42      M.J. Pires
Bach          Toccata & fugue                9      P. Burmester
Beethoven     Concerto piano N°4            33      M. Pollini
Mozart        Symphonie N°40                29      F. Bruggen
Mozart        Concerto piano N°22           35      S. Richter
Beethoven     Concerto piano N°3            37      S. Richter
```

يحتوي الحقلين **a_naiss** و **a_mort** على سنة الميلاد و سنة موت الملحنين . مدة تنفيذ هذا في دقائق . بالطبع يمكنك إضافة العديد من السجلات للملحنين و المؤلفين التي تردها, لكن تلك المذكورة أعلاه ينيغي أن تكون كافية لبقية المظهر .

في ما يلي, نحن نفتض أنك قد قمت بترميز البيانات في االجدولين أعلاه . إذا كانت لديك صعوبة في كتابة السكريبت المطلوب, يرجى الرجوع إلى تمرين 16.1 في صفحة 437 .

السكريبت الصغير بالأسفل يوفر أغراض المعلومات فقط . بل هو عميل SQL بدائي, الذي يسمح لك بالإنصال بقاعدة البيانات "**musique** - موسيقى" التي يجب أن تكون الأن موجودة في الدليل الخاص بك, و يتم فتح المؤشر لإتخدامه للإستعلام . لاحظ مرة أخرى أننا que rien n’est transcrit sur le disque tant que la méthode **commit()** n’a pas été invoquée.

```
#إستخدام قاعدة بيانات ضغيرة تقبل تعليمات SQL

import sqlite3
```

```
baseDonn = sqlite3.connect("musique.sq3")
cur = baseDonn.cursor()
while 1:
  print("Veuillez entrer votre requête SQL (ou <Enter> pour terminer) :")
  requete = input()
  if requete =="":
    break
  try:
    cur.execute(requete)        #تشغيل إستعلام SQL
  except:
    print('*** Requête SQL incorrecte ***')
  else:
    for enreg in cur:           #إظهار الناتج
      print(enreg)
  print()

choix = input("Confirmez-vous l'enregistrement de l'état actuel (o/n) ? ")
if choix[0] == "o" or choix[0] == "O":
  baseDonn.commit()
else:
  baseDonn.close()
```

هذا التطبيق البسيط جدا من الواضح أنه مثال . ينبغي أن نضيف خيار لإختيار قاعدة البيانات و الدليل . إستخدمنا لمنع السكريبت من "زرع" عندما يقوم المستخدم بترميز إستعلام غير صحيح, إستخدمنا هنا معالجة الإستثناءات التي قمنا بشرحها في الصفحة 117 .

إستعلام التحديد (select)

واحدة من أقوى تعليمات لغة SQL هي التعليمة **select**, التي سنرى الأن بعض مميزاته . و تذكر مرة أخرى أننا سنتناول هنا جزء صغير جدا من هذا الموضوع : الوصف التفصيلي ل SQL يجب أن يشرح في كتب عديدة .

شغل إذا السكريبت أعلاه, وحلل بدقة ما يحدث عند تقديم الإستعلامات التالية :

```
select * from oeuvres

select * from oeuvres where comp = 'Mozart'

select comp, titre, duree  from oeuvres order by comp

select titre, comp from oeuvres where comp='Beethoven' or comp='Mozart'
   order by comp

select count(*) from oeuvres

select sum(duree) from oeuvres

select avg(duree) from oeuvres

select sum(duree) from oeuvres where comp='Beethoven'

select * from oeuvres where duree >35 order by duree desc

select * from compositeurs where a_mort <1800

select * from compositeurs where a_mort <1800 limit 3
```

لكل واحدة من هذه الإستعلام, للتعبير عن أفضل ما سيحدث . في لأساس, قمت بتفعيل المرسحات على قواعد البيانات للتحديد و الفرز .

الإستعلامات التالية هي أكثر تطويرا, لأنها تصل جدولين في المرة الواحدة .

```
select o.titre, c.comp, c.a_naiss from oeuvres as o, compositeurs as c where o.comp
=c.comp

select comp, titre, a_naiss from oeuvres join compositeurs using(comp)

select * from oeuvres join compositeurs using(comp) order by a_mort

select comp from oeuvres intersect select comp from compositeurs

select comp from oeuvres except select comp from compositeurs

select comp from compositeurs except select comp from oeuvres

select distinct comp from oeuvres union select comp from compositeurs
```

لا يمكننا أن نطور لغة إستعلام في السياق المحدود من هذا الكتاب . و مع ذلك سوف نختبر مثال أخر لتجسيد بايثون عند إستخدام قواعد البيانات, لكن على إفتراض أن الوقت قد حان لإجراء إتصال بنظام خادم مستقل (و التي يمكن أن تكون على سبيل المثال خادم قواعد بيانات كبير للشركات, خادم وثائق ف مدرسة, إلخ .) . كما أن هنالك العديد من البرامج الممتازة الحرة و المفتوحة المصدر, يمكنك البدء بسهولة

إستخدام خادم فاعل للغاية مثل PostgreSQL[83]. سوف يكون التمرين للإهتمام بوجه خاص إذا كنت تريد أن تأخذ عناء تثبيت برنامج خادم على جهاز منفصل عن محطة العمل الخاصة بك و , و سوف تربط الإثنين بإتصال عبر الشبكة من نوع TCP/IP .

مشروع برنامج عميل ل PostgreSQL

لإنهاء هذا الفصل, فسوف نقترح عليك في الصفحات القادم مثال تطبيق عملي . لن نصنع برنامج حقيقي (الموضوع يتطلب كتاب مخصص). لكن مشروع (نموذج) تحليلي, مصمم ليبين لك كيف يمكنك "التفكير مثل مبرمج" عندما نحصل على مشكلة معقدة .

التقنية التي سننفذها هنا هي إقتراحات بسيطة, و التي نحاول إستخدام أفضل أدوات التي إكتشفتها من خلال تعلمك في الفصول السابقة, و هي : هياكل البيانات عالية المستوى (القوائم و القواميس). و البرمجة الشيئية(بواسطة الكائنات) . و غني عن القول أن أقوم بإنتقاد نطاق واسع من الخيارات التي في هذا التمرين : يمكنك بالطبع علاج نفس المشاكل بإستخدام طرق مختلفة .

هدفنا هو الحصول على عميل بدائي بسرهة, قادر على تواصل "حقيقي" مع خادم قاعدة البيانات ز نحن نريد لعميلنا أن يبقى أدات صغيرة عامة جدا : يجب أن يكون قادر على إنشاء قاعدة بيانات صغيرة مع جداول متعددة, و إنتاج سجلات لكل واحدة, و السماح لنا بإختبار نتائج الإستعلامات SQL الأساسية .

في الأسطر التالية, نحن نفترض أن لديك بالفعل وصول إلى خادم PostgreSQL, التي بها قاعدة بيانات "discotheque" التي تم صنعها من للمستخدم "jules" و الذي كلمة مروره هي "abcde" .

---

[83] إن PostgreSQL هو SGBDR حر, متاحة تحت رخصة من نوع BSD .

هذا النظام متطور جدا قادر على منافسع غيره من نظم إدارة قواعد البيانات, الحرة (مثل MySQL و Firebird), أو الخاصة(مثل Oracle, Sybase, DB2 و Microsoft SQL Server) . المشاريع الحرة مثل Apache و لينكس و PostgreSQL لا يتم التحكم بها من قبل شركة واحدة, و لكن عن طريق مجتمع عالمي من المطورين و الشركات .

ملايين النسخ من PostgreSQL مثبة على خوادم ويب و خوادم تطبيقات .

هذا الخادم يمكن أن يتواجد على جهاز بعيد يمكن الوصول إليه عبر الشبكة, أو محليا على جهاز الحاسوب الخاص بك .

التكوين الكامل لخادم PostgreSQl هو خارج نطاق هذا الكتاب, لكن تثبيت أساسي ليس معقد على نظام تشغيل لبنكس عن طريق توزيعة كلاسيكية مثل دبيان و أبنتو و ريدهات و سوزي ... يكفي أن تثيت الحزمة التي تحتوي على الخادم (على سبيل المثال الحزمة Postgresql-8.4 في النسخة الحالية لتوزيعة أبنتو في وقت كتابة هذه السطور), ثم بتنفيذ العمليات القليلة التالية .

أدخل كمسؤول عن النظام لينطس(روت), و قم بتعديل ملف التكوين **pg_hba.conf** الذي ينبغي أن يكون في الدليل **/etc/postgresql** أو في **/var/lib/postgresql** . في هذا الملف, جميع الأسطر التعليقات تبقى كما هي (و هذا معناه الأسطر التي تبدأ بالرمز #), بإستثناء ما يلي .

```
local    all    postgres    ident
local    all    all         md5
host     all    all    0.0.0.0    0.0.0.0    reject
```

بمساعدة الأمر (النظام) **sudo passwd**, يمكنك إختيار كلمة مرور للمستخدم postgres . و هذا المستخدم تم إنشاءه تلقائيا أثناء تثبيت الحزمة, و التي سوف تكون رئيس الكبير (أو postmaster) لخادم PostgreSQL الخاص بك .

أعد تشغيل خدمة PostgreSQL, و ذلك بإستخدام الأمر :

```
sudo  /etc/init.d/postgresql-8.4  restart
```

يجب عليك تسجيل دخول بعد ذلك إلى نظام لينكس كمستخدم postgres, (في البداية, هو الوحد القادر على صنع مستخدمين جدد ل SGBDR), و قم بتشغيل الأمر **createuser** :

```
createuser jules -d -P
Saisir le mot de passe pour le nouveau rôle : *****
Le saisir de nouveau : *****
Le nouveau rôle est-il super-utilisateur ? (o/n) n
Le nouveau rôle est-il autorisé à créer de nouveaux rôles ? (o/n) n
```

هذه الأوامر لتعريف مستخدم جديد "jules" لنظام PostgreSQL, و هذا المستخدم يمكن الإتصال بكلمة السر الخاصة به (في تمريننا, "abcde") . إسم المستخدم هو إجراء تعسفي : لا يتوافق بالضرورة مع إسم المستخدم المدرج بالفعل في نظام لينكس .

يمكنك الأن إستئناف هوية المعتادة, و صنع قاعدة بيانات واحدة أو أكثر بإسم "jules", بمساعدة الأمر **createdb** :

```
createdb -U jules  discotheque
Mot de passe :  abcde
```

هذا يكفي في هذه المرحلة, الخادم PostgreSQL مستعد الأن للتفاعل مع عميل البايثون الذي سيتم شرحه في الصفحات القادمة .

وصف قاعدة بيانات في قاموس تطبيق

التطبيق الذي سيتفاعل مع قاعدة بيانات هو دائما تقريبا تطبيق معقد . فهو يحتوي بالضرورة على أسطر من التعليمات البرمجية, فمن الأفضل هيكلتها من خلال تجميعها في أصناف (أو على الأقل في دالات) مغلفة جيدا .

في أجزاء كثيرة من الكود, في كثير من الأحيان بعيدة جدا عن بعضا البعض , يجب على كتل البيانات أن تأخذ بالإعتبار هيكل قاعدة البيانات, و هذا يعني, قطع(تقسيمها) إلى جداول و حقول, و كذلك إقامة علاقات التسلسل الهرمي في السجلات .

يبدوا أن تجربة تبين لك هيكل قاعدة البيانات النهائية . خلال التطوير, نحن ندرك أنه غالبا ما يكون ضروريا إضافة أو إزالة حقول, و في بعض الأحيان تستبدل جدول مصمم بشكل سيئ بجدولين آخرين, إلخ . فإنه ليس من الحكم برمجة أجزاء برمجية خاصة جدا لهيكل معبنة, "من صعب" .

بدلا من ذلك, من المستحسن للغاية وصف بنية كاملة من قاعدة البيانات في نقطة واحدة في البرنامج, و من ثم إستخدام هذا الوصف كمرجع لصنع نصف-ألي تعيمات محددة حول الجدول أو الحقل . هذا يتجنب, إلى حد كبير, كابوس الحاجة إلى تعقب و تعديل عدد كبير من التعليمات في الكود, في كل مرة هيكل قاعدة البيانات يتغير قليلا . و بدلا من ذلك, مجرد تغيير وصف فقط من المرجع, و الجزء الأكبر من الكود لا يحتاج إلى تعديل .

> لدينا هنا فكرة واحدة لتحقيق تطبيقات قوية : و يجب دائما على برنامج لمعالج البيانات أن يكون مبني على أساس تطبيق قاموس .

ما نعنيه هنا أن "قاموس التطبيق" ليس بالضرورة أن يكون قاموس بايثون . أي بنية بيانات منسابة يمكن تحويلها, و الشيئ المهم هو بناء مرجع مركزي واصفا البيانات التي تقترح على التعامل, مع ربما مجموعة من المعلومات حول التنسيق .

بسبب قدرتها على جمع في كيان واحد من أي نوع, قوائم و أنفاق و قواميس تعمل لهذا العمل . في المثال في الصفحات القادمة, قمنا بإستخدام قاموس, قيمه هي قوائم من أنفاق و لكن يمكنك إختيار تنظيم مختلف من نفس المعلومات .

لترسيخ كل هذا, لايزال علينا حل سؤال مهم ك أين سنثبت هذا قاموس التطبيق ؟

و ينبغي النظر إلى المعلومات الخاصة من أي مكان في البرنامج . و لذلك فإنه تثبيته داخل متغير عام شيئ إلزامي, كبيانات أخرى اللازمة لتشغيل جميع برنامجنا . أنت تعرف أنه ليس من المستحسن إستخدام متغيرات عامة : لأنها تنطوي على مخاطر, تزداد مع زيادة حجم البرنامج . على أي حال, المتغيرات التي قلنا إنها عامة, لكنها في الحقيقة عامة داخل وحدة فقط . فإذا أردنا تنظيم برنامجنا على أنه مجموعة من الوحدات (و الذي هو ممارسة جيدة), لن يكون لدينا الوصول للمتغيرات العامة الخاص بنا سوى بين بعضها .

لحل هذه المشكلة الصغيرة, يوجد حل بسيطة و أنيق : جمع في صنف معين كافة المتغيرات التي تتطلب حالة عامة في التطبيق . ثم سوف نغلفها في مساحة أسماء الصنف, هذه المتغيرات يمكن إستخدامهم دون مشاكل في أي وحدة : يكفي أن يتم إستدعاء النصف . بالإضافة إلى ذلك, فإن إستخدام هذه التقنية تنطوى على نتيجة مثيرة للإهتمام : الرمز "عام - global" هي متغيرات تم تعريفها بهذه الطريقة لتظهر بوضوح في إسمها المؤهل, لأن هذا الإسم يجب أن يبدأ بالصنف الذي يحتويه .

إذا إخترت على سبيل المثال, إسم وصفي مثل **Glob** للصنف الهدف لإستيعاب متغيرتك "العالمية", يجب عليك صنع مرجع لعذع المتغيرات جميعه في الكود مع أسماء موصفية مثل **Glob.cela** ,**Glob.ceci** إلخ ...[84]

---

[84] يمكنك أيضا وضع متغيراتك "العامة" في وحدة تسمي Glob.py, ثم تقوم بإستدعائها . إن إستخدام وحدة أو صنف كمساحة للأسماء لتخزين متغيرات هي تقنية مماثلة تماما . إن إستخدام صنف قد يكون أكثر مرونة و قابلية للقراءة, لأن يمكن أن تصاحب بقية السكريبت,

هذه التقنية التي سوف تكتشفه في السطور الأولى لسكريبتنا . لقد قمنا بتعريف صنف **Glob()**, التي ليس لديها سوى منشئ بسيط . و لن تم تمثيل أي كائن من هذا الصنف, و في الواقع با تحتوي على أي أسلوب . متغيراتنا العامة سيتم تعرفهم كمتغيرات بسيطة للصنف, و حتى نتمكن من صنع مرجع لهم لبقية البرنامج كسمات ل **Glob()** . إسم قاعدة البيانات, على سبيل المثال, يمكن العثور عليه في المتغير **Glob.dbName** , إسم أو عنوان IP السيرفر موجود في المتغير **Glob.host**, إلخ :

```
1#     class Glob(object):
2#        """Espace de noms pour les variables et fonctions <pseudo-globales>"""
3#
4#        dbName = "discotheque"     #إسم قاعدة البيانات
5#        user = "jules"             #المالك أو المستخدم
6#        passwd = "abcde"           #كلمة السر
7#        host = "127.0.0.1"         #للخادم IP إسم و عنوان
8#        port =5432
9#
10#       #هيكل قاعدة البيانات. قاموس من جداول و حقول :
11#       dicoT ={"compositeurs":[('id_comp', "k", "clé primaire"),
12#                    ('nom', 25, "nom"),
13#                    ('prenom', 25, "prénom"),
14#                    ('a_naiss', "i", "année de naissance"),
15#                    ('a_mort', "i", "année de mort")],
16#          "oeuvres":[('id_oeuv', "k", "clé primaire"),
17#                ('id_comp', "i", "clé compositeur"),
18#                ('titre', 50, "titre de l'oeuvre"),
19#                ('duree', "i", "durée (en minutes)"),
20#                ('interpr', 30, "interprète principal")]}
```

قاموس التطبيق يصف هيكل قاعدة البيانات التي تحتوي داخل المتغير **Glob.dicoT** .

في هذا لقاموس, يوجد مفاتيح و أسماء الجداول . أما القيم, فكل منها عبارة عن قائمة تحتوي على وصف جميع الحقول(مجالات) في الجدول, على شكل أنفاق .

كل نفق يصف حقل معين في الجدول . لتجنب تبعثر تمريننا, سوف نحدد هذا الوصف إلى ثلاثة معلومات فقط : إسم الحقل و نوعه و تعليق قصير . في التطبيقات الحقيقية, فإنه سيتم وضع المزيد من المعلومات هنا, على سبيل المثال للقيم الحد التي لأي حقل حد من البيانات, و تنسيق تطبيق عندما يتم عرضه على الشاشة أو لطباعته, و النص يجب وضعه في الجزء الأعلى من العمود عندما نريد تقديمه في جدول, إلخ .

ثم الوحدت هي بالضرورة في ملف منفصل .

قد يبدو مملا جدا وصف بالتفصيل هيكل البيانات, لذلك سوف تبدأ على الفور بالتفكير حول مختلف الخوارزميات التي يجب أن يتم تنفيذها من أجل معالجتها . و التي يجب أن يتم القيام بها بشكل جيد, و هذا الوصف المنظم سوف يوفر لك الكثير من الوقت في وقت لاحق, لأنه سوف يسمح لك d'automatiser العديد من الأشياء . سوف ترى بعد قليل . بالإضافة إلى ذلك, يجب أن تقنع نفسك أن هذه المهمة الصعبة تؤهلك لهيكلة بشكل صحيح عملك : تنظيم النماذج و الإختبارات و إلخ .

تعريف صنف كائنات-واجهة

الصنف **Glob()** (تم وصفه سابقا) سيتم تثبيته في بداية السكريبت, أو في وحدة منفصلة يتم إستدعاءها في بداية السكريبت . و للبقية, سوف نفترض أنه يستخدم الصيغة الأخيرة : سوف نقوم بحفظ الصنف **Glob()** في وحدة تسمى **dict_app.py**, حيث يمكننا إستدعائها في السكريبت التالي .

هذا السكريبت الجديد يعرف صنف كائنات-الواجهة . في الوقع نحن نحاول الإستفادة مما تعلمناه في الفصول السابقة, و بالتالي التركيز على البرمجة الشيئية(الكائنات), لصنع قطعة من التعليمات البرمجية المغلفة و القابلة للإسخدام على نطاق واسع .

كائنات-الواجهات التي تريد صنعها ستكون مشابهة لكائنات-الملفات التي إستخدمناها لمعالجة الملفات في الفصل 9 . أنت تتذكر على سبيل المثال أننا نفتح الملف عن طريق إنشاء كائن-ملف, بإستخدام دالة-الصنع **open()** . بطريقة مماثلة, سوف نقوم بفتح التواصل مع قاعدة البيانات, نبدأ بصنع كائن-واجهة بإستخدام الصنف **GestionBD()**, و الذي سيقوم بتأسيس الإتصال . للقراءة أو للكتابة في ملف مفتوح, سوف نستخدم مختلف أساليب كائن-الملف . على نحو مماثل, سوف نجعل عملياتنا على قاعدة البيانات من خلال أساليب مختلفة لكائن-الواجهة .

```
1#    import sys
2#    from pg8000 import DBAPI
3#    from dict_app import *
4#
5#    class GestionBD(object) :
6#       """Mise en place et interfaçage d'une base de données PostgreSQL"""
7#       def __init__(self, dbName, user, passwd, host, port =5432):
8#          "Établissement de la connexion - Création du curseur"
9#          try:
10#            self.baseDonn = DBAPI.connect(host =host, port =port,
11#                                 database =dbName,
12#                                 user=user, password=passwd)
13#         except Exception as err:
```

```
14#            print('La connexion avec la base de données a échoué :\n'\
15#                'Erreur détectée :\n%s' % err)
16#            self.echec =1
17#          else:
18#            self.cursor = self.baseDonn.cursor()   #صنع مؤشر
19#            self.echec =0
20#
21#        def creerTables(self, dicTables):
22#          "Création des tables décrites dans le dictionnaire <dicTables>."
23#          for table in dicTables:          #تدوير مفاتيح القاموس
24#            req = "CREATE TABLE %s (" % table
25#            pk =''
26#            for descr in dicTables[table]:
27#              nomChamp = descr[0]       #تسمية حقل الذي تريد إنشاءه
28#              tch = descr[1]          #نوع الحقل الذي تريد إنشاءه
29#              if tch =='i':
30#                typeChamp ='INTEGER'
31#              elif tch =='k':
32#                #حقل 'مفتاح الأساسي' (عدد صحيح لزيادة تلقائيا)
33#                typeChamp ='SERIAL'
34#                pk = nomChamp
35#              else:
36#                typeChamp ='VARCHAR(%s)' % tch
37#              req = req + "%s %s, " % (nomChamp, typeChamp)
38#            if pk == '':
39#              req = req[:-2] + ")"
40#            else:
41#              req = req + "CONSTRAINT %s_pk PRIMARY KEY(%s))" % (pk, pk)
42#            self.executerReq(req)
43#
44#        def supprimerTables(self, dicTables):
45#          "Suppression de toutes les tables décrites dans <dicTables>"
46#          for table in list(dicTables.keys()):
47#            req ="DROP TABLE %s" % table
48#            self.executerReq(req)
49#          self.commit()                  #نقل -> القرص
50#
51#        def executerReq(self, req, param =None):
52#          "Exécution de la requête <req>, avec détection d'erreur éventuelle"
53#          try:
54#            self.cursor.execute(req, param)
55#          except Exception as err:
56#            #عرض إستعلام و رسالة خطأ النظام
57#            print("Requête SQL incorrecte :\n{}\nErreur détectée :".format(req))
58#            print(err)
59#            return 0
60#          else:
61#            return 1
62#
63#        def resultatReq(self):
64#          "renvoie le résultat de la requête précédente (une liste de tuples)"
65#          return self.cursor.fetchall()
66#
67#        def commit(self):
68#          if self.baseDonn:
69#            self.baseDonn.commit()       #نقل المؤشر -> القرص
70#
71#        def close(self):
72#          if self.baseDonn:
73#            self.baseDonn.close()
```

تعليقات

* الأسطر من 1 إلى 3 : بالإضافة إلى وحدة **dict_app** التي تحتوي على متغيرات "العامة", قمنا بإستدعاء وحدة **sys** التي تضم بعض دالات النظام, و خاصة وحدة **pg8000** التي تشمل كل ما يلزم للتواصل مع PostgreSQL . هذه الوحدة ليست جزءا من البايثون القياسية . بل هي وحدة واجهة بايثون-PostgreSQL متوفرة بالفعل للبايثون 3 . العديد من المكتبات الأخرى العديدة أكثر كفاءة, متوفر منذ مدة طويلة للإصدارات السابقة من البايثون, سوف تتكيف بالتأكيد (السائق الممتاز **psycopg2** سوف يكون جاهزا قريبا) .

> لتثبيت *pg8000*, أنظر إلى صفحة *Error: Reference source not found*.

* السطر 7 : عند إنشاء كائنات-الواجهة, سوف نوفر برامترات الإتصال : إسم قاعدة البيانات, إسم المستخدم, الإسم أو عنوان IP للجهاز الذي يقع الخادن . رقم منفذ الإتصال عادة ما يكون إفتراضيا . يفترض أن تكون كل هذه المعلومات لديك .

* الأسطر من 9 إلى 19 : من المستحسن للغاية وضع تعليمات برمجية لإجراء إتصال داخل معالج الإستثناء try-except-else (أنظر للصفحة 117), لأننا لا نستطيع أن نفترض أن الخادم سيكون يمكن الوصول إليه ضروريا . لاحظ أن الأسلوب**__init__()** لا يمكنه أن يرجع قيمة (بإستخدام التعليمة return), لأنها يتم إستدعائها تلقائيا بواسطة البايثون عندما تقون بتمثيل الكائن . في الواقع : ما يتم إرجاعه في هذا البرنامج المستدعي هو كائن بني حديثا . و بالتالي ليس هنالك تقرير نجاح أو فشل الإتصال بالبرنامج الإستعاد بإستخدام قيمة رجوع . و هنالك حل بسيط لهذه المشكلة الصغيرة لتخزين نتيجة محاولة الإتصال في سمة مثيل (المتغير **self.echec**), و يمكن للبرنامج الذي تم إستدعائه إختبار ما quand bon lui semble .

السطور من 21 إلى 42 : هذا الأسلوب automatise صنع جميع الجداول لقاعدة البيانات, للإستفادة من وصف قاموس التطبيق, يجب عليك تمرير برامتر . و سوف تكون هذه الالية أكثر أهمية من الواضح, لأن هيكل قاعدة البيانات سوف تكون أكثر تعقيدا (تخيل على سبيل المثال قاعدة بيانات

تحتوي على 35 جدول !) . حتى لا نعقد الإثبات, سوف نستخدم هذه القدوة لهذا الأسلوب لصتع حقول من أنواع integer و varchar . لا تتردد فب إضافة تعليمات لإنشاء حقول من أنواع أخرى .

إذا أردت تفصيل الكود, سوف تجد أنه ببساطة يمكنك بناء إستعلام SQL لكل جدول, قطعة قطعة, في سلسلة نصية **req** . ثم سوف يتم تمرير للأسلوب **executerReq()** لتنفيذه . فإذا كنت ترغب في عرض الإستعلام ثم بنائها, يمكنك بالطبع إضافة تعليمة **print(req)** مباشرة بعد السطر 42.

يمكنك أيضا إضافة للأسلوب قدرة تنفيذ قيود التكامل المرجعي, على أساس مكمل للقاموس التطبيق الذي يصف هذه القيود . نحن لا نطور هذه المشكلة هنا, لكن لا ينبغي vous poser de problème si vous savez de quoi il retourne.

* السطور من 44 إلى 49 : أبسط كثيرا من سابقتها, هذا الأسلوب يستخدم نفس مبدأ لحذف جميع الجداول الموضحة في قاموس التطبيق .

* السطور من 51 إلى 61 : هذا الأسلوب يوجه ببساطة إستعلام كائن المؤشر . فائدته هي تبسيط الوصول إليها و إنتاج رسالة خطأ إذا لزم الأمر .

* السطور من 63 إلى 73 : هذه الأساليب ليست سوى تتابع بسيط لكائنات التي يتم صنعها من وحدة **pg8000** : كائن-الإتصال ينتج بواسطة دالة-الصنع **DBAPI.connect()**, و كائن النؤشر المطابق . يسمح لهم بتبسيط الكود قليلا للبرنامج التي إستدعاه .

بناء نموذج مولد

لقد قمنا بإضافة هذا الصنف لتمريننا شرح كيفية يمكنك إستخدام نفس القاموس التطبيق لتطوير كود المولد . الفكرة هنا هي تحقيق صنف كائنات-أشكال قادرة على دعم ترميز السجلات أي الجداول , ببناء تلقائيا التعليمات الإدخال المناسبة بإستخدام المعلومات من قاموس التطبيق .

في تطبيق حقيقي, يجب أن يكون هذا النموذج مبسط و معدل بشكل كبير, و سوف يكون على الأرجح على شكل نافذة متخصصة, بها حقول الإدخال و الملاصق الخاص التي تتولد تلقائيا . نحن لا ندعي أنه ليس مثالا جيدا, لكننا نريد فقط أن نظهر لك كيف يمكنك automatiser منشئه إلى حد كبير . حاول القيام بنماذجك الخاصة بإستخدام مبادئ مماثلة .

```
1#     class Enregistreur(object):
2#         """classe pour gérer l'entrée d'enregistrements divers"""
```

```
3#      def __init__(self, bd, table):
4#        self.bd =bd
5#        self.table =table
6#        self.descriptif =Glob.dicoT[table]   # descriptif des champs
7#
8#      def entrer(self):
9#        "procédure d'entrée d'un enregistrement entier"
10#       champs ="("        #مسودة سلاسل لأسماء الحقول
11#       valeurs =[]        #قائمة للقيم المقابلة
12#       #طلب قيمة على التوالي لكل حقل ;
13#       for cha, type, nom in self.descriptif:
14#         if type =="k":   #نحن لن نطلب رقم تسجيل
15#           continue       #من المستخدم (ترقيم تلقائي)
16#         champs = champs + cha + ","
17#         val = input("Entrez le champ %s :" % nom)
18#         if type =="i":
19#           val =int(val)
20#         valeurs.append(val)
21#
22#       balises ="(" + "%s," * len(valeurs)      #علامات التحويل
23#       champs = champs[:-1] + ")"   #حذف الفاصلة الأخيرة,
24#       balises = balises[:-1] + ")"  #و إضافة قوس
25#       req ="INSERT INTO %s %s VALUES %s" % (self.table, champs, balises)
26#       self.bd.executerReq(req, valeurs)
27#
28#       ch =input("Continuer (O/N) ? ")
29#       if ch.upper() == "O":
30#         return 0
31#       else:
32#         return 1
```

تعليقات

* السطور من 1 إلى 6 : في وقت التمثيل, كائنات من هذا الصنف سوف يتلقى مرجع واحدة من جداول القاموس ز هذا ما يتيح لهم الوصول إلى وصف الحقول .

* السطر 8 : الأسلوب **enter()** يولد نموذج نفسه . وهو يدعم سجلات الإدخال في الجداول, من خلال التكيف مع هيكل الخاص بها من خلال وصف الموجود في القاموس . وظيفتها عملية بينة مرة أخرى قطعة قطعة سلسلة نصية التي ستصبح إستعلام SQL, كما في الأسلوب **creerTables()** للصنف **GestionBD()** الموضح سابقا . يمكنك بالطبع إضافة إلى هذا الصنف أساليب أخرى, لمعالجة على سبيل المثال حذف و\أو تعديل السجلات .

* السطور من 12 إلى 20 : سمة المثيل **self.descriptif** تحتوي على قائمة و أنفاق, و كل واحد منها يتكون من 3 عناصر, و هي إسم الحقل و نوع البيانات التي سوف يتلقاها, و الوصف "بوضوح" . الحلقة for في السطر 13 تقوم بتكرارا هذه القائمة و عرض لكل حقل رسالة موجهة

على الأساسا لوصف الذي يصاحب هذا الحلق . عندما يقوم المستخدم بإدخال القيم المطلوبة, سبتم حفظها في قائمة المنشئ, في حين يتم إضافة إسم الحقل إلى تنسيق السلسلة .

* السطور من 22 إلى 26 : عندما يتم تكرار جميع الحقول, يتم تجميع الإستعلام و تنفيذه . كما شرحنا في الصفحة 285, لا ينبغي أن تكون القيم المدرجة في السلسلة الإستعلام نفصها, لكن تم تمريرها كبرامتر للأسلوب **execute()** .

جسم التطبيق

لا يبدو مفيدا تطوير المزيد من هذا التمرين في الإطار اليدوي للشروع . إذا كنت مهتما, يجب أن نعرف الأن ما يكفي بدء أي تجاربك الشخصية . يرجى الإطلاع كتب مراجع, كما على سبيل المثال : How to program - كيف تبرمج ل Deitel و coll, أو مواقع متخصصة لملحقات البايثون .

السكريبت التالي هو تطبيق صغير مصم لإختبار أصناف التي تم وصفها في الصفحات السابقة . لا تتردد في تحسينها, ثم أكتب واحدة أخرى مختلفة تماما !

```
1#      ###### البرنامج الرئيسي : #########
2#
3#      # صنع كائن واجهة مع قاعدة البيانات :
4#      bd = GestionBD(Glob.dbName, Glob.user, Glob.passwd, Glob.host, Glob.port)
5#      if bd.echec:
6#        sys.exit()
7#
8#      while 1:
9#        print("\nQue voulez-vous faire :\n"\
10#          "1) Créer les tables de la base de données\n"\
11#          "2) Supprimer les tables de la base de données ?\n"\
12#          "3) Entrer des compositeurs\n"\
13#          "4) Entrer des oeuvres\n"\
14#          "5) Lister les compositeurs\n"\
15#          "6) Lister les oeuvres\n"\
16#          "7) Exécuter une requête SQL quelconque\n"\
17#          "9) terminer ?                    Votre choix :", end=' ')
18#        ch = int(input())
19#        if ch ==1:
20#          # صنع جميع الجداول الموصوفة في القاموس
21#          bd.creerTables(Glob.dicoT)
22#        elif ch ==2:
23#          # إزالة صنع جميع الجداول الموصوفة في القاموس :
24#          bd.supprimerTables(Glob.dicoT)
25#        elif ch ==3 or ch ==4:
26#          # صنع <enregistreur> لملحنين أو ... :
27#          table ={3:'compositeurs', 4:'oeuvres'}[ch]
28#          enreg =Enregistreur(bd, table)
29#          while 1:
30#            if enreg.entrer():
31#              break
32#        elif ch ==5 or ch ==6:
33#          # قائمة كل الملحنين, ou toutes les oeuvres :
34#          table ={5:'compositeurs', 6:'oeuvres'}[ch]
35#          if bd.executerReq("SELECT * FROM %s" % table):
36#            # : تحليل نتيجة الإستعلام أدناه
37#            records = bd.resultatReq()      # سيكون نفق من الأنفاق
38#            for rec in records:          # => كل سجل
39#              for item in rec:          # => كل حقل للسجل
40#                print(item, end=' ')
41#              print()
42#        elif ch ==7:
43#          req =input("Entrez la requête SQL : ")
44#          if bd.executerReq(req):
45#            print(bd.resultatReq())        # سيكون نف من الأنفاق
46#        else:
47#          bd.commit()
48#          bd.close()
49#          break
```

تعليقات

* يجب أن نفترض بالطبع أن الأصناف المذكورة أعلاه موجودة في نفس السكريبت, أو تم إستدعائه .

* السطور من 3 إلى 6 : يتم إنشاء كائن-الواجهة هنا . فإذا فشل, سمة المثيل **bd.echec** ستكون قيمتاه 1 . و سطور 5 و 6 يسمحون بإغلاق التطبيق فورا (الدالة **exit()** لوحدة **sys** تستخدم خصيصا لهذا) .

* السطر 8 : ما تبقى من التطبيق هو تقديم نفس القائمة بإستمرار إلى أن يختار المستخدم الخيار رقم 9 .

* السطور 27 و 28 : الصنف **Enregistreur()** يقبل معالجة السجلات من أي جدول , لتحديد أي يجب أن تستخدم عند التمثيل, إستخدم قاموس صغير الذي يشير إلى إسم الحفظ, و هذا يتوقف على الإهتيار الذي أدلى به المستخدم (الخيار رقم 3 أو رقم 4) .

* السطور من 29 إلى 31 : الأسلوب **entrer()** لكائن-الحافظ يقوم بإرجاع القيمة 0 أو 1 إعتمادا إذا كان إختيار المستخدم مواصلة إدحال السجلات, أو التوقف . إن إختبار هذه القيم تسمح بوقف حلقة التكرار وفقا لذلك .

* السطور من 35 إلى 44 : يقوم الأسلوب **executerReq()** بإرجاع قيمة 0 أو 1 إعتمادا على ما إذ تم قبول الإستعلام أو لا من قبل الخادم . يمكننا إختبار هذه القيمة لنقرر إذا كان الناتج يجب عرضه أو لا .

تمارين

16.2 قم بتعديل السكريبت الموضح في هذه الصفحات من أجل إضافة جدول إضافي إلى قاعدة البيانات . يمكن أن يكون على سبيل المثال "**orchestres** - عصابات", الذي يحتوي كل سجل منها على إسم المجرم و زعيمه و العدد الإجمالي للصكوك .

16.2 قم بتعديل أنواع أخرى من حقو إلى أحد الجداول (على سبيل المثال حقل من نوع حقيقي أو نوع **date** - تاريخ), و قم بتغيير السكريبت وفقا لذلك .

17

تطبيقات الويب

ربما قد تعلمت سابقا أشياء كثيرة عن كتابة صفحات الويب .هل تعلم أن هذه الصفحات هي عبارة عن مستندات مكتوبة بلغة الهتمل و التي يمكن الوصول إليها عن طريق الإنترنات بإستخدام برامج خاصة تسمى بالمتصفحات (مثل فايرفوكس و إنترنات إكسبلورر و سفاري و أوبرا و غاليون و كونكيورر ...) .يتم تثبيت صفحات الهتمل في دلائل عامة (فهرسات عامة) من حاسوب أخر يشغل بشكل مستمر برنامج يدعى خادم الويب (مثل أباتشي و لايت هتتبد و زيتامي و إإس) و عندما يتم تأسيس إتصال بين الحاسوب و حاسوبك ,يمكن للمتصفح التفاعل مع برنامج الخادم عن طريق إرسال طلبات (عن طريق مجموعة متنوعة من الأجهزة و البرمجيات التي لن تتم مناقشتها هنا مثل : خطوط الهاتف و الموجهات (الراوترات) و بروتوكولات الإتصال ...) و يعرض المتصفح النتائج في إستجابة لهذه الطلبات .

صفحات ويب تفاعلية

بروتوكول http الذي يدير إنتقال من صفحات الويب يتيح تبادل البيانات في كلا الإتجاهين . و لكن إذا كان الموقع بسيطا (لا تتفاعل مع الموقع من خلال إدخال أشياء إلخ)يقام نقل البيانات في إتجاه واحد من اثنين , و هو الخادم إلى المتصفح : النصوص و الصور و الملفات المختلفة التي يتم إرسالها إلى المتصفح بأعداد كبيرة (هذه هي الصفحات التي يتم تصفحها) و مع ذلك , يرسل المتصفح إلى الخادم كميات ضئيلة جدا من المعلومات إلى الخادم : في الأساس عناوين الصفحات التي يرغب المستخدم في التفاعل .

و أنت تعرف أن هنالك العديد من المواقع حيث يطلب منك المزيد من المعلومات (أن تتفاعل أكثر) مثل : مراجع الشخصية للتسجيل في النادي أو حجز في فندق , ومن المعلومات التي سترسلها هي

مثلا : رقم هاتفك أو رقم بطاقة الائتمان لطلب بندا على موقع للتجارة الإلكترونية أو أرائك أو إقتراحاتك , إلخ .. .

في هذه الحالات , يمكنك أن تتخيل أن المعلومات المعتمدة ستنقل إلى جانب الخادم من خلال برنامج محدد. بالتالي يجب ربطها بهذا البرنامج عن بعد في

الخادم. أما بالنسبة لصفحات الويب مصممة لاستيعاب هذه المعلومات (وتسمى النماذج)، وسوف توفر لهم ترميز الحاجيات المختلفة (حقول الإدخال، خانات ومربعات القوائم، الخ.)، بحيث يمكن للمتصفح تقديم طلب إلى خادم جنبا إلى جنب مع وسائط.

إذا يمكن للخادم التعامل مع هذه المدخلات ببرنامج معالج (يعني يتعامل مع المدخلات) خاص و إعتمادا على هذه النتائج يرسل البرنامج جواب مناسب للمستخدم تحت شكل صفحة ويب جديد (معناه الإنتقال إلى صفحة أخرى بها الإجابة مثل صفحة تم إنهاء التسجيل) .

هنالك طرق مختلفة لتقوم بعمل مثل هذه البرامج الخاصة و التي نسميها الأن تطبيقات الويب .

واحدة من الطرق الأكثر شعبية في الوقت الحاضر هو إستخدام صفحات هتمل "تم إثرائها" بإستخدام برامج نصية مكتوبة (السكريبتات) بمساعدة لغات برمجة معينة مثل بي أتش بي . يتم إدراج هذه السكريبتات مباشرة في شيفرة الهتمل بين وسوم خاصة و سيتم تنفيذها من قبل خادم الويب (و على سبيل المثال أباتشي) على أن يتم تجهيز هذا الكود مع وحدة المترجم المناسبة . يمكنك أن تفعل ذلك مع البايثون عبر صيغة معدلة بشكل طفيف من لغة تسمى PSP (صفحات خادم بايثون) .

هذا المنهج لديه عيب و هو خلط أكواد الهتمل و من هذه الخلوطات(جمع كلمة خلط) التلاعب بخلط أجزاء من سكريبتات بي أتش بي أو بي أس بي داخل الوسوم مما يعرض لمشاكل عند القراءة بشكل عام.

و أفضل أسلوب هو العمل على كتابة النصوص (بشكل منفصل) التي تولد كود هتمل كلاسيكية في شكل سلاسل و توفير وحدة (موديل) على خادم الويب لتفسير هذه النصوص و تعود ب كود هتمل ردا على إستفسارات المتصفح (على سبيل المثال **mod_python** في حالة أباتشي)

و لكن مع البايثون ,يمكننا دفع هذا النوع من المنهج إلى أبعد من ذلك من خلال تطوير بأنفسنا خادم ويب حقيقي متخصص , مستقل تماما , في واحدة تحتوي على وظائف برنامج الخاصة المطلوبة

لتطبيقنا . نستطيع القيام بذلك مع البايثون لأنها بنيت كافة المكتبات اللازمة لإدارة بروتوكول HTTP إلى اللغة . على هذا الأساس قد أنتجت العديد من المبرمجين المستقلين أيضا و أتاحة للمجتمع مجموعة من الأدوات التطوير لتسهيل تطوير مثل هذه التطبيقات الويب . في الفترة المتبقية من دراستنا , سوف نستخدم واحدة منهم . إخترنا CherryPy لأنه يبدو جيدا و مناسب مع أهداف هذا العمل .

مهم

> الذي سنشرحه في الفقرات التالية سيكون عمليا مباشرة على الإنترنات من مدرستك أو في شركتك . فيما يتعلق بسلبيات الأنترنات الأمور القليلة أكثر تعقيدا , بالإضافة إلى أن تثبيت الرنامج على حاسوب خادم (سيرفر) متصل بشبكة الأنترنات لا يمكن أن يتم إلا بموافقة مالكها . إذا كان المزود قد أتاح لك مساحة حيث يسمح لك بتثبيت صفحات ويب ساكنة (و هذا يعني بعض الوثائق البسيطة للإستشارة) هذا لا يعني أنك لن تكون قادر على تشغيل برامج ! من أجل تشغيلها سيكون من الضروري أن تحصل على تصريح و عدد من المعلومات إلى المزود .معظمهم يرفضون السماح لتثبيت تطبيقات مستقلة عن النوع الذي وصفناه أدناه , و لكن يمكنك بسهولة تخويلها بحيث تكون صالحة للإستخدام أيضا مع *mod_python* لأباتشي اتي عموما موجودة[85].

خادم ويب في بايثون نقية

لقد أصبح الاهتمام في تطوير الشبكة مهم جدا في عصرنا الحاضر، وهناك طلب قوي على واجهات البرمجة وبيئات مناسبة تماما لهذه المهمة. لكن حتى لو انه لا يمكن مطالبة لغات عالمية مثل C / C + +، وبالفعل بايثون على نطاق واسع في جميع أنحاء العالم لكتابة البرامج الطموحة، بما في ذلك في مجال خوادم التطبيقات على شبكة الإنترنت. وقد اجتذبت متانة وسهولة التنفيذ من لغة للمطورين

[85] يرجي الرجوع إلى أعمال أكثر تخصص مثل Cherrypy Essentials, par Sylvain Hellegouarch, Packt Publishing, Birmingham, 2007, كتاب مرجعي عن Cherrypy .

العديد من الموهوبين الذين جعلوا من أدوات تطوير مواقع الويب إلى أعلى مستوى. قد العديد من هذه التطبيقات تهمك إذا كنت تريد أن تفعل ذلك بنفسك مواقع الويب التفاعلية من مختلف الأنواع.

المنتجات الحالية هي في معظمها البرمجيات الحرة. فإنها يمكن أن تغطي مجموعة واسعة من الاحتياجات، من موقع شخصي صغير من بضع صفحات على موقع تجاري كبير التعاونية، وقادرا على الإجابة على آلاف الطلبات يوميا، ومختلف القطاعات التي تدار من قبل أشخاص من دون تدخل من مختلف المهارات (مصممي الغرافيك والمبرمجين، وقاعدة بيانات، الخ ...).

الأكثر شهرة من هذه المنتجات هو برنامج Zope، التي سبق اعتمادها من قبل كبرى المنظمات الخاصة والعامة لتطوير الشبكات الداخلية والخارجية التعاونية. هذا هو في الواقع خادم تطبيق النظام، والأداء العالي، وآمنة، بشكل كامل تقريبا مكتوب في بايثون، ويمكن أن تدار عن بعد باستخدام واجهة ويب بسيطة. فمن غير الممكن وصفنا استخدام Zope في هذه الصفحات: هذا الموضوع هو واسعة جدا، والكتاب بأكمله لا تكفي. تكون على علم بأن هذا المنتج هو قادر تماما على التعامل مع مواقع الشركات الكبيرة جدا وتوفير مزايا كبيرة على حلول أفضل المعروفة مثل PHP أو جافا.

من الأدوات أخرى أقل طموحا ولكن مثيرة للاهتمام على حد سواء المتاحة. مثل Zope، معظمهم يمكن أن تحملهم مجانا من الإنترنت. حقيقة أن هي مكتوبة في بايثون كما يضمن قابلية لديها، بحيث يمكنك استخدامها على عدة أنظمة ويندوز أو لينكس أو ماك على حد سواء . ويمكن استخدام كل بالتزامن مع خادم الويب "كلاسيكي" مثل أباتشي أو Xitami (وهو أيضا من جيد إذا كان الغرض من موقع لتحقيق حمولة من الروابط المهمة على شبكة الإنترنت)، ولكن معظمهم تشمل الخادم الخاص بهم، مما يسمح لهم العمل على ما يرام بشكل مستقل تماما. هذا الاحتمال المثير للاهتمام بشكل خاص في تطوير الموقع، لأنه يسهل استكشاف الأخطاء وإصلاحها.

الحكم الذاتي الكامل وسهولة التنفيذ جعل هذه المنتجات حلول جيدة لتحقيق مواقع الإنترانت المتخصصة، بما في ذلك الشركات الصغيرة والمتوسطة، أوالحكومات، أو المدارس. إذا كنت ترغب في بناء تطبيق بايثون التي يتم الوصول إليها عن بعد عبر متصفح الإنترنت، فهذه الأدوات قد صنعت لك. هناك مجموعة واسعة:

جانغو، Turbogears، أبراج، CherryPy، Webware، Karrigell، SPYCE، كيخوته، ملتوية، الخ.[86]. اختار وفقا لحاجاتك، وسوف يكون لك مجموعة كبيرة للاختيار منها .

في ما يلي، ونحن نصف خطوة خطوة لتطوير تطبيق ويب يشغيل باستخدام CherryPy. الذي يمكنك أن تجده في موقع :

http://www.cherrypy.org. هو يجد الحل لتطوير الشبكة لمبرمجي بايثون ودية للغاية، لأنه يسمح له لتطوير موقع على شبكة الانترنت كتطبيق بايثون الكلاسيكية، واستنادا إلى مجموعة من الكائنات. هذه توليد كود HTML استجابة لطلبات HTTP الموجهة إليها عبر أساليبها، وينظر إلى هذه الأساليب ب عناوين العادي من قبل المتصفحات.

> لبقية هذا النص, فإننا نفترض أن لديك بعض من أساسيات *HTML*, و نحن نفترض أيضا أنه تم تثبيت مكتبة *CherryPy* على محطة العمل الخاصة بك (تم وصف هذا التبيت في المرفق صفحة *Error: Reference source not found*).

أول مشروع: إطلاق الموقع على شبكة الإنترنت من الحد الأدنى من مكونات الصفحة.

في مجلد العمل الخاص بك، وإصنع ملف نصي صغير وسمه **tutoriel.conf**، والتي سوف يحتوي على التالي:

```
[global]
server.socket_host = "127.0.0.1"
server.socket_port = 8080
server.thread_pool = 5
tools.sessions.on = True
tools.encode.encoding = "Utf-8"
[/annexes]
tools.staticdir.on = True
tools.staticdir.dir = "annexes"
```

هذا هو ملف بسيط للإعدادات يجعل خادم الويب CherryPy لدينا يتشاور عند بدء التشغيل. لاحظ رقم المنفذ (8080 في مثالنا). ولعلكم تعلمون أن المتصفحات برامج تنتظر لتجد الخدمات على

---

[86] في وقت كتابة هذه السطور, CherryPy أتيح للتو نسخة 3 من البايثون . من بين الأدوات الأخرى المذكورة هنا, العديد منها يجري تكييفها, لكنها على أي حال قابلة للإستخدام تماما مع الإصدارات السابقة من البايثون .

شبكة الإنترنت من المنفذ 80 افتراضيا. إذا كنت صاحب الجهاز، ولا تريد تثبيت أي برنامج خادم ويب أخر، لذلك ربما كنت من أفضل أن تحل محل 8080 ب 80 في ملف التكوين هذا :
والمتصفحات التي سيتم الاتصال بملقم يجب أن يتم تحديد رقم منفذ في العنوان. ومع ذلك، إذا كنت تفعل هذه التمارين على جهاز لم تكن أنت المسؤول عنه ، يعني لم يكن لديك الحق في استخدام أرقام المنافذ مثل 1024 (لأسباب أمنية). في هذه الحالة، يجب استخدام رقم منفذ أعلى من 80، مثل التي وضعناها . وهذا ينطبق حتى لو كان خادم الويب آخر(اباتشي، وعلى سبيل المثال) قيد التشغيل على الجهاز الخاص بك، لأن هذا البرنامج يستخدم على الأرجح المنفذ 80 بالفعل افتراضيا.

السطر **server.thread_pool = 5** يشير إلى عدد من المواضيع خادم CherryPy ستفتح بالتوازي لمعالجة طلبات من نفس الوقت لمختلف المستخدمين. المواضيع هي "ابن" من التنفيذ المتزامن للبرنامج: سنضع شرحها في الفصل 19.

لاحظ أيضا خط الترميز. هذا هو الترميز الذي سوف نتستخدم في CherryPy في إنتاج صفحات الويب. فمن الممكن أن بعض المتصفحات الأخرى نتوقع أن UTF-8 هوالترميز الافتراضي. اذا كنت تحصل على أحرف غير صحيحة في المتصفح الخاص بك (في بعض الأحيان على شكل مربعات) عندما تواجهت هذه المشكلة الموضحة أدناه، وكرر المحاولة عن طريق تحديد الترميزات المختلفة في هذا الخط.

السطور الثلاثة الأخيرة في الملف تشير إلى مسار الدليل حيث تقوم بوضع 'ثابت' الوثائق التي قد يحتاج موقعك (صور، الشكل، الخ.).

الأن أكتب ملف السكريبت أدناه :

```
1#    import cherrypy
2#
3#    class MonSiteWeb(object):        # الصنف السيدج للتطبيق
4#       def index(self):          # أسلوب يتم إستدعائه كجذر (/)
5#          return "<h1>Bonjour à tous !</h1>"
6#       index.exposed = True        # الأسلوب يجب أن يكون "منشور"
7#
8#    ###### البرنامج الرئيسي : ##############
9#    cherrypy.quickstart(MonSiteWeb(), config ="tutoriel.conf")
```

شغل البرنامج النصي(السكريبت) . إذا كان كل شيء على ما يرام، ستحصل على بعض سطور من معلومات مشابهة لما يلي في طرفيتك. فإنها تؤكد أن "شيئا ما" بدأ، وينتظر الأحداث:

```
[07/Jan/2010:18:00:34] ENGINE Listening for SIGHUP.
[07/Jan/2010:18:00:34] ENGINE Listening for SIGTERM.
[07/Jan/2010:18:00:34] ENGINE Listening for SIGUSR1.
[07/Jan/2010:18:00:34] ENGINE Bus STARTING
[07/Jan/2010:18:00:34] ENGINE Started monitor thread '_TimeoutMonitor'.
[07/Jan/2010:18:00:34] ENGINE Started monitor thread 'Autoreloader'.
[07/Jan/2010:18:00:34] ENGINE Serving on 127.0.0.1:8080
[07/Jan/2010:18:00:34] ENGINE Bus STARTED
```

كانت في الواقع مجرد أسطر لبدء خادم الويب!

سوف نتأكد فقط من أنها تعمل بشكل جيد، وذلك باستخدام متصفحك المفضل. إذا كنت تستخدم هذا المتصفح على نفس الجهاز كملقم، أشر نحو عنوان مثل http://localhost:8080، لوكل هوست هو مصطلح يستخدم لوصف الجهاز المحلي (يمكنك أيضا تحديد ذلك في باستخدام عنوان IP التقليدية: 127.0.0.1)، و 8080 رقم المنفذ المحدد من ملف configuration[87]. يجب أن تشاهد الصفحة الرئيسية التالية:

يمكنك أيضا الوصول إلى نفس الصفحة الرئيسية من جهاز آخر, عن طريق توفير لمتصفحه عنوان IP أو إسم السيرفر الخاص بك على الشبكة المحلية, بدلا من localhost .

دعونا الأن نجرب السكريبت الخاص بنا قليلا . الإيجاز ملفت للنظر : فقط 6 أسطر فعالة !

بعد إستدعاء وحدة cherrypy, قمنا بتعريف صنف جديد **MonSiteWeb()** . الكائنات المنتجة بإستخدام هذا الصنف تكون معالجات الطلبات(الإستعلامات) . يتم إستدعاء أساليبها من خلال Cherrypy الداخلية, التي تحول عنوان URL المطلوب من قبل المتصفح, نطلب الأسلوب مع الإسم المعادل (سوف نوضح هذه الآلية على نحة أفضل مع المثال التالي) . إذا كان عنوان URL المستلم لا يحتوي على أي إسم صفحة, كما في حالتنا, سيتم البحث عن إسم الفهرس بشكل إفتراضي, بعد إتفاقية راسخة على شبكة الإنترنات . و لهذا السبب فإن إسمه أسلوب فريد من نوعه, الذي ينتظر الطلبات التي تعالج جذر الموقع .

* السطر 5 : أساليب هذا الصنف ستقوم بمعالجة الطلبات من المتصفح, و إرجاع رد كسلسلة نصية تحتوي على نص مكتوب بلغة HTML . لهذا التمرين الأول, قمنا بتبسيط كود HTML المنتج إلى أقصى حد, تم تلخيص ذلك في رسالة صغيرة بين علامات العنوان (<h1> و <h1/>) . بعبارة أدق, ينبغي أن تدرج كل

[87] إذا إخترت رقم المنفذ الإفتراضي (80) في ملف التكوين, يجدر التذكير في العناوين, أن هذا هو منفذ الذي يتم إستخدامه بشكل إفتراضي في معظم المتصفحات . يمكنك في هذه الحالة الإتصال بموقعك الجديد عن طريق إدخال ببساطة : *http://localhost* .

شيئ بين <html></html> و <body></body> من أجل تحقيق تخطيط صحيح . و لكن بما أنه يعمل, سوف ننتظر قليلا بعد قبل أن نريك طرق جيدة .

* السطر 6 : الأساليب التي ستقوم بمعالجة طلبات HTTP و تقوم بإرجاع صفحة ويب, يجب أن تكون "منشورة" بإستخدام سمة **exposed** التي تحتوي على قيمة "صحيح" .يجب أن نأمن سلامة تنفيذ CherryPy, والذي يتم إفتراضيا, جميع الأساليب التي كتبتها سوق يتم حمايتها وجها لوجه من يحاول الوصول إليها . الأساليب الوحيدة المتاحة هي التي يجب علينا qui auront été délibérément rendues publiques à l'aide de cet attribut.

* السطر 9 : الدالة **quickstart()** لوحدة cherrypy تقوم بتشغيل الخادم الفعلي . يجب أن يتم توفير برامتر مرجع كائن معالج الطلبات الذي سيكون جذر الموقع, و مرجع الملف التكوين العام .

إضافة صفحة ثانية

نفس كائن المعالج يمكن بالطبع أن يأخذ عدة صفحات :

```
1#    import cherrypy
2#
3#    class MonSiteWeb(object):
4#
5#       def index(self):
6#          #تحتوي على رابط إلي صفحة أخرى HTML إرجاع صفحة
7#          #سيتم إنشائها من أسلوب اخر من نفس الكائن):
8#          return '''
9#             <h2>Veuillez <a href="unMessage">cliquer ici</a>
10#            pour accéder à une information d'importance cruciale.</h2>
11#         '''
12#      index.exposed = True
13#
14#      def unMessage(self):
15#         return "<h1>La programmation, c'est génial !</h1>"
16#      unMessage.exposed = True
17#
18#    cherrypy.quickstart(MonSiteWeb(), config ="tutoriel.conf")
```

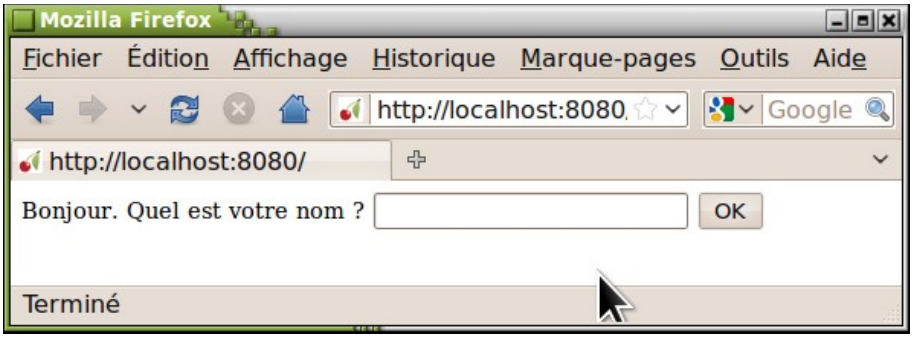
هذا السكريبت يشتق من سابقه . الصفحة تقوم بإرجاع بواسطة الأسلوب index() التي تحتوي هذه المرة على علامة-رابط <"a href="unMessage> برامتر URL لصفحة أخرى . إذا كان هذا URL إسمه بسيطة, من المفترض أن تكون الصفحة موجودة في الدليل الجذر للموقع . في منطق تحويل ال URL التي يتم إستخدامها من قبل Cherrypy, فإنها تقوم بإستدعاء أسلوب كائم الجذر مع إسم معادل . في مثالنا, صفحة المرجع سيتم إنتاجها من قبل الأسلوب **unMessage()** .

عرض و معالجة نموذج

و سوف تصبح الأمور مثيرة للإهتمام أكثر مع السكريبت التالي :

```
1#    import cherrypy
2#
3#    class Bienvenue(object):
4#      def index(self):
5#        #: نموذج طلب إسم من المستخدم
6#        return '''
7#          <form action="salutations" method="GET">
8#          Bonjour. Quel est votre nom ?
9#          <input type="text" name="nom" />
10#         <input type="submit" value="OK" />
11#         </form>
12#       '''
13#     index.exposed = True
14#
15#     def salutations(self, nom =None):
16#       if nom:        #: صفحة الرئيسية للمستخدم
17#         return "Bonjour, {0}, comment allez-vous ?".format(nom)
18#       else:          #: لم يتم كتابة أي إسم
19#         return 'Veuillez svp fournir votre nom <a href="/">ici</a>.'
20#     salutations.exposed = True
21#
22#   cherrypy.quickstart(Bienvenue(), config ="tutoriel.conf")
```

الأسلوب **index()** لكائن الجذر الخاص بنا أصبح يعرض هذه المرة للمستخدم صفحة ويب تحتوي على نموذج : كود HTML تم وضعه داخل علامات <form> و </form> و الذي قد يحتوي على مجموعة من الويدجات المختلفة, و التي يمكن للمستخدم التعامل معها , يمكنها أن تقوم بترميز المعلومات و أن تنفذ تفعيل نشاط معين : حقول الإدخال و خانات و أزرار راديو و علب قوائم و إله . لهذا المثال الأول, حقل إدخال و زر كافي :



* السطر 7 : العلامة <form> يجب أن تحتوي على إثنين من التفاصيل الأساسية : العمل الذي ستقوم به عند إرسال النموذج (إنها في الواقع توفير URL لمصدر ويب قادر على تلقي إستعلام مع البرامترات), و الأسلوب (GET أو POST) تستخدم لتمرير هذه البرامترات .

> الفرق بين *GET* و *POST* هو كيفية ربط البرامترات للإستعلام, في الرأس (*GET*), أو في ملحق (*POST*) . ل *CherryPy*, هذا التمييز غير مهم . يمكنك إستخدام أي واحدة .

* السطر 9 يحتوي على علامات HTML التي تعرف حقل الإدخال (العلامة input type="text"> name="nom" />) . سمة name تسمح بربط ملصق بسلسلة نصية التي سوف تقوم بترميزها من قبل المستخدم . عندما يقوم المتصفح بتمرير إستعلام HTTP للخادم, و هذه سوف تحتوي على برامتر موصوف بشكل جيد . كما سبق و أوضحنا أعلاه, يقوم Cherrypy بتحويل هذا الإستعلام بإستدعاء أسلوب كلاسيكي, و التي ترتبط مع وصف البرامتر, كالعادة في البايثون .

* السطر 10 يعرف ويدجت من نوع "زر إرسال" (العلامة **<"input type="submit>**) . النص الذي سيتم عرضع على الزر عن طريق سمة **value** .

* السطور من 15 إلى 20 يتم تعريف الأسلوب الذي سيقبل الإستعلام, عندما يتم إرسال النموذج إلى الخادم . برامتر **nom** سوف تتلقى البرامتر المناسب, التي يعرفها من إسمها(هشام إستبدل الوصف بالإسم) نفسه . كالعادة في البايثون, يمكنك تحديد قيم إفتراضية لكل برامتر (إذا ترك حقل النموذج فارغ من قبل

المستخدم, لا يتم تمرير البرامتر) . في مثالنا, البرامتر **nom** يحتوي إفتراضيا كائن فارغ : سيكون من السهل جدا التأكد برمجيا ما إذا قام المستخدم بإدخال إسم فعلا أو لا .

عمل هذه الآليات هي طبيعة جدا و بسيطة جدا : يتم تحويل ال URL الذي تم إستدعائه من خلال Cherrypy بإستدعاء أساليب تحمل نفس الإسم, و التي تمرر البرامترات بطريقة كلاسيكية .

تحليل الإتصالات و الأخطاء

عند تجربة السكريبتات التي تم شرحها حتى هنا, ستلاحظ أن رسائل مختلفة تظهر في نافذة الطرفية أين بدأ التنفيذ . هذه الرسائل تبلغكم(جزئيا) على حوار الذي جرى بين الخادم و العملاء . يمكنك تأسيس إتصال مع خادمك من أجهزة أخرى (إذا كان خادمك متصل بشبكة, بطبيعة الحال) :

```
[12/Jan/2010:14:43:27] ENGINE Started monitor thread '_TimeoutMonitor'.
[12/Jan/2010:14:43:27] ENGINE Started monitor thread 'Autoreloader'.
[12/Jan/2010:14:43:27] ENGINE Serving on 127.0.0.1:8080
[12/Jan/2010:14:43:27] ENGINE Bus STARTED
127.0.0.1 - - [12/Jan/2010:14:43:31] "GET / HTTP/1.1" 200 215 "" "Mozilla/5.0
(X11; U; Linux i686; fr; rv:1.9.1.6) Gecko/20091215 Ubuntu/9.10 (karmic)
 Firefox/3.5.6"
127.0.0.1 - - [12/Jan/2010:14:44:07] "GET /salutations?nom=Juliette HTTP/1.1"
 200 39 "http://localhost:8080/" "Mozilla/5.0 (X11; U; Linux i686; fr;
 rv:1.9.1.6) Gecko/20091215 Ubuntu/9.10 (karmic) Firefox/3.5.6"
```

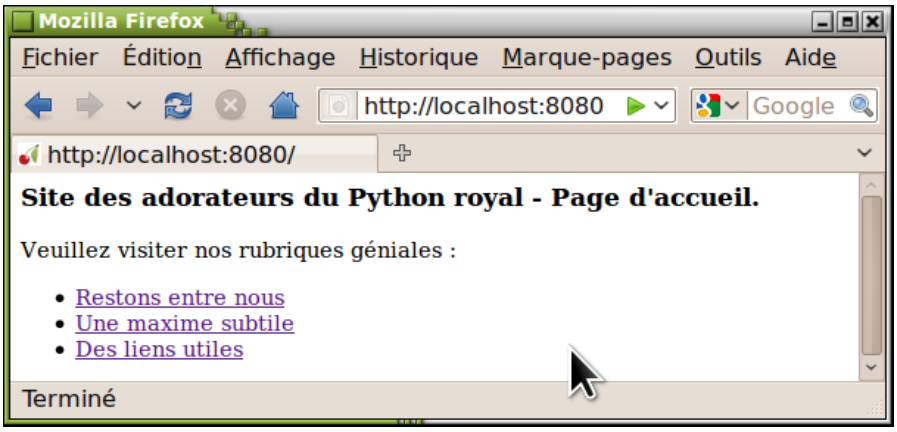
في نافذة الطرفية حين تجد رسائل الخطأ (على سبيل المثال, أخطاء في تركيب الجملة) التي تتصل ببرنامج في نهاية المطاف قبل بدئ تشغيل الخادم . فإذا تم الكشف عن خطأ أثناء العمل (خطأ في أسلوب معالج الإستعلامات), رسالة الخطا تظهر في نافذة المتصفح, و الخادم يواصل عمله . الشكل في الصفحة 311 تظهر على سبيل المثال رسالة التي حصلنا عليها من خلال إضافة خطأ لإسم أسلوب **()format**, في السطر 17 من السكريبت الخاص بنا(**formate(nom)** بدلا من **format(nom)**) .

يمكنك التحقق من أن الخادم يعمل دائما, من خلال العودة إلى الصفحة السابقة و إدخال هذه المرة إسم فارغ . هذا الخيار لا يمنع عنددما يتم الكشف عن خطأ في غاية الأهمية لخادم الويب, لأن هذا سوف يسمح بالإستمرار في تلبية معظم طلبات التي يتلقاها, حتى لو كان ينبغي أن يرفض بعضها لأن لا يزال هنالك بعض مشاكل الصغير في البرنامج . و هذا قد يبدو من الوهلة الأول, لأنه حتى الأن ستجد دئما أن خطأ في وقت التنفيذ يسبب في إغلاق البرنامج .

هذا لا يحدث هذه المرة, لأن عند بدأ تشغيل تطبيقنا, Cherrypy يطلق عدة أسطر تنفيذ البرنامج و التي نسميها **threads** (مواضيع), و واحد فقط تم إقافه بالخكطأ . و سيتم شرح المزيد من **threads**-المواضيع بالتفصيل في الفصل 19 (صفحة Error: Reference source not found).

هيكل موقع متعدد الصفحات

سوف نرى الأن كيفية هيكلة موقعنا, و إنشاء تسلسل هرمي للأصنفا بطريقة مماثلة لتلك بين الدلائل و الدلائل الفرعية في نظام الملفات .



في السكريبت أدناه, سوف يتولى إهتمام خاص بتعريف العلامات-الوصلات <a href=... > :

```
1#    import cherrypy
2#
3#    class HomePage(object):
4#      def __init__(self):
5#        # كائنات معالجة للطلبات(إستعلامات) يمكنها أن تقوم بتمثيل نفسها كمعالجات أخرى "عبيدة", و هكذا
6#        self.maxime = MaximeDuJour()
7#        self.liens = PageDeLiens()
8#        # يمكن بالطبع أن تقوم بتمثيل كائنات معالجة الإستعلامات في أي مستوى من البرنامج .
9#
10#     def index(self):
11#       return '''
12#         <h3>Site des adorateurs du Python royal - Page d'accueil.</h3>
13#         <p>Veuillez visiter nos rubriques géniales :</p>
14#         <ul>
15#           <li><a href="/entreNous">Restons entre nous</a></li>
16#           <li><a href="/maxime/">Une maxime subtile</a></li>
17#           <li><a href="/liens/utiles">Des liens utiles</a></li>
18#         </ul>
19#       '''
20#     index.exposed = True
21#
22#     def entreNous(self):
23#       return '''
24#         Cette page est produite à la racine du site.<br />
```

```
25#         [<a href="/">Retour</a>]
26#       '''
27#     entreNous.exposed =True
28#
29#   class MaximeDuJour(object):
30#     def index(self):
31#       return '''
32#         <h3>Il existe 10 sortes de gens : ceux qui comprennent
33#         le binaire, et les autres !</h3>
34#         <p>[<a href="../">Retour</a>]</p>
35#       '''
36#     index.exposed = True
37#
38#   class PageDeLiens(object):
39#     def __init__(self):
40#       self.extra = LiensSupplementaires()
41#
42#     def index(self):
43#       return '''
44#       <p>Page racine des liens (sans utilité réelle).</p>
45#       <p>En fait, les liens <a href="utiles">sont plutôt ici</a></p>
46#       '''
47#     index.exposed = True
48#
49#     def utiles(self):
50#       # لاحظ كيف تم تعريف الإرتباط إلى الصفحات الأخرى :
51#       # on peut procéder de manière ABSOLUE ou RELATIVE.
52#       return '''
53#         <p>Quelques liens utiles :</p>
54#         <ul>
55#           <li><a href="http://www.cherrypy.org">Site de CherryPy</a></li>
56#           <li><a href="http://www.python.org">Site de Python</a></li>
57#         </ul>
58#         <p>D'autres liens utiles vous sont proposés
59#         <a href="./extra/"> ici </a>.</p>
60#         <p>[<a href="../">Retour</a>]</p>
61#       '''
62#     utiles.exposed = True
63#
64#   class LiensSupplementaires(object):
65#     def index(self):
66#       #: لاحظ الإرتباط النسبي للعودة إلى الصفحة الرئيسية
67#       return '''
68#         <p>Encore quelques autres liens utiles :</p>
69#         <ul>
70#           <li><a href="http://pythomium.net">Le site de l'auteur</a></li>
71#           <li><a href="http://ubuntu-fr.org">Ubuntu : le must</a></li>
72#         </ul>
73#         <p>[<a href="../">Retour à la page racine des liens</a>]</p>
74#       '''
75#     index.exposed = True
76#
77#   racine = HomePage()
78#   cherrypy.quickstart(racine, config ="tutoriel.conf")
```

السطور من 4 إلى 10 : أسلوب منشئ الكائنات الجذر هو المكان الميثالي لتمثيل كائنات أخرى "عبيد" . فإننا نصل إلى أساليب معالجة الإستعلامات و هذا تماما كما يمكننا الوصول إلى الدلائل الفرعية من الدليل الجذر (أنظر أدناه) .

السطور من 12 إلى 22 : إن صفحة الرئيسية توفر روابط للصفحات الأخرى للموقع . ستلاحظ أن بينة الجمل المستخدمة في علامات-الوصلات, تستخدم هنا لتعريق مسار كامل :

* أساليب الكائن الجذر يتم عمل مرجع لهم عن طريق الرمز / متبوع بإسم واحد . الرمز / يشير أن "مسار" جزء من الجذر الموقع . على سبيل المثال : **/enterNous** .

* إن أساليب الجذر الكائنات العبيد يتم عمل مرجع لهم بإستخدام رمز / بسيط ثم إسم هذه الكائنات الأخرى . على سبيل المثال : **/maxime/** .

* أساليب الكائنات العبيد يتم عمل مرجع لهم بإستخدام إسم المدرج في مسار الكامل : على سبيل المثال **/ liens/utiles** .

الأسطر 36 و 62 و 75 : لإيجاد جذر المستوى السابق, إستخدمنا هذه المرة مسار نسبي, مع نفس تكوين الجملة التي إستخدمت للعودة إلى الدليل السابق في نظام الملفات (نقطتين) .

الأسطر 41 و 42 : عليك أن تعرف أننا ثبتنا تسلسل هرمي في شكل شجرة ملفات, و قمنا بتمثيل كائنات "العبيد" عن بعضها البعض . بعد هذا المنطق, ينبغي أن المسار الكامل يؤدي إلى الأسلوب **index()** لهذا الصنف أن يكون **/liens/extra/index** .

دعم الجلسات

عند تطوير موقع إلكتروني تفاعلي, نحن نأمل أن الشخص يقوم بزيارات متكررة للموقع, و يمكن تحديد و توفير عدد من المعلومات في زياراته في كامل الصفحات المختلفة (على سبيل المثال, ملئ سلة من خلال التشاور من موثع تجاري), و كل هذه المعلومات سوف يتم تخزينها في مكان ما حتى ينهي زيارته . بالطبع, يجب علينا أن نربطه بكل عميل متصل بشكل مستقل, لأننا لا يمكن أن ننسى أن الغرض الموقع على شبكة الإنترنات هو إستخدامه جنبا إلى جنب مع مجموعة متنوعة من الأشخاص .

سيكون من الممكن نقل هذه المعلومات من صفحة إلى صفحة أخرى, وذلك بإستخدام حقول النموذج المخفي (العلامة **<"INPUT TYPE="hidden>**), لكن هذا سيكون معقد و مرهق . و لذلك سيكون من الأفضل تجهيز الخادم مع آلية خاصة, لتعيين لكل عميل جلسة خاصة, و التي سوف تقوم بتخزين جميع المعلومات الخاصة بهذا العميل . Cherrypy يحقق هذا الهدف من خلال الملفات تعريف الإرتباط (كوكيز) .

عندما يدخل زائر جديد لهذا الموقع, سيقوم الخادم بإنشاء ملف تعريف الإرتباط (و هذا يعني حزمة صغيرة من المعلومات التي تحتوي على معرف جلسة فريدة من نوعها لسلسلة عشوائية من وحدات البايت) و تقوم بإرجاعها لمتصفح الويب, الذي يقوم بحفظها . في إتصال مع ملف تعريف الذي تم عمله, سيقوم الخادم بالإحتفاظ لبعض الوقت كائن-جلسة الذي سيقوم بتخزين جميع المعلومات الخاصة بالزائر .

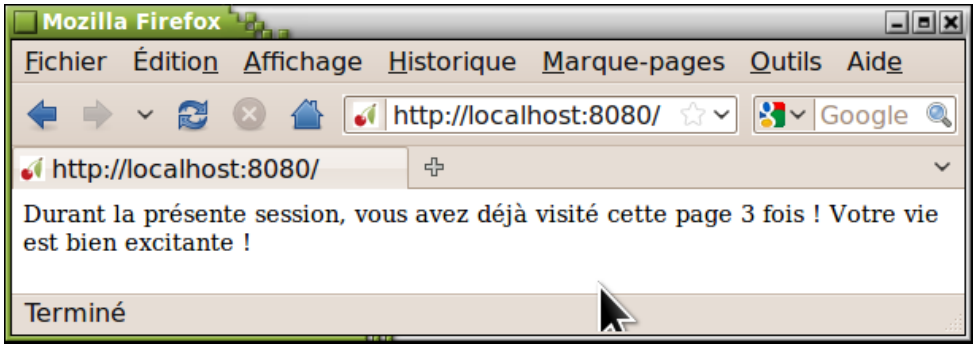
في إتصال مع الكوكيز(ملف تعريف الإرتباط) الذي تم صنعه, سيقوم الخادم بالحتفاظ لبعض الوقت كائن-جلسة الذي سيتم حفظ جميع معلومات الخاصة بالزائر . عند فتح صفحات الموقع الأخرى, يقوم المتصفح بإعادة إرسال كل مرة محتوى كوكيز الخادم, و هذا يسمح بتحديد و إسترداد كائن-الجلسة . لا يزال كان-الجلسة متاح في جميع الصفحات التي يزورها الزاشر : هو كائن بايثون عادي, يتم من خلاله تخزين أي عدد من المعلومات في شكل سمات .

من حيث البرمجة, هكذا تسير الأمور :

```
1#    import cherrypy
2#
3#    class CompteurAcces(object):
4#      def index(self):
5#          #مثال بسيط : تزايد العداد
6#          #نبدأ بإسترجاع إجمالي العد الحالي :
7#          count = cherrypy.session.get('count', 0)
8#          #زيادة ما يلي ... :
9#          count += 1
10#         #نخزين القيمة الجديدة في قاموس الجلسة ... :
11#         cherrypy.session['count'] = count
12#         #و نقوم بعرض العد الحالي ... :
13#         return '''
14#           Durant la présente session, vous avez déjà visité
15#           cette page {0} fois ! Votre vie est bien excitante !
16#         '''.format(count)
17#      index.exposed = True
18#
19#    cherrypy.quickstart(CompteurAcces(), config='tutoriel.conf')
```

يمكنك ببساطو إعادة طلب الصفحة التي ينتجها هذا السكريبت مرارا و تكرارا ستلاحظ أنه في كل مرة يزداد فيها عداد الزيارة مرة, كما هو مبين في الشكل أدناه .

يجب أن يكون السكريبت يفسر نفسه بنفسه . و يجدر الإشارة إلى أن وحدة cherrypy تم تكيفها لكائن **session** الذي يتصرف (على ما يبدو) كقاموس كلاسيكي . يمكننا أن نضيف مقاتيح لإدارتها, و نربطها



بمفاتيح أي قيم .

يمكنك ببساطو إعادة طلب الصفحة التي ينتجها هذا السكريبت مرارا و تكرارا ستلاحظ أنه في كل مرة يزداد فيها عداد الزيارة مرة, كما هو مبين في الشكل أدناه .

يجب أن يكون السكريبت يفسر نفسه بنفسه . و يجدر الإشارة إلى أن وحدة cherrypy تم تكيفها لكائن **session** الذي يتصرف (على ما يبدو) كقاموس كلاسيكي . يمكننا أن نضيف مقاتيح لإدارتها, و نربطها بنفاتيح أي قيم .

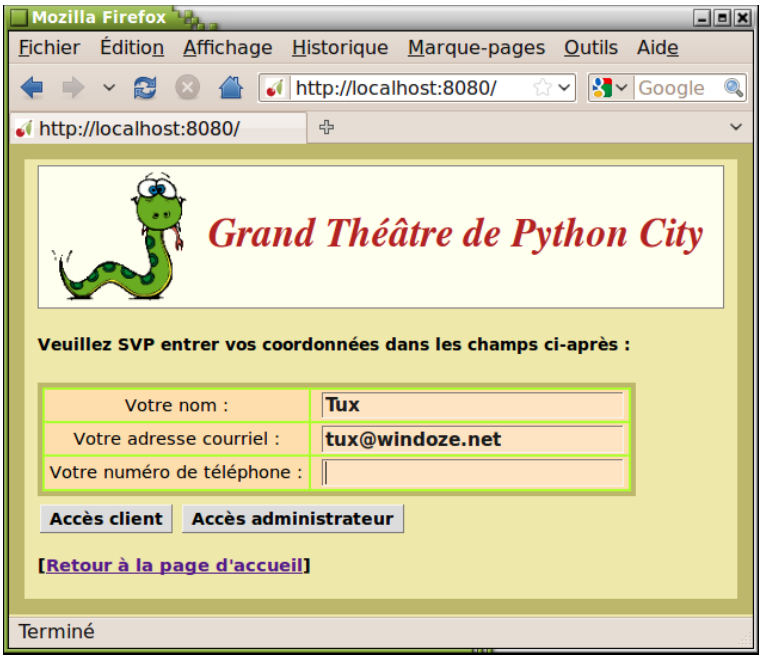
في السطر 7 من مثالنا, نحن إستخدمنا الأسلوب **get()** للقوميس, لإيجاد قيمة المرتبطة بمفتاح **count** (أو صفر, إذا كان المفتاح غير موجود) . في السطر 11 سنقوم بتسجيل هذه القيمة, المتزايدة, في نفس القاموس . و هكذاي يمكننا أن نرى مرة أخرى أن Cherrypy يوفر لنا بيئة برمجية مؤلوفة إلى البايثون العادية .

لاحظ دائما أن الكائن **session**, الذي يتصرف لنا كقاموس بسيط, هو في الواقع آلية داخلية معقدة, لأننا نخدم تلقائيا المعلومات التي تتوافق مع العميل معين من موقعنا, الذي حدد عن طريق الكوكيز الجلسة . لترى

هذا, جرب الوصول إلى خادمك من خلال متصفحين مختلفين[88] (على سبيل المثال,فايرفوكس و أوبرا) : سوف تجد أن عدد الزيارات هو في الواقع مختلف لكل واحدة منهم .

عمل موقع تفاعلي ملموس على شبكة الإنترنات

مع كل ما تعلمناه حتى الأن, يجب أن نكون قادرين على النظر إلى مشروع أكثر أهميو . هذا ما سنقدمه أدناه, تفاصيل تطوير موقع على شبكة الإنترنات التي يمكن إستخدامه نوعا ما لحجز مفعد في المسرح[89] :



---

[88] إنتباه : لا يمكن تمييز الجلسات سوى من بداية الاليات المختلفة (إسستخدام أي متصفح), أو المتصفحات المختلفة التي تعمل على نفس الجهاز . فإذا قمت بتشغيل مثيلين من المتصفح نفسه على نفس الجهاز, سوف تستخدم ملفات التعريف الإرتباط نفسها, و هذا يعني أن الخادم لا يمكنه تمييز بين الطلبات الواردة من أي واحدة . و بعبارة أخرى, فإن هذان المثيلين من المتصفح نفسه يشاركون نفس الجلسة .

[89] رسم الثعبان هو شعار لرنامج حر WebCamSpy (و هذا البرنامج كتب بالبايثون) .
Noir : *http://webcamspy.sourceforge.net*

يمكن ل "مدراء" الموقع إضافة مشاهدين إلى القائمة و عرض الحجوزرات . و يمكن "للعملاء" التسجيل و حجز أماكن لمشاهد أو أكثر, و يمكنه مشاهدة الأماكن الذي إشتراها .

في الحقيقة يوجد دائما, تمرين, وظائف هذا التطبيق الصغير ربما قد تكون غير مكتملة . فهي تفتقر إلى تحكم دخول المدراء (على سبيل المثال,عن طريق نظام تسجيل دخول), و القدرة على حذف أو تعديل المشاهدين الحاليين و الحجوزات, و معالجة التواريخ و عناوين البريد الالكتروني و أرقام الهواتف (هنا سلاسل نصية بسيطة), إلخ .

التطبيق يتضمن قاعدة بيانات صغيرة علائقية تضم ثلاثة جداول, و تربط بواسطة إتصالين من نوع "واحد إلى العديد" . تم صنع صفحات الويب بتخطيط مشترك, و الديكورها تم إستخدام CSS . كودات تطبيق البايثون و كودات HTML "الرئيسة" منفصلة في ملفين منفصلين .

لقد قمنا بدمج هذا المثال بأقصى حد من المفاهيم المفيدة, لكم تم ترك عمدا كود التحكم الذي من شأنه أن تكون له حاجة حقيقية في التطبيق للتحقق من مدخلات المستخدم, و الكشف عن أخطاء الإتصال مع قاعدة البيانات ... إلخ , حتى لا تشوه المظاهر .

بغض النظر عن قاعدة البيانات (و التي سوف تعتمد على SQLite), ينقسم الجزء الأساسي من التطبيق إلى 3 ملفات منفصلة, و ذلك لفصل معالجة البيانات, و تجهيزه المستخدم و تزيين الوقع :* معالج البيانات يتم

توفيره من قبل سكريبت البايثون الذي سنصفه بعد قليل . و يتم تضمينه في ملف واحد **spectacles.py** لكن نحن نقدم لكم عدة قطع لتسهيل الشرح و عرض النص .

هذا السكريبت يستخدم آلية الجلسات لتذكر تفاصيل المستخدم خلال زيارته .

* إن تقديم البيانات تضمنه مجموعة من صفحات الويب, بما في ذلك كود HTML الذي يتم ترجمة الجزء الأكبر منه في ملف نصي منفصل **spectacles.htm** . حسب البرنامج, نقوم بإستخراج من هذا الملف سلاسل نصية الذي سيتم تنسيقها عن كريق إدخال فيم متغيرات, و ذلك بإستخدام تقنية الموضحة في الفصل 10 . و سوف تكون صفحات المختوى ثابتة لا تشوش السكريبت نفسه . سوف نصنع محتويات هذا الملف في الصفحة Error: Reference source not found.* تزيين الصفحات سيتم عن طريق إستخدام ورقة الأنماط CSS, و التي سوف تكون في ملف منفصل **spectacles.css** . نحن لن نشرح في هذه الصفحات كود CSS المستخدم, لأن هذا خارج نطاق دورة برمجة البايثون . لقد قمنا بإدراج أسماء وثائق يمكن تحميلها من موقع هذا الكتاب .

السكريبت

يبدأ سكريبت **spectacles.py** بتعريف صنف **Glob()** الذي سيستخدم كحاوي للماغيرات التي نريد معالجتها كمتغيرات عامة . و يتضمن وصف الجداول في قاعدة البيانات في قاموس, و ذلك بإستخدام تقنية مشابهة لما تم شرحه في الفصل السابق :

```
1#      import os, cherrypy, sqlite3
2#
3#      class Glob(object):
4#          # Données à caractère global pour l'application
5#          patronsHTML ="spectacles.htm"      # الملف الذي يحتوي على "علامات" HTML
6#          html ={}          # Les patrons seront chargés dans ce dictionnaire
7#          # سيتم تحميل العلامات في هذا قاموس هيكل قاعدة البيانات . قاموس الجداول و الحقول :
8#          dbName = "spectacles.sq3"          # إسم قاعدة البيانات
9#          tables ={"spectacles":(("ref_spt","k"), ("titre","s"), ("date","t"),
10#                      ("prix_pl","r"), ("vendues","i")),
11#              "reservations":(("ref_res","k"), ("ref_spt","i"), ("ref_cli","i"),
12#                      ("place","i")),
13#              "clients":(("ref_cli","k"), ("nom","s"), ("e_mail","s"),
14#                      ("tel", "i")) }
```

تليها تعاريف ثلاثة دالات . الأول (من السطر 16 إلى 33) لن تستخدم سوى مرة واحدة عند بدء التشغيل . دورها هو قراءة ملف النصي **spectacles.htm** لإستخراج كودات HTML التي سوف تستخدم لتنسيق صفحات الويب . فإذا فحصت هيكل هذا الملف (لقد قمنا بتكرار محتوى الملف في الصفحات من Error: Reference source not found-Error: Reference source not found), سوف ترى أنه يحتوي على سلسلة من الأقسام, واضحة من كل واحد من التكرارين : علامة الفتح (شكلت نفسها من نجمتين و خطافين) و سطر أخير يتكون من ما لا يقل عن خمسة رموز # . و يمكن إستخراج كل قسم عن حدة و

تخزينها في قاموس عما **glob.html** . سيتم العثور على مقاتيح هذا القاموس و القيم الأقسام المقابلة, كل واحد منها يحتوي على صفحة من كود HTML, مع علامات التحويل {0}, {1}, {2}, إلخ . و الذي يمكن إستبدالها بقيم متغيرات . و يجب أن يؤخذ في الإعتبار ترميز هذا الملف النصي, بالطبع (السطر 18) .

```
15#
16#   def chargerPatronsHTML():
17#       #تحميل جميع "علامات" صفحات ال HTML في قاموس
18#       #يتم تحديد الترميز, في حال إذا كان يختلف عن الإفتراضي) :
19#       fi =open(Glob.patronsHTML, "r", encoding ="Utf8")
20#       try:          #لضمان أن يتم إغلاق الملف دائما
21#           for ligne in fi:
22#               if ligne[:2] =="[*":          #العثور على التسمية ==>
23#                   label =ligne[2:]          #مسح [*
24#                   label =label[:-1].strip()    #suppression LF et esp évent.
25#                   label =label[:-2]         #مسح *]
26#                   txt =""
27#               else:
28#                   if ligne[:5] =="#####":
29#                       Glob.html[label] =txt
30#                   else:
31#                       txt += ligne
32#       finally:
33#           fi.close()        #سيتم إغلاق الملف في جميع الحالات
34#
```

الدالة الثاني, على الرغم من أنها بسيطة, فهي تؤدي عمل رائع : بل في الواقع من شأنه أن يسمح لنا بتقديم جميع الصفحات المتشابهة , بإدراجها في نمط شائع . هذا النمط, مثل جميع الأخرين, بأتي من ملف **spectacles.htm** لكن الدالة السابقة قد قدمته لنا في قاموس **Glob.html** تحت إسم " **miseEnPage**" :

```
35#   def mep(page):
36#       #تقوم بإرجاع <الصفحة> الممررت, التي وضع بها برامتر في رأس و  HTML : توليد دالة "التخطيط" لل
      تذييل الصفحة الملائمة .
37#       return Glob.html["miseEnPage"].format(page)
38#
```

الدالة الثالثة تصنع قطعة قطعة سلسلة نصية التي تحتوي على كود HTML الازم لوصف الجدول . هذا الجدول سيتم ملئه تلقائيا مع قائمة البرامج المدرجة حاليا في قاعدة البيانات . نرى هنا بشكل واضح جدا مساهمة البرمجة الأساسية لتحقيق موقع ديناميكي, و هذا يعني موقع الذي يتم تحديث صفحاته بإستمرار إستنادا إلى البيانات التي يقدمها الزوار بأنفسهم. أو وفقا لمختلف الأحداث :

```
39#   def listeSpectacles():
40#       #إنشاء قائمة من العروض المتاحة, في جدول HTML
41#       req ="SELECT ref_spt, titre, date, prix_pl, vendues FROM spectacles"
42#       res =BD.executerReq(req)          #==> res sera une liste de tuples
43#       tabl ='<table border="1" cellpadding="5">\n'
44#       tabs =""
```

```
45#     for n in range(5):
46#        # ملاحظة : لتظهر كسلسلة منسقة, يجب أن تقوم بمضاعفة الأقواس :
47#        tabs += "<td>{{{0}}}</td>".format(n)
48#     ligneTableau ="<tr>" +tabs +"</tr>\n"
49#     #: السطر الأول من الجدول تحتوي على رؤوس الأعمدة
50#     tabl += ligneTableau.\
51#        format("Réf.", "Titre", "Date", "Prix des places", "Vendues")
52#     #: الأسطر التالية : يتم إستخراج محتواها من BD
53#     for ref, tit, dat, pri, ven in res:
54#        tabl += ligneTableau.format(ref, tit, dat, pri, ven)
55#     return tabl +"</table>"
56#
```

السطور 42 و 43 ستستعلم عن قاعدة البيانات لإستخراج المعلومات حول العروض المتاح من خلال صنف-الواجهة الذي تم وصفه في وقت لاحق . عند بداية السكريبت, سوف يتم إنشاء مثيل يسمى **BD** كائن من هذا الصنف, الذي يقوم الأسلوب **executerReq()** بإرجاع نتيجة إستعلام SQL في السطر 42 في شكل سلسلة و أنفاق .

الأسطر التالية تظهر لك كيفية بناء جدول HTML من خلال البرمجة, و الإستفادة من تقنيات السلاسل التنسيق التي تم وصفها في الفصل 10 .

الأسطر من 45 إلى 50 تقوم ببناء أولا سلسلة التنسيق :

>"

tr><td>{0}</td><td>{1}</td><td>{2}</td><td>{3}</td><td>{4}</td>

"<</tr

لتكون بمثابة قالب رئيس لتوليد كود HTML الذي يصف أسطر الجدول . لاحظ الحاجة إلى مصاعفة الرموز { و } بحيث يتم إدراجها على هذا النحو في السلسلة .

الأسطر التالية تقوم ببناء كود HTML بنفسها, و تستكمل السلسلة التي تم بدأها في السطر 44, مع وصف كامل لأسطر الجدول . علامات التنسيق للرئيس ستم إستبدالها واحدة تلو الأخرى بالمعلومات التي تم إستخراجها من قاعدة البيانات (تدوير قائمة الأنفاق, الأسطر 55 و 56), و يتم إنهاء السلسلة بإرجاع للبرنامج الذي تم إستدعائه في السطر 57 :

إن **GestionBD()** تضمن التواصل مع قاعدة البيانات . و التي هي نسخة مبسطة من صنف بنفس الإسم تم وصفه في نهاية الفصل السابق . تم تحسينه بالتأكيد[90]. و قاعدة البيانات تحتوي بنفسها تماما مثل الملف **spectacles.sq3** . فإذا قمت بحذف هذا الملف, سيكون من الممكن إعادة صنعه تلقائيا من خلال الأسلوب **creaTables()** :

```
57#   class GestionBD(object):
58#       #. التنفيذ و التواصل مع قاعدة البيانات SQLite .
59#
60#       def __init__(self, dbName):            # أنظر إلى ملاحظات الكتاب حول الخيوط
61#           self.dbName =dbName
62#
63#       def executerReq(self, req, param =()):
64#           #تشغيل الإستعلاك <req>, مع إرسال النتيجة المحتملة
65#           connex =sqlite3.connect(self.dbName)   #إنشاء إتصال
66#           cursor =connex.cursor()            #إنشاء المؤشر
67#           cursor.execute(req, param)          #تنفيذ إستعلام SQL
68#           res =None
69#           if "SELECT" in req.upper():
70#               res =cursor.fetchall()          #<res> = قائمة أنفاق
71#           connex.commit()                    #تسجيل منتظم
72#           cursor.close()
73#           connex.close()
74#           return res                 #أو قائمة من الأنفاق None سوف يرجع
75#
76#       def creaTables(self, dicTables):
77#           #إنشاء جداول من قاعدة البيانات إذا لم تكن موجوة بالفعل
78#           for table in dicTables:            #تدوير مفاتيح القاموس
79#               req = "CREATE TABLE {0} (".format(table)
80#               pk =""
81#               for descr in dicTables[table]:
82#                   nomChamp = descr[0]        #تسمية الحقل الذي سيتم إنشاؤه
83#                   tch = descr[1]             #نوع الحقل الذي سيتم إنشاؤه
84#                   if tch =="i":
85#                       typeChamp ="INTEGER"
86#                   elif tch =="k":
```

[90] سترى على وجه الخصوص أننا أعدنا صنع كائن إتصال جديد عند كل إستعلام, و هذا ليس بالأمر السعيد لكن فعلنا هذا لأن SQLite لا تسمح بإستخدام ضمن خيط خاص, و كائن إتصال الذي تم إنشاؤه في خيط اخر . (و الخيوط هي أسطر برمجية يتم تنفيذه بالتوازي في نفس البرنامج . و تطبيق ويب مثل Cherrypy هو إحداها, و ذلك من أجل المعالجة في نفس الوقت تقريبا إستعلامات المستخدمين المختلفة . و سيتم شرح الخيوط في الفصل 19) . إذا كنت تريد صنع كائن إتصال واحد لمعالجة مجموعة من الإستعامات, و ينبغي أن : إما أن لا يستخدم البرنامج في كل شيئ سوى خيط واحد, و إما إضافة الوظيفة المطلوبة لتعريف الخيوط الحالية, و إما إستخدام SGBDR أخر غير SQLite (على سبيل المثال, PostgreSQL) . و كل هذا سوف يكون ممكنااا, و لكن يتطلب الكثير من الشحر و هذا سيكون خارج نطاق هذا الكتاب .

```
87#                 #حقل "مفتاح أساسي" (عدد صحيح متزايد تلقائيا)
88#                 typeChamp ="INTEGER PRIMARY KEY AUTOINCREMENT"
89#                 pk = nomChamp
90#               elif tch =="r":
91#                 typeChamp ="REAL"
92#               else:              # للتبسيط, سوف نعتبر جميع الأنواع الاخرى كنصوص
93#                 typeChamp ="TEXT"
94#               req += "{0} {1}, ".format(nomChamp, typeChamp)
95#             req = req[:-2] + ")"
96#             try:
97#               self.executerReq(req)
98#             except:
99#               pass              # الجدول على الأرجح موجود بالفعل
100#
```

يتم عتماد الوظيفة الرئيسية للموقع بواسطة الصمف التالي : **WebSpectacles()** . عند بداية السكريبت, يقوم Cherrypy بتمثيل عدة كائنات من هذا الصنف لمواضيع (threads) مختلفة من أجل صنع يالتوازي إستعلامت من مختلف المستخدمين , هذه الآلية تعدد المهام سوف يتم شرحها في الفصل القادم, لكن يجب أن لا تقلق بشأن ذلك الأن لأن Cherrypy يوفر طريقة أكثر شفافية . لفهم ما يلي, أنت تعرف أن Cherrypy يقوم بتحويل كل واحدة من طلبات URL المطلوبة من قبل متصفح الويب الزائر بإستدعاء أساليب هذا الصنف, منا قمنا بوصقه موجوز له في الصفحات الأولى من هذا الفصل .

```
101#  class WebSpectacles(object):
102#    # صنف مولد كائنات معالجة إستعلامات HTTP .
103#
104#    def index(self):
105#        الصفحة الرئيسية للموقع . و سوف تستخدم متغيرات الجلسة لتحديد العمليات التي تجري بالفعل (أو لا) من قبل الزار :
106#        nom =cherrypy.session.get("nom", "")
107#        #: تتكيف مع وضع الزائر HTML إرجاع صفحة
108#        if nom:
109#          acces =cherrypy.session["acces"]
110#          if acces =="Accès administrateur":
111#            #: ثابتة " HTML إرجاع صفحة
112#            return mep(Glob.html["accesAdmin"])
113#          else:
114#            #: منسقة مع إسم الزائر HTML إرجاع صفحة
115#            return mep(Glob.html["accesClients"].format(nom))
116#        else:
117#          return mep(Glob.html["pageAccueil"])
118#    index.exposed =True
119#
```

يتم إنشاء الصفحة الأولى من قبل الأسلوب **index()** . و هذا هو شكل من أشكال HTML, ثم سنقوم بتحميل الكود في قاموس **Glob.html** (تحت إسم "**pageAccueil**") أثناء مرحلة تهيئة البرنامج . سيقوم الأسلوب **index()** بإرجاع هذا الكود (في السطر 120), لكن فقط إذا لم يحدد زائر الموقع . و إذا حدد, سوف تحتوي القيم في كائن-الجلسة على **cherrypy.session["nom"]** و **"]**

cherrypy.session["acces, سيقوم البرنامج بإرجاع الإحالات المرجعية لصفحات الأخرى على الويب, الأنماط التي سوف يتم تخزينها في **Glob.html** (الأسطر من 111 إلى 118) .

جميع الصفحات سيقوم بإرجاع "ما يرتدونه" مسبقا بإستخدام الدالة **mep()**, التي سوف "تزين" بطريق مماثلة للمؤشرات التالية التي هي ورقة أنماط CSS . في السطر 118 سيقوم بجمع بين إثنين من التنسيقات المتتالية, الأول لدمج كود HTML المنتج محليا (إسم تم إدخاله من قبل المستخدم) الذي سيتم إستخراج النمك من قاموس **glob.html** و الثانية ل"الإلتفاف" على كل رئيس أخر, عبر دالة **mep()** .

الأسلوب التالي هو معقد قليلا . لفهم وظيفته, فمن الأفضل أن نتفحص أولا محتويات الصفحة الرئيسية التي تم إرجاعها للمستخدم عن طريق الأسلوب **index()** (في السطر 120) . هذه الصفحة تحتوي على شكل HTML حتى يتسنى لنا إنتاجه فيما بعد . و يحدها نموذج مثل هذا عن طريق علامات **<form>** و **/> form>** :

```
<form action="/identification" method=GET>
<h4>Veuillez SVP entrer vos coordonnées dans les champs ci-après :</h4>
<table>
<tr><td>Votre nom :</td><td><input name="nom"></td></tr>
<tr><td>Votre adresse courriel :</td><td><input name="mail"></td></tr>
<tr><td>Votre numéro de téléphone :</td><td><input name="tel"></td></tr>
</table>
<input type=submit class="button" name="acces" value="Accès client">
<input type=submit class="button" name="acces" value="Accès administrateur">
</form>
```

تستخدم سمة **action** في علامات **<form>** تشير إلى الذي سيتم إستدعائه عندما ينقر الزائر على أحد أزرار من نوع إرسال (submit) . هذا العنوان (URL) سيتم تحويله عبر Cherrypy بإستدعاء أسلوب من نفس الإسم, على جذر الكوقع منذ أن الإسم السابق بسيط "/" . و لذلك فإن الأسلوب **identification()** لصنفنا الرئيسي هو الذي سيتم إستدعائه . و إن علامات من نوع **<input name= "..."/>** تقوم بتعريف حقل الإدخال, ولكل واحدة منها إسمها الخاص الذي تم الإشارة إليه بسمة **name** . و هذه التسميات التي تسمح ل Cherrypy بتمرير قيم تم ترميزها داخل الحقول, إلى برامترات من نفس أسماء الأسلوب **identification()** . أنظر الأن إلى هذا :

```
120#     def identification(self, acces="", nom="", mail="", tel=""):
121#       #: يتم تحزين إحداثيات الزائر في متغير الجلسة :
122#       cherrypy.session["nom"] =nom
123#       cherrypy.session["mail"] =mail
124#       cherrypy.session["tel"] =tel
125#       cherrypy.session["acces"] =acces
126#       if acces =="Accès administrateur":
127#         return mep(Glob.html["accesAdmin"])
128#       else:
```

```
129#        #: يستخدم لحجز مقاعد للمشاهدين التي إختارها الزائر "caddy" متغير الجلسة
130#        cherrypy.session["caddy"] =[]        #(قائمة فارغة, في البداية)
131#         return mep(Glob.html["accesClients"].format(nom))
132#    identification.exposed =True
133#
```

البرامترات التي تم تلقيها في المتغيرا المحلية **mail**, **nom**, **acces** و **tel** . نأمل أن يتم تخزين هذه القيم الخاصة لكل مستخدم, و هو مايدفعنا إلى إسناد إلى رعاية كائن-الجلسة **cherrypy-session**, الذي يقدم لنا تحت الغطاء قاموس بسيط (الأسطر من 125 إلى 128 و 134) .

السطر 129 : البرامتر **acces** سيقوم بتلقي قيمة المقابلة لزر الإرسال (**submit**) الذي سيستخدم من قبل الزائر, أي المسؤولين عن "دخول المديرين" أو "دخول العملاء" . و هذا يسمح لنا بتوجيه الزوار إل الصفحات التي تناسبهم .

السطر 134 : سو نقوم بحفظ الحجوزات التب طلبها الزائر في فائمة من الأنفاق, التي سوف تملئ بطريقة خاص . قياسا على ما يتم على المواقع التجارة الإلكترونية على شبكة الإنترنات, سوف ندو هذه القائمة ب "سلة - panier" أو "عربة تسوق - caddy" . و أما عن حفظ هذه الحجوزات في قاعدة البيانات سيتم في وقت لاحق, في خطوة منفصلة, و فقط عندما يريد المستخدم حفظ هذه القائمة المحددة طوال فترة زيارة الموقع, و نحن قد وضعنا في متغير يمسى "caddy" (سوف نسمي الأن متغيرات الجلسة القيم المخزنة في كائن-الجلسة **cherrypy.session**) .

تقوم صفحة الوب بإرجاع للمستخدم "العميل" (السطر 135) صفحة بسيطة ثابتة, التي توفر وصلات لصفحات أخرى . بقية السكريبت يحتوي على الأساليب المقابلة :

```
134#    def reserver(self):
135#      #: تقديم نموذج الحجز إلى الزائر "العميل"
136#      nom =cherrypy.session["nom"]          #البحث عن إسمه
137#      #: البحث في BD قائمة المشاهدين المقترحة
138#       tabl =listeSpectacles()
139#       return mep(Glob.html["reserver"].format(tabl, nom))
140#    reserver.exposed =True
141#
142#    def reservations(self, spect="", places=""):
143#      #: تخزين الحجوزات المطلوبة, في متغير الجلسة
144#      spect, places = int(spect), int(places)   #تحويل إلى أرقام
145#      caddy =cherrypy.session["caddy"]          #إستعادة الوضع الحالي
146#      caddy.append((spect, places))             #إضافة نفق إلى القائمة
147#      cherrypy.session["caddy"] =caddy          #تخزين القائمة
148#       nSp, nPl = len(caddy), 0
149#      for c in caddy:                           #مجموع الحجوزات
150#          nPl += c[1]
151#       return mep(Glob.html["reservations"].format(nPl, nSp))
152#    reservations.exposed =True
```

153#

السطور من 138 إلى 144 : هذا الأسلوب يولد إستمار حجز في مسارح الموجودة . فهي تستخدم الدالة **listeSpectacles()**, المذكورة أعلاه, لتوليد قائمة كجدول HTML . ثم دمج ذلك في تخطيط نموذج نفسه, الذي يمكن العثور على كود "ثابت" مرة أخرى في قاموس "الرئيس" **Glob.html** عند بدء السكريبت . دراسة هذا "الرئيس: (أنظر للصفحة Error: Reference source not found) يخبرنا أنه سيتم إستدعاء هذه المرة الأسلوب **reservations()** عندما يقوم المستخدم بالضغط على زر **>Enregistrer<** (حفظ) . هذا الأسلوب يتلقى برامترات **spect** و **places** و القيم المدخلة من قبل الزائر في النموذج, ثم يقوم بتجميعها في نفق, و يضيفها إلى قائمة المودود في متغير الجلسة "caddy" . ثم يقوم بإرجاع للمستخدم صفحة صغيرة لتطوير طلباته .

السطر 148 : جميع البرامترات التي تم تمريرها عن طريق نموذج HTML هي سلاسل نصية . إذا كانت البرامترات رقمية, لابد من تحويلها إلى نوع مناسب قبل إستخدامها على هذا النحو .

الأساليب التالية تسمح للمستخدم "عميل" بإغلاق زيارته للموقع و تسجيل الحجوزات, أو مراجعة الحجوزات التي قدمت سابقا . الأساليب للدالات المحجوز لل "مديرين" تأتي الأن . جميع هذه الأساليب بنية على نفس المبدأ و لا تتطلب التعليقات .

ينبغي لإستعلامات SQL الواردة في الأسطر التالية أن تكون تفسيرية . سوف تتعرف على نوعين من الصلات .

> الوصف الفصل لهذه الأسطر سيكون خارج نطاق الكتاب . فإذا كانت تبدو معقدة قليلا, لا تشعر بالإحباط : تعلم هذه اللغة يمكن أن يكون تدريجيا . إعلم أن هذا الأمر لا بد منه إذا كنت ترغب بأن تصبح مطور ماهر حقيقي .

```
154#        def finaliser(self):
155#        #حفظ "caddy" للعميل في قاعدة البيانات .
156#         nom =cherrypy.session["nom"]
157#         mail =cherrypy.session["mail"]
158#         tel =cherrypy.session["tel"]
159#         caddy =cherrypy.session["caddy"]
160#        #: حفظ معلومات الخاصة بالعميل في جدول مخصص
161#         req ="INSERT INTO clients(nom, e_mail, tel) VALUES(?,?,?)"
162#         res =BD.executerReq(req, (nom, mail, tel))
163#        #: إسترجاع مرجع الذي تم تعيينه تلقائيا
164#         req ="SELECT ref_cli FROM clients WHERE nom=?"
165#         res =BD.executerReq(req, (nom,))
166#        client =res[0][0]        #إستخراج العنصر الأول من النفق الأول
167#        #: تدوير caddy - تسجيل أماكن لكل مشاهد
168#         for (spect, places) in caddy:
169#           #: البحث عن أخر رقم مكان تم حجزه لهذا المشاهد
170#           req ="SELECT MAX(place) FROM reservations WHERE ref_spt =?"
171#           res =BD.executerReq(req, (int(spect),))
172#           numP =res[0][0]
173#           if numP is None:
174#             numP =0
175#           #: توليد أرقام الأماكن التالية, حفظ
176#           req ="INSERT INTO reservations(ref_spt,ref_cli,place) VALUES(?,?,?)"
177#           for i in range(places):
178#             numP +=1
179#             res =BD.executerReq(req, (spect, client, numP))
180#           #: حفظ عدد الأماكن المباعة لهذا المشاهد
181#           req ="UPDATE spectacles SET vendues=? WHERE ref_spt=?"
182#           res =BD.executerReq(req, (numP, spect))
183#        cherrypy.session["caddy"] =[]     #تفريغ لب caddy
184#        cherrypy.session["nom"] =""       #نسيان "الزائر"
185#         return mep("<h3>Session terminée. Bye !</h3>")
186#       finaliser.exposed =True
187#
188#       def revoir(self):
189#        #. العثور على حجوزات التي تم تنفيذها من قبل عميل معين
190#        #: (نجد إشارته بإستخدام عنوان بريد الإلكتروني)
191#         mail =cherrypy.session["mail"]
192#         req ="SELECT ref_cli, nom, tel FROM clients WHERE e_mail =?"
193#         res =BD.executerReq(req, (mail,))
194#         client, nom, tel =res[0]
195#        #Spectacles pour lesquels il a acheté des places :
196#         req ="SELECT titre, date, place, prix_pl "\
197#             "FROM reservations JOIN spectacles USING (ref_spt) "\
198#             "WHERE ref_cli =? ORDER BY titre, place"
199#         res =BD.executerReq(req, (client,))
200#        #: إنشاء جدول HTML لقائمة المعلومات الموجودة
201#         tabl ='<table border="1" cellpadding="5">\n'
202#         tabs =""
```

```
203#        for n in range(4):
204#            tabs +="<td>{{{0}}}</td>".format(n)
205#        ligneTableau ="<tr>" +tabs +"</tr>\n"
206#        #: أول صف من الجدول يحتوي على رؤوس الأعمدة :
207#        tabl += ligneTableau.format("Titre", "Date", "N° place", "Prix")
208#        #: الصفوف التالية
209#        tot =0                    #: حساب السعر الأجمالي
210#        for titre, date, place, prix in res:
211#            tabl += ligneTableau.format(titre, date, place, prix)
212#            tot += prix
213#        #: إضافة سطر في أسفل الجدول مع المجموع بشكل بارز
214#        tabl += ligneTableau.format("", "", "Total", str(tot))
215#        tabl += "</table>"
216#        return mep(Glob.html["revoir"].format(nom, mail, tel, tabl))
217#    revoir.exposed =True
218#
219#    def entrerSpectacles(self):
220#        #: إجاد قائمة المشاهدين الحالية
221#        tabl =listeSpectacles()
222#        #: إرجاع نموذج لإضافة مشاهد جديد
223#        return mep(Glob.html["entrerSpectacles"].format(tabl))
224#    entrerSpectacles.exposed =True
225#
226#    def memoSpectacles(self, titre ="", date ="", prixPl =""):
227#        #تخزين مشاهد جديد
228#        if not titre or not date or not prixPl:
229#            return '<h4>Complétez les champs ! [<a href="/">Retour</a>]</h4>'
230#        req ="INSERT INTO spectacles (titre, date, prix_pl, vendues) "\
231#            "VALUES (?, ?, ?, ?)"
232#        msg =BD.executerReq(req, (titre, date, float(prixPl), 0))
233#        if msg: return msg        #رسالة خطأ
234#        return self.index()       #الرجوع إلى الصفحة الرئيسية
235#    memoSpectacles.exposed =True
236#
237#    def toutesReservations(self):
238#        #عرض الحجوزات التي أدلى بها كل عميل
239#        req ="SELECT titre, nom, e_mail, COUNT(place) FROM spectacles "\
240#            "LEFT JOIN reservations USING(ref_spt) "\
241#            "LEFT JOIN clients USING (ref_cli) "\
242#            "GROUP BY nom, titre "\
243#            "ORDER BY titre, nom"
244#        res =BD.executerReq(req)
245#        #: إنشاء جدول HTML لعرض المعلومات الموجودة
246#        tabl ='<table border="1" cellpadding="5">\n'
247#        tabs =""
248#        for n in range(4):
249#            tabs +="<td>{{{0}}}</td>".format(n)
250#        ligneTableau ="<tr>" +tabs +"</tr>\n"
251#        #: الصف الأول من الجدول يحتوي على رؤوس الأعمدة
252#        tabl += ligneTableau.\
253#            format("Titre", "Nom du client", "Courriel", "Places réservées")
254#        #: الصفوف التالية
255#        for tit, nom, mail, pla in res:
256#            tabl += ligneTableau.format(tit, nom, mail, pla)
257#        tabl +="</table>"
258#        return mep(Glob.html["toutesReservations"].format(tabl))
259#    toutesReservations.exposed =True
260#
261#  #=== البرنامج الرئيسي ===
262#  #: فتح قاعدة البيانات - يتم إنشائها إذا كانت غير موجودة
```

```
263#  BD =GestionBD(Glob.dbName)
264#  BD.creaTables(Glob.tables)
265#  #: تحميل "علامات" صفحات في قاموس عام
266#  chargerPatronsHTML()
267#  #: إعادة تكوين و بدء خادم الويب
268#  cherrypy.config.update({"tools.staticdir.root":os.getcwd()})
269#  cherrypy.quickstart(WebSpectacles(), config ="tutoriel.conf")
```

في نهاية السكريبت, نجد كالعادة أسطر قليلة من البرنامج الرئيسي, و التي ستكون مسؤولة عن تمثيل الكائن BD للتواصل مع قاعدة البيانات, و تحميل "رؤساء" HTML في قاموس **Glob.html** و بدء عمل خادم الويب Cherrypy لتضمين مرجع الصنف الرئيسي لمعالج الإستعلامات .

السطر 272 يضمن الدليل الجذر للموقع و الذي هو الدليل الحالي .

"رؤساء" ال HTML

"رؤساء" ال HTML تستخدم من قبل السكريبت (كسلاسل تنسيق) في ملف نصي واحد(**spectacles.htm**), و نحن سوف نعيد إنتاجه بالكامل أدناه :

```
1#    [*miseEnPage*]
2#    <html>
3#    <head>
4#    <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
5#    <link rel=stylesheet type=text/css media=screen href="/annexes/spectacles.css">
6#    </head>
7#    <body>
8#    <h1><img src="/annexes/python.gif">Grand Théâtre de Python City</h1>
9#    {0}
10#   <h3><a href="/">Retour à la page d'accueil</a></h3>
11#   </body>
12#   </html>
13#   ##########
14#   [*pageAccueil*]
15#   <form action="/identification" method=GET>
16#   <h4>Veuillez SVP entrer vos coordonnées dans les champs ci-après :</h4>
17#   <table>
18#   <tr><td>Votre nom :</td><td><input name="nom"></td></tr>
19#   <tr><td>Votre adresse courriel :</td><td><input name="mail"></td></tr>
20#   <tr><td>Votre numéro de téléphone :</td><td><input name="tel"></td></tr>
21#   </table>
22#   <input type=submit class="button" name="acces" value="Accès client">
23#   <input type=submit class="button" name="acces" value="Accès administrateur">
24#   </form>
25#   ##########
26#   [*accesAdmin*]
27#   <h3><ul>
28#   <li><a href="/entrerSpectacles">Ajouter de nouveaux spectacles</a></li>
29#   <li><a href="/toutesReservations">Lister les réservations</a></li>
30#   </ul></h3>
31#   ##########
32#   [*accesClients*]
33#   <h3>Bonjour, {0}.</h3>
34#   <h4>Veuillez choisir l'action souhaitée :<ul>
35#   <li><a href="/reserver">Réserver des places pour un spectacle</a></li>
```

```
36#   <li><a href="/finaliser">Finaliser l'enregistrement des réservations</a></li>
37#   <li><a href="/revoir">Revoir toutes les réservations effectuées</a></li>
38#   </ul></h4>
39#   ##########
40#   [*reserver*]
41#   <h3>Les spectacles actuellement programmés sont les suivants : </h3>
42#   <p>{0}</p>
43#   <p>Les réservations seront faites au nom de : <b>{1}</b>.</p>
44#   <form action="/reservations" method=GET>
45#   <table>
46#   <tr><td>La réf. du spectacle choisi :</td><td><input name="spect"></td></tr>
47#   <tr><td>Le nombre de places souhaitées :</td><td><input name="places"></td></tr>
48#   </table>
49#   <input type=submit class="button" value="Enregistrer">
50#   </form>
51#   Remarque : les réservations ne deviendront effectives que lorsque vous
52#   aurez finalisé votre "panier".
53#   ##########
54#   [*reservations*]
55#   <h3>Réservations mémorisées.</h3>
56#   <h4>Vous avez déjà réservé {0} place(s) pour {1} spectacle(s).</h4>
57#   <h3><a href="/reserver">Réserver encore d'autres places</a></h3>
58#   N'oubliez pas de finaliser l'ensemble de vos réservations.
59#   ##########
60#   [*entrerSpectacles*]
61#   <h3>Les spectacles actuellement programmés sont les suivants :
62#   {0}
63#   Spectacle à ajouter :
64#   <form action="/memoSpectacles">
65#   <table>
66#   <tr><td>Titre du spectacle :</td><td><input name="titre">></td></tr>
67#   <tr><td>Date :</td><td><input name="date">></td></tr>
68#   <tr><td>Prix des places :</td><td><input name="prixPl">></td></tr>
69#   </table>
70#   <input type=submit class="button" value="Enregistrer">
71#   </form>
72#   </h3>
73#   ##########
74#   [*toutesReservations*]
75#   <h4>Les réservations ci-après ont déjà été effectuées :</h4>
76#   <p>{0}</p>
77#   ##########
78#   [*revoir*]
79#   <h4>Réservations effectuées par :</h4>
80#   <h3>{0}</h3><h4>Adresse courriel : {1} - Tél : {2}</h4>
81#   <p>{3}</p>
82#   ##########
```

مع هذا المثال المطور قليلا, نأمل أنك قد فهمت الفائدة من تعليمات البرمجية للبايثون المنفصلة و كودات HTML في ملفات منفصلة, كما فعلنا, حتى يختفظ برنامجك بالحد الأقصى من قابلية القرائة . تطبيق الويب هو في الواقع مقصود به في كثير من الأحيان أن ينمو و يصبح أكثر تعقيدا مع مرور الوقت . لذا يجب عليك أن تضع كل فرص جانبك لتظل دائما منظما و سهل الفهم . بإستخدام تقنيات حديثة مثل برمجة الشيئية, و أنت كنت بالتأكيد على الطريق الصحيح لتنمو بسرعة و تصبح مثمر للغاية .

تمارين

السكريبت السابق يمكن إستخدامه لتنفيذ إختبارات المهارة الخاصة بك في العديد من المجالات .

17.1 كما هو موضح أعلاه, يمكننا تنظيم موقع ويب على شبكة الإنترنات عن طريق تقسيم إلى عدة أصناف . قد يكون من الحكمة الفصل بين أساليب "العملاء" و "الإدراة" للموقع في أصناف مختلفة .

17.2 كما هو, السكريبت لا يعم بالشكل الصحيح إلا إذا كان المستخدم قد قام بتعبئة جميع الحقول المتوفرة . و لذلك سيكون من المفيد إضافة سلسلة من التعليمات للسيطرة على القيم التي تم ترميزها, مع رسائل خطأ للمستخدم عند الضرورة .

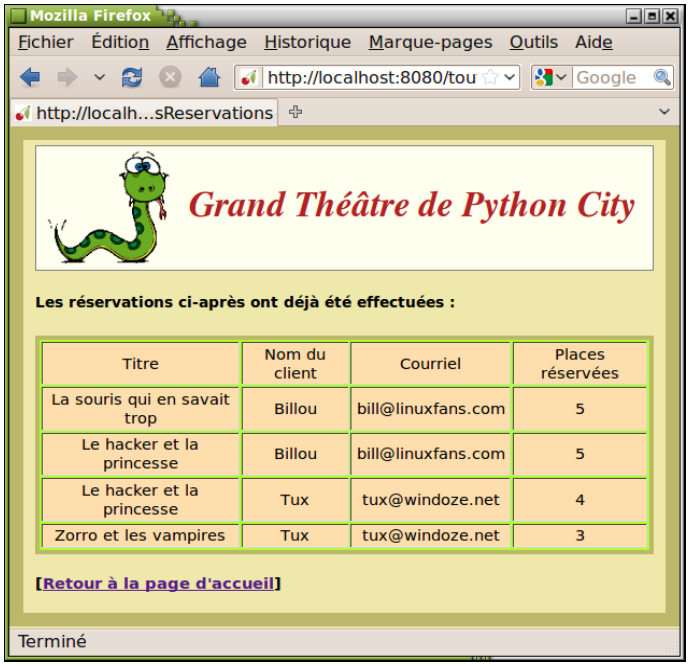
17.3 وصول المدراء يسمح فقط بإضافة عدد من عروض المسرحية مسارح,لكن لا يمكن تعديل أو حذف تلك المشفرة بالفعل . أضف إذا أساليب لتنفيذ هذه المهام .

17.4 وصول المدير حر . قد يكون من الحكمة إضافة سكريبت آلية لتوثيق بكلمة المرور حتى يقتصر الوصول فقط للذين يعرفون السمسم(إفتح يا سمسم) .

17.5 مستخدم "عميل" الذي يرتبط عدة مرات في كل مرة يتم تخزين فيها كعميل جديد, و من ثم ينبغي أن يكون قادر على إضافة المزيد من الحجوزات إلى حسابه, و ربما تعديل بياناته الشخصية, و ما إلى ذلك, يجب أن يتم إضافة كل هذه المميزات .

17.6 ربما قد لاحظة أن جداول HTML يتم إنشاؤها من خلال سكريبت في دالة **listeSpectacles()**, من جهة, و في أسلوب **revoir()** و **toutesReservations()**, من جهة أخرى يتم إنشاء خوارزمية مشابهة جدا . سيكون من المثير للإهتمام أن نكتب دالة عامة قادرة على إنتاج مثل هذا الجدول, التي تصلنا وصفها في قاموس أو في قائمة . إستلهم من دالة **listeSpectacles()** كقاعدة للبداية .

17.7 تزيين صفحات الويب تم إنشائها عن طريق سكريبت تم تعريفه في ملف CSS مرفق (**spectacles.css**) . لا تتردد في تجربة ما يحدث إذا قمت بإزالة ورقة نمط الإرتباط (السطر الخامس من ملف **spectacles.htm**) أو إذا قمت بتغيير محتوياته, و التي تصف أسلوب لتطبيقه على كل علامة .



تطويرات أخرى

إذا أردت جعل موقعك أكثر طموحا, يجب أن تأخذ عناء دراسة حزم أخرى مثل Karrigell, Django, Pylons, TurboGears, Twisted, Zope, Plone ... المرتبطة بنظام خادم أباتشي لنظاك الخادم, و MySQL أو PostgreSQL لمعالجة قواعد البيانات . عليك أن تعرف أن هذا المجال واسع جدا, حيث يمكنك ممارسة إبداعاتك لفترة طويلة ...

18

الطباعة مع البايثون

عملنا جاهدين حتى الأن لتعلم كيفية تطوير تطبيقات يمكن إستخدامها حقا . لمزيد من التقدم في هذا الإستجاه, سوف نكتشف الان كيف يولد, كع هذه التطبيقات, مستندات قابلة للطباعة بجودة عالية . هذا يعني أننا سوف نكون قادرين على برمجة صفحات مطبوعة تحتوي على نصوص و صور و رسومات ...

> القليل من التاريخ
>
> عندما ظهرت أول أجهزة الحاسوب في منتصف القرن الماضي, فإنهم كانوا يفكرون في تقنيات الطباعة على الورقة النتائج التي تنتجها برامج الحاسوبية . و في وقت هذه الالات, لم تكن الشاشات موجودة بعد, و كانت أساليب تخزين المعلومات بدائية جدا و مكلفة للغاية . لذلك تم إختراع الطابعات الأولى (بتكيي ال "TTY" المعاصرة) ليس فقط للتشاور, و لكن لتخزين البيانات المنتجة أيضا .
>
> هذه الالات الأولى لم تكن تطبع في البداية سوى أرقام و أحرف كبيرة غير معلمة, و التي كان كافيا في وقت الذي كان يستخدم فيه أجهزة الحاسوب في حل المشاكل العلمية و الحسابات للشركات الكبيرة .
>
> هذا الوضع لم يتغير كقير إلا أن ظهرت أجهزة الحاسوب الشخصية الأولى : في بداية الثمانينات', في الواقع, الطابعات الأولى (عامة جدا) لا تستطيع طباعة سوى نص . و لغات البرمجة في ذلك الوقت كانت قادرة على إدارة الطباعة على الورق بإستخدام بعض التعليمات البسيطة : في الواقع كانت تكفي أن تكون قادرة على

> شحن هذه الطابعات لطباعة سلاسل الأحرف, تتخللها بعض الرموز "الغير مطبعية" المخصصة لأوامر خاصة : علامة تبويت, نهاية أسطر, نهاية صفحات ... إلخ
> هذه التعليمات بسيطة, على سبيل المثال تعليمة *LPRINT* للغة *BASIC*, تستخدم نفس الطريقة لعرض الحروف على الشاشة, أو حفظ سطور النصوص في ملف (مثل الدالة *print*() أو أسلوب *write*() للبايثون) .

الطابعات الحديثة متنوعة جدا و فعالة : معظم الطاعبات الرسومية, التي تطبع أكثر مجموعات المحارف معرفة مسبقا في خط واحد, لكن مجموعات من النقاط الصغيرة من ألوان مختلفة . حتى لا يكون هنالك المزيد من التقييد يمكن للطابعة أن تكون : محاذات الأحرف المطبعية المختلفة من أي أبجدية في مجموعة متنوعة بلا حدود للخطوط و الأحجام, لكن أيضا للصور و الرسومات من أي نوع, أو صور فتوغرافية .

من الواضح أن هذا ذا فائدة للمستخدم النهائي, لأنه يمكن الحصول على مجموه من الإمكنيات الإبداعية الهائلة, لم يجروء أحد على حلم الطباعة مرة واحد , لمبرمج و مع ذلك, هذا يعني أولا تعقيد أكبر بكثير .

للتحكم في طباعنة نص إلى طابعة حديثة, فإنه لا يكفي إرسال سلسلة من الأحرف من النص, كما كنا نحفظ النص نفسه . يجب علينا ان ننظر أن كل حرف هو رسم صغير التي تقوم الطابعة بإعادة رسمه نقطة نقطة في منطقة معينة من الصفحة . يجب أن يكون برنامج الحاسوب يهدف على إعداد تقارير مطبوعة من بين الأمور الأخر لمكتبة رموز[91] أو أكثر مقابلة للخطوط و الأنماط التي نريد إستخدامها, و الخوارزميات الفعالة لترجمة هذه الرموز بمصفوفات من النقاط, مع دقة محددة و في حجم معين .

لنقل كل هذه المعلومات, لابد من بغة حقيقية محددة . كما كنت تتوقع, يوجد العديد من هذه اللغات, مع وجود إختلاف في تحكم مختلف النماذج و العلامات التجارية من الطابعات في السوق .

يمكن للواجهة الرسومية المساعدة

91يسمى رسم لمحرف بحرف رسومي (أنظر أيضا إلى صفحة Error: Reference source not found).

في بداية هذا الكتاب, شرحنا بإختصار أن هنالك تسلسل هرمي للغات الحاسوب, سواء لبرمجة جسم من تطبيق, ليقوم بإعادة إرسال إستعلام لخادم قواعد البيانات, لوصف صفحة ويب, ... إلخ . أنت تعلم جيدا أن هنالك لغات ذات مستوى منخفض, يكون فيها تكوين الجمل غامضا في كثير من الأحيان, و مرهقة لأنها تتطلب إستخدام تعليمات عديدة لأي أمر, لكن لحسن الحظ يوجد أيضا لغات ذات مستوى عال (مثل البايثون), إن إستخدام تكوين جملتها ممتع للغاية لأن يشبه (نوعا ما) اللغة البشرية (الإنكليزية في أغلب الأحيان), و هي أكثر كفاءة بشكل عام بسبب تنفيذها السريع .

الذي يتحاور مع الطابعات الحديثة, هو نظام تشغيل الحاسوب. الذي يعالج باللغات منخفضة المستوى . فهي لا تستخدم سوى لكتابة برامج تسمى المترجمات الخاصة بالتحكم بالطابعاتو و هذه يتم تقديمها عادة من قبل شركات المصنع لهذه الطابعات, و مكتبات البرامج التي توفر واجهة مع لغات برجمة عامة الأكثر .

لأسباب واضح لتنسيث, مكتبات البرامج تسمح بصنع هذه التطبيقات مع واجهة رسومية التي تقوم بإظهار صور بجميع أنواعها في نوافذ الشاشة (نص منسق, صور نقطية, إلخ) هي فالبا ما تكون قادرة على إدارة تنفيذ صفحة الطباعة من نفس الصور . يتم دمج بعض من هذه المكتبات في نظام التشغيل نفسه (في حالة API لنظام ويندوز), لكن هنالك إعتماديات أقل, والتي يجب عليك أن تعطي ما تفضله . أليس من المؤسف, بل تحد من قابلية برامج سكريبتات البايثون ؟

مكتبات الواجهة الرسومية تتوفر للغة برمجتك . فهي توفر أصناف كائنات تسمح ببناء صفحات طباعة بمساعدة الويدجات, تشبه قليلا عندما تصنع نوافذ على الشاشة . مع Tkinter, على سبيل المثال, يمكنك إعداد رسم صفحة للطباعة داخل اللوحة, ثم إرسال تمثيلها إلى الطابعة, شرط أن هذه الطابعة "تفهم" لغة الطباعة PostScript (للأسف غير منشرة بكثرة) . مكتبات واجهة أخرى مثل WxPython أو PyQt تقدم المزيد من الإمكنيات .

و الميزة الرئيسية لتقنيات الطباع أنها تستغل مكتبات واجهة المستخدم الرسومية, هو في الواقع أن في سكريبت البايثون تكتب أساس التحكم كله : لذلك يمكنك التأكد من أنه صفحة الطباعة صنعت قطعة قطعة في الشاشة من قبل المستخدم, و تظهر كما ستظهر في الطباعة, و تبدأ في طباعة نفسها بالطريقة التي تناسبك .

هذه التقنية لديها بعض العيوب : قابلية نقلها لا تعمل في جميع أنظمة التشغيل, و تنفيذها يتطلب تعلم مفاهيم عابس إلى حد ما (سياق الجهاز, إلخ). بالإضافة إلى ذلك, فإن هذا منهج يبين لك حدود عند النظر لإنتاج مستندات مطبوعة بحجم معين, مع نصوص التي تحتوي على فقرات متعددة, تتخللها أرقام, و تتحكم في اي واحدة منها بدقة في التخطيط, مع وجود إخلاف في النمط, و المسافات, و علامات التبويت, و إلخ ...في بداية هذا الكتاب, شرحنا بإختصار أن هنالك تسلسل هرمي للغات الحاسوب, سواء لبرمجة جسم من تطبيق, ليقوم بإعادة إرسال إستعلام لخادم قواعد البيانات, لوصف صفحة ويب, ... إلخ . أنت تعلم جيدا أن هنالك لغات ذات مستوى منخفض, يكون فيها تكوين الجمل غامضا في كثير من الأحيان, و مرهقة لأنها تتطلب إستخدام تعليمات عديدة لأي أمر, لكن لحسن الحظ يوجد أيضا لغات ذات مستوى عال (مثل البايثون), إن إستخدام تكوين جملتها ممتع للغاية لأن يشبه (نوعا ما) اللغة البشرية (الإنكليزية في أغلب الأحيان), و هي أكثر كفاءة بشكل عام بسبب تنفيذها السريع .

الذي يتحاور مع الطابعات الحديثة, هو نظام تشغيل الحاسوب. الذي يعالج باللغات منخفضة المستوى . فهي لا تستخدم سوى لكتابة برامج تسمى المترجمات الخاصة بالتحكم بالطابعاتو و هذه يتم تقديمها عادة من قبل شركات المصنع لهذه الطابعات, و مكتبات البرامج التي توفر واجهة مع لغات برجمة عامة الأكثر .

لأسباب واضح لتنسيق, مكتبات البرامج تسمح بصنع هذه التطبيقات مع واجهة رسومية التي تقوم بإظهار صور بجميع أنواعها في نوافذ الشاشة (نص منسق, صور نقطية, إلخ) هي فالبا ما تكون قادرة على إدارة تنفيذ صفحة الطباعة من نفس الصور . يتم دمج بعض من هذه المكتبات في نظام التشغيل نفسه (في حالة API لنظام ويندوز), لكن هنالك إعتماديات أقل, والتي يجب عليك أن تعطي ما تفضله . أليس من المؤسف, بل تحد من قابلية برامج سكريبتات البايثون ؟

مكتبات الواجهة الرسومية تتوفر للغة برمجتك . فهي توفر أصناف كائنات تسمح ببناء صفحات طباعة بمساعدة الويدجات, تشبه قليلا عندما تصنع نوافذ على الشاشة . مع Tkinter, على سبيل المثال, يمكنك إعداد رسم صفحة للطباعة داخل اللوحة, ثم إرسال تمثيلها إلى الطابعة, شرط أن هذه الطابعة "تفهم" لغة الطباعة PostScript (للأسف غير منشرة بكثرة) . مكتبات واجهة أخرى مثل WxPython أو PyQt تقدم المزيد من الإمكنيات .

و الميزة الرئيسية لتقنيات الطباع أنها تستغل مكتبات واجهة المستخدم الرسومية, هو في الواقع أن في سكريبت البايثون تكتب أساس التحكم كله : لذلك يمكنك التأكد من أنه صفحة الطباعة صنعت قطعة قطعة في الشاشة من قبل المستخدم, و تظهر كما ستظهر في الطباعة, و تبدأ في طباعة نفسها بالطريقة التي تناسبك .

هذه التقنية لديها بعض العيوب : قابلية نقلها لا تعمل في جميع أنظمة التشغيل, و تنفيذها يتطلب تعلم مفاهيم عابس إلى حد ما (سياق الجهاز, إلخ). بالإضافة إلى ذلك, فإن هذا منهج يبين لك حدود عند النظر لإنتاج مستندات مطبوعة بحجم معين, مع نصوص التي تحتوي على فقرات متعددة, تتخللها أرقام, و تتحكم في اي واحدة منها بدقة في التخطيط, مع وجود إخلاف في النمط, و المسافات, و علامات التبويت, و إلخ ...

لغة تصف صفحة للطباعة, PDF

في الواقع, ترتبط هذه القيود بهذه المشكلة التي تحدثنا عنها : فإذا إستخدمنا هذه التقنيات, مستوى اللغة المنخفض جدا يستخدم في التواصل مع الطابعة يتطلب منا إعادة إختراع في سكريبتاتنا آليات معقدة, في حين أنها قد صنعت و أهتبرت و صقلت من قبل فريق من المطورين المهرة .

> عندما كنا في الفصل 16 قمنا بشرح إدارة قواعد البيانات, على سبيل المثال, رأينا أنه من الأفصل أن توكل هذه المهمة المعقدة إلى نظام برمجي متخصص (محرك قاعدة البيانات, *SGBDR*), بدلا من محاولة إختراع كل شيئ في سكريبتاتنا . و يمكن لهذه التعليمات ببساطة توليد تعليمات للغة عالية المستوى (*SQL*) مناسبة بشكل مثالي لوصف الإستعلامات الأكثر تعقيدا, و السماح ل *SGBDR* بفكها و تشغيلها على نحو أفضل .

بطريقة مختلفة قليلا, سوف نعرض لك في الصفحات التالية كيف يمكنك بسهولة بناء عن طريق بايثون بتعليماتها عالية المستوى لتصف و بقدر كبير من التفاصيل أن الطابعة يجب أن تظهر على صفحة مطبوعة . هذه اللغة هي PDF (تنسيق المستندات المحمولة) .

> وضعت أصل من قبل *Adobe Systems* في عام 1993 لوصف الوثائق التي سيتم طباعتها بطريقة مستقلة تماما من أي تظام برنامجي أو نظام تشغيل, ال *PDF* يتم تقديمه في الكثير من الأحيان كتنسيق ملف, لأن عادة ما يتم إنشائه من خلال مكتبات دالات أو من خلال برامج متخصصة كسكريبتات كاملة . و هذا يعني لغة وصف للصفحات متطورة جدا, التي إكتسبت صفة معايير العالمية .
>
> في البداية كان خاص, و أصبح هذا المعيار مفتوح في عام 2008 . لا تقلق بشأن متانته أو حرية إستخدامه مع تطبيقاتك .

مستند PDF هو سكريبت يمكنه وصف النصوص و الخطوط و الأنماط و الكائنات الرسومية و تخطيط من مجموعة من الصفحات للطباعة, التي يجب أن تعاد بنفس الطريقة, بغض النظر عن التطبيق و المنهاج المستخدم منصة لقراءته .

مميزة من مميزات لغة PDF أنها عامة غير مفهومة بسعولة من قبل الطابعات العادية . لتفسيرها فمن الضروري إستخدام برامج متخصصة مثل Acrobat Reader, Foxit reader, Sumatra reader, Evince...

لا ينبغي أن تنظر إليه على أنه عيب, لأن هذه البرامج لديها ميزة كبير بالسماح لمعاينة النص للطباعة . يمكن لمستند PDF إذا أرشفته و إلخ . دون الحاجة إلى طباعة واقعية . كل هذه البرامج مجانية, و يوجد على ما لا يقل عن واحد منهم في تكوين القياسي من أي جهاز حاسوب حديث .

هذا الفصل لا يهدف إلى شرح كيفية بمكنك التحكم مباشرة بالآلة طبيعة من سكريبت بايثون . بدلا من ذلك, سوف يظهر لك كم هو من السهل صنع مستندات PDF ذات جودة عالية بإستخدام تعليمات قوية و مقروءة . هذا المنهج يضمن قابلية البرامج الخاصة بك في حين يسمح لهم بقدرات الطباعة واسعة النطاق, و التي سوف تناسب بشكل جيد تطبيقات التي تقوم بتطويرها للويب .

معظم ما نحتاحه متاح في وحدة بايثون موزعة بترخيص حر : مكتبة الأصناف ReportLab .

> في الواقع إن *ReportLab* هو إسم شركة لندنية التي تطور مجموعة من أصناف البايثون لتوليد وثائق *PDF* ذات جودة عالية برمجيا . و تقوم الشركة بتوزيع جميع

> مكتباتها لقاعدة النظام برخصة حرة, مما يسمح لك بإستخدامها بحرية . و هي تقوم ببيع تحت رخصة ملكية مختلفة البرامج تكميلية لمعالجة *PDF*, و تطوير التطبيقات المخصصة للشركات . و لكنالمكتبات الأساسية سوف تكون أكثر من كافية لجعلك سعيدا .

في هذه المرحلة من التفسيرات, ينبغي لنا أن نكون بالفعل قادريت على أن نقدم مثال سكريبت صغير يظهر لك كيفية إستدعاء مكتبة ReportLab و صنع ملفات PDF بدائية . للأسف هذا غير ممكن في الوقت الحالي, لأننا يجب أن نحل مشكلة أولا : وحدة ReportLab الوحيدة المتاحة في وقت كتابة هذه السطر (كانون الأول/ديسمبر 2011) للإصدار 2 للبايثون . لا يوجد حتى الأن وحدة ReportLab للبايثون 3 !

إذا, ماذا نفعل ؟

تثبيت بايثون 2.6 أو 2.7 لإستخدام وحدات البايثون 2

في نهاية سنة 2008, قرر فريق تطوير البايثون كسر لأول مرة توافق إصدارات المتتالية للغة بايثون من 2.5 إلى 3.0, و أعربوا عن أملهم أن في هذا الإصدار الجديد سوف يعتمد عليه بسرعة من قبل المطوري مكتبات الطرف الثالث مثل ReportLab . لكن لسوء الحظ, لم يكن هذا الحال : العديد من الملحقات متاحة منذ فترة طويلة للبايثون 2 و مايزالون غير متواجدين للبايثون 3 (ليس فقط ReportLab, و لكن حتى PIL (Python Imaging Library), py2exe, Django ...إلخ) . أنت تعرف أن هذا الوضع محرج, و إذا تلاحظ أيضا أن العديد من الشركات لا تزال تفضل إستخدام البايثون 2 على الرغم من عيوبها القليلة, بدلا من إعتماد الإصدار 3 و هو ما يتطلب منهم تحويل العديد من السكريبتات الموجودة, و لقد أصدر المطوري بايثون بسرعة نسخة من التحول للغة, النسخة 2.6, ثم الثانية (من المفترض أن تكون الأخيرة) الإصدار 2.7 .

هذان الإصداري بايثون متوافقين تماما مع كافة الإصدارات السابقة, و لكنها تقبل حيثما كان ذلك ممكنا تكون جمل التي أدخلت في البايثون 3, بالإضافة إلى العديد من وظائف الجديدة .

تخضع لتغييرات طفيفة في الأسطر الأولى, يتم تشغيل سكريبتات البايثون 3 تماما مثل الملفات 2.6 أو 2.7, و الذي يسمح لهم بإستخدام جميع مكتبات الجهات الأخر التي غير متاحة بعد للبايثون 3 !

لبقية هذا الفصل, فإنه نفترض بالإضافة إلى البايثون 3, مثية أيضا بايثون 2.6 أو 2.7 على محطة العمل الخاصة بك . لا تقلق من تثبيت بايثون 2 و بايثون 3 على نفس الجهاز / هذه الإصدارات مستقلة عن اللغة لا تعوق بعضها البعض .

> ليعلم النظام مع أي إصدار بايثون لتشغيل سكريتاتك . حدد ببساطة في بداية الأمر :
>
> `python nom_du_script` ليعمل بالبايثون 2 .
>
> `python3 nom_du_script` ليعمل بالبايثون 3 .

العديد من سكريبتات الفصول السابقة من هذا الكتاب تعمل دون تغيير تقريبا للبايثون 2.6 أو 2.7 . على سبيل المثال, تقريبا جميع سكريبتات فصول 8, 13, 14, 15 (برمجة واجهة المستخدم الرسومية) تعمل بدون مشاكل إذا إستبدلت ببساطة "t" الصغيرة في التعليمة : **from tkinter ***
import ب "T" كبيرة : **from Tkinter import *** .

كل واحد من إصدارات البايثون لديه في الواقع إصدار خاص لمكتبة Tkinter, الإختلاف بيت الأسماء متعمد لتجنب إستدعاء وحدة نمطية عن طريق الخطأ عند إصداري بايثون (مع مكتبات Tkinter لكل واحدة منهم) موجود على نفس الجهاز .

كما أنه من السهل جدا تغيير هذه السكريبتات لتعمل على أي إصدار من إصدارين . يكفي ببساطة تغيير بعض التعليمات في بداية البرنامج النصي للكشف عن أي إصدار بايثون يستخدم, و بالتالي يتم تغيير اللازم .

للقيام بذلك, يمكنك على سبيل المثال إستخدام وحدة **sys** لمكتبة القياسية, بسمة **version** التي تشير إلى إصدار مفسر البايثون المستخدم لتشغيل السكريبت (في شكل سلسلة تحتوي على عدد قليل من المعلومات[92]). مظاهر:

```
>>> import sys
>>> sys.version
'2.7.1+ (r271:86832, Apr 11 2011, 18:05:24) \n[GCC 4.5.2]'
```

92سلسلة تشير إلى رقم إصدار الكامل (2.7.1+ في المثال أعلاه), و رقم و تاريخ تجميع للمترجم, و رقم النسخة المستخدمة (GCC 4.5.2 في مثالنا) .

في بداية سكريبتاتك, يجب أن تشمل الأسطر التالية :

```
#2) تحديد نسخة بايثون المستخدمة .xx أو 3.xx) :
import sys
if sys.version[0] =='2':        # الحرف الأول من السلسلة تكفينا
    from Tkinter import *       # لبايثون 2 وحدة Tkinter
else:
    from tkinter import *       # لبايثون 3 وحدة Tkinter
```

يمكنك ايضا إستخدام إحدى تقنيات من التقنيات الأخرى الأكثر "همجية", على سبيل المثال محاولة إستيراد مكتبات وراء تعليمة try :, و ترتد إليك إذا لم تعمل (أنظر إلى معالجة الإستثناءات, صفحة 117) .

```
try:
    from tkinter import *
except:
    from Tkinter import *
```

مع الكود المصدري للأمثلة في هذا الكتاب الذي هو تحت تصرفك على الويب (أنظر للصفحة XIII), نحن نقدم لك في دليل الفرعي py3onpy2 إصدارات "مختلطة" من سكريبتات الفصول 8 و 13 و 14 و 15 : و التي يمكنك تشغيلها إما تحت بايثون 3 أو بايثون 2 .

جميع السكريبتات تستخدم واجهة المستخدم الرسومية Tkinter لمدخلاتها/مخرجاتها القابلة بسهولة على التكيف مع أي نسخة من البايثون, حقيقة بسيطة و هي أن هنالك إصدارات من هذه مكتبة الواجهة, و أنت تستخدم واحد أو الأخرى حسب الحالة . و مع ذلك فإن الحالة تتعقد إلى حد ما للسكريبتات التي تدير بنفسها المدخلات/المخرجات, و هذا معناه السكريبتات التي تستخدم الدالات مثل **input()** و **print()** (وحدة التحكم بطرفية, لوحة المفاتيح/الشاشة), أو **open()** (قراءة أو كتابة حروف في ملف) .

لقد رأينا في الفصل 10 أن واحدة من أهم التغييرات في الإصدار 3 من البايثون هو إعادة تعرف نوع string كما يجري الأن في سلسلة أحرف Unicode, بدلا من سلسلة من البيتات (و التي تتطابق مع نوع byte) . تواصل الحاسوب مع ملحقاته يزال أدائه بالبايتات . و لذلك يجب على جميع التعليمات

التي تتحكم في إخرا الأحرف الأن تشمل الأن آلية ترميز, و جميعها تدير آلية لفك أحرف المدخلات, و هذا الذي لم يكن في إصدارات البايثون "القديمة" .

يبقى فقط تكيف بسيط لسكريبتات بايثون 3 بإستخدام هذه التعليمات, بحيث يمكن تشغيلها تحت بايثون 2.6 و 2. في كلا الإصدارين, هو من الممكن فرض توافقية من العديد من دالات الباثون 3 مع تعليمة إستدعاء خاصة, لتكوين الجملة : ***** **from __future__ import** . مع هذه التعليمة, فإن البايثون 2 سيقون بإستبدال ***** بما يعادلها من البايثون 3 بإستدعاء ببساطة وحدة الخاص **__futur__** .

لنبدأ, سوف نعمل على معالجة السلاسل النصية الحرفية (و هذا يعني سلاسل نصية تم تعريفها في سكريبت نفسه) و لتعامل على أنها سلاسل Unicode, و ذلك بإستخدام تعليمة :

```
from __future__ import unicode_literals
```

و سوف نضمن بعد ذلك أن الدالة **print()** للبايثون 3 يمكن إستخدامها في تعليمة print للبايثون 2 (التي لا تزال وظيفية), و ذلك بإضافة التعليمة :

```
from __future__ import print_function
```

يجب أن تكون الإستدعاءات أعلاه في بداية السكريت . لا يمكن وضعها في كتلة التعليمات التي تتبع الكشف عن نسخة البايثون المستخدمة . و هذه ليس مشكلة, لأنه بمجرد أن يمكنك تجاهلها إذا شغلت السكريبت في البايثون 3 .

الدالة **input()** للبايثون 2 تعمل بشكل مختلف عن البايثون 3 . و مع ذلك يمكننا إعادة تعريف بسهولة السكريبت نفسه, كما في التعليمات البرمجية أدناه :

```
1#    import sys
2#    if sys.version[0] =="2":
3#        encodage =sys.stdout.encoding
4#        def input(txt =""):
5#            print(txt, end="")
6#            ch =raw_input()
7#            return ch.decode(encodage)
```

الوحدة sys يتم إستدعائها في السطر 1 تسمح بتحديد في كل مرة إصدار البايثون و الترميز في نص الطرفية المستخدمة . سمته **stdout.encoding** تحتوي في الواقع على "cp437", "cp1252", "cp850"" أو "UTF-8" بعد أن تقوم بتشغيل السكريبت في طرفية سابق . نافذة

MSDOS أو نافذة IDLE (واجهة المستخدم الرسومية للبايثون) في ويندوز XP, أو على نظام تشغيل حديث .

دالة **input()** للبايثون 3 تعمل مثل دالة **raw_input()** في البايثون 2, لكنها تستخدم سلاسل نصية Unicode, في حين أن Python 2 تستخدم سلاسل نصية بايتات . و هذا يفرض علينا إستخدام الدالة **print()** لعرض النص الذي تم وضعه كبرامتر لإستدعاء الدالة (السطر 5), و لفك ترميز الإخراج (السطر 7) سلسلة بايتات التي سوف تتلفقاها من لوحة المفاتيح في ترميز واضح, و التي يمكن لحسن الحظ تحديدها بواسطة السطر 3 .

فيما يتعلق بعمليات القراءة/الكتابة في ملفات, أخيرا, يجب إستبدال دالة **open()** القياسية للبايثون 2, التي لا تؤدي إلى أي معابجة على السلاسل البايتات المنتقلة, واحد من برامج ترميز الوحدة, و التي تعمل مثل البايثون 3[93] :

```
from codecs import open
```

هذا إلى حد كبير ... على الأقل فيما يتعلق بمفاهيم التي تمت مناقشتها في هذا الكتاب للمبتدئين . كما و سبق ذكره أعلاه, سوف تجد في الدليل الفرعي py3onp2 تعليمات برمجية مفتوحة للكتاب, موجودة على الويب, و بعض الأمثلة على هذه السكريبتات "المختلطة" في أي بايثون 2 أو 3 (أنظر على سبيل المثال حل تمرين 10.45) .

> بإختصار, يمكنك الإستمرار في تعلم البرمجة بإستخدام البايثون 3, دون أي خوف من أن تجد نفسك في طريق مسدود . إذا وجدت في أي من مشاريعك دالات مكتبات ليست متاحة بعد للبايثون 3, يمكنك بسهولة جدا ضبط سكريبتاتك لتنفيذها مؤقتا تحت البايثون 2.7 (أو حتى 2.6) و إستغلال جميع مكتباتها, بإنتظار نسخة بايثون 3 .

تشغيل مكتبة ReportLab

93حول هذا الموضوع, راجع : "التحويلات التلقائية عند معالجة الملفات", في صفحة Error: Reference source not found.

يمكننا الأن أن نكتب أول سكريبتاتنا لتوليد مستندات PDF . هذه السكريبتات تم كتابتها بتكوين جملة بايثون 3, كما يتم تنفيذ كافة برامج النصية في هذا الكتاب, لكن يمكن ذلك (موقتا) في البايثون 2.6 أو 2.7 . عندما تكون مكتبة ReportLab متاحة للبايثون 3, الأسطر التكييف في بداية كل سكريبت يمكن حذفها . (لتثبيت ReportLab, أنظر إلى صفحة 382) .

أول مستند PDF بدائية

```
1#      ### مسودة وثيقة PDF مبسطة مصنوعة بإستخدام ReportLab ###
2#      # السكريبت مكتوب بالبايثون 3, لكن يمكن تشغيله بالبايثون 2.6 أو 2.7, بما أن المكتبة غير متوفرة #
3#
4#      from __future__ import unicode_literals      # بدون فائدة في البايثون 3
5#
6#      # : أستدعاء بعض عناصر مكتبة ReportLab
7#      from reportlab.pdfgen.canvas import Canvas    # صنف كائنات "لوحة"
8#      from reportlab.lib.units import cm            # قيمة 1 سم في نقطة 1 بيكا
9#      from reportlab.lib.pagesizes import A4        # أبعاد A4
10#
11#     # 1) : إختيار إسم الملف لوثيقة التي سيتم صنعها
12#     fichier ="document_1.pdf"
13#     # 2) : تمثيل "كائن لوحة" مرتبط لهذا الملف Reportlab :
14#     can = Canvas("{0}".format(fichier), pagesize=A4)
15#     # 3) : تركيب عناصر مختلفة على اللوحة
16#     texte ="Mes œuvres complètes"          # سطر للطباعة
17#     can.setFont("Times-Roman", 32)         # إختيار الخط
18#     posX, posY = 2.5*cm, 18*cm             # موقعه على الورقة
19#     can.drawString(posX, posY, texte)      # رسم النص على اللوحة
20#     # 4) : حفظ النتيجة في ملف PDF
21#     can.save()
```

بعد تشغيل السكريبت, سوف تجد أن وثيقتك PDF الأولى في الدليل الحال, بإسم **document_1.pdf** و ستكون بشكل DIN A4[94] مع جملة صغيرة في الوسط "Mes œuvres complètes" تم صنها بخط Times-Roman ب 24 نقطة .

تحليل السكريبت الصغير يظهر لك أن مع ReportLab لديك لغة عالية المستوى لوصف صفحات مطبوعة . يمكنك أن تختصر الكود في 4 أسطر فقط, بإستخدام التعليمات المركبة !

* السطر 5 : التعليمة **from __future__ import unicode_literals** في بداية السكريبت تفرض مفسر البايثون 2 بتحويل السلاسل الحرفية في السكريبت إلى سلاسل Unicode (كما في البايثون 3) . بدون هذه التعليمة, سيتم ترميز السلاسل البايتات وفقا لمعيار المستخدم لمحرر النص

---

94 DIN (Deutsches Institut für Normung) هو معيار الوطني الألماني .

الخاص بك [95]. إذا كان المعيار ليس Utf-8 (على سبيل المثال,في حالة ويندوز XP), هذه السلاسل لا يمكن أن تكون مقبولة من قبل مكتبة ReportLab . و هذا لا يقبل في الواقع إلا سلاسل Unicode أو سلاسل بايتات مشفرة ب Utf-8 (إما) .

* السطر 8 : مكتبة ReportLab كبيرة . فنحن لن نستدعي سوى عناصر الضرورية لهذا العمل . واحدة من أهم الأصناف هي صنف **Canvas()**, التي لديها قدرة هائلة من الأساليب للتخلص من أي شيئ على صفحة مطبوهة : شظايا النص و الفقرات و الرسومات المتجهة, و الصور النقطية ... إلخ .

* الأسطر 9 و 10 : الإستدعاءات هنا هي ببساطة قيم لتحسين إمكانية قراءة الكود . وحدة القياس في ReportLab هي نقطة المطبعية بيكا, الذي هي 1/72 بوصة, هو 0.353 مم . A4 هي نفق بسيط (595.28, 841.89) تعبر عن حجم DINA4 في هذه الوحدة, و cm لقيمة سنتيمتر (28.346), التي من شأنها أن تستخدم على نطاق واسع في أمثلتنا, للتعبير عن أبعاد الأكثر وضوحا و مواقع على الصفحة .

* السطر 15 : تمثيل كائن لوحة . يجب أن يوفر المنشئ إسم الملف الذي سيتلقى الوثيق, جنبا إلى جنب, ربما مع البرامترات الإختيارية . حجم الصفحة الإفتراضية هي A4, لكن ليس سيئ بتحديد بشكل واضح .

* السطر 18 : الأسلوب **setFont()** يستخدم لتحديد خط الكتابة و حجمه . واحدة من النقاط القوة ل PDF يكمن في حقيقة أن جميع البرامج لتفسرها (Acrobat Reader ...إلخ) يجب أن تكون مكتبة الرموز نفسها للخطوط التالية : Courier, Courier-Bold, Courier-BoldOblique, Courier-Oblique, Helvetica, Helvetica-Bold, Helvetica-BoldOblique, Helvetica-Oblique, Symbol, Times-Bold, Times-BoldItalic, Times-Italic, Times-Roman, ZapfDingbats . هذه الخطوط تستخدم للعديد من الإستخدامات : هي في الواقع خك أحادي المسافة (Courier), و خطين مناسبين بدون رقيق (Times-Roman, Helvetica), كل واحد منها لديه أربعة أنكاط مختلفة (عادي, كبير, مائل, كبير و مائل) و أخيرا خطين من الرموز المختلفة (Symbol, ZapfDingbats).

95أنظر إلى صفحة Error: Reference source not found : «مشاكل ممكنة مع الأحرف المعلمة».

الحقيقة أن بالفعل "يعرف" برنامج المفسر يعفيك من وصفها في المستند نفسها : إذا أردت يمكنك الحصول على هذه الخطوط, و مستندات ال PDF و الإحتفاظ بحجم صغير جدا .

> أداء تضمين المزيد من الخطوط المتجهة ؟
>
> من الممكن تماما إستخدام خطوط أخرى متجهة (*TrueType* أو *Adobe Type 1*), لكن يجب بعد ذلك توفير وصف رقمي في الوثيقة نفسها, مما يزيد بشكل كبير من حجم و تعقيد بعض الأشياء . و نحن لن نشرح هذا في هذا الكتاب .

* السطر 20 : الأسلوب **drawString(x, y, t) positionne** يحدد موقع جزء النص **t** في اللوحة بمحاذات على يسار نقطة الإحداثيات **x** و **y** . من المهم جدا أن نلاحظ هنا أنه ينبغي تحديد هذه الإحداثيات في الزاوية اليسرى السفلى للصفحة, كما هي العادة في الرياضيات, و عادة لا يتم الإعتماد على الزاوية أقصى اليسار لإحداثيات الشاشة . فإذا تريد كتابة العديد من الأسطر النص المتتالي في الصفحة, يجب عليك التأكد من أن إحداثيات العمودية تقلل من الأول إلى الأخير .

* السطر 22 : الأسلوب **save()** تكمل العمل و تغلق الملف . و سوف نرى لاحقا كيفية توليد مستندات متعددة الصفحات.

توليد مستند أكثر تفصيلا

السكريبت التالي يولد مستند غريبة قليلا, لتسليط الضوء على بعض من الكثير من الإحتمالات التي أصبحت الأن متاحة لنا, بما في ذلك إعادة إنتاج صور نقطية على الصفحة . فإذا كنت ترغب في إستخدام هذه الميزة, يجب أن تثبت مكتبة معالجة الصور (Python Imaging Library (PIL)) و هي أداة ملحوظة التي يمكن أن تكون كائن كتاب وحدها . لاحظ أن مستندات المفصلة ل ReportLab و PIL متاحة على الويب, على الأقل باللغة الإنكليزية . شرح تثبيت PIL موجود في الصفحة 382 .

```
#=== مع أنواع مختلفة من المسارات PDF إنشاء وثيقة ===
#
#: تكييف السكريبت لجعله يعمل مع البايثون 2.6 أو 2.7
#(للبايثون 3 Reportlab يمكنك حذف هذه الأسطر إذا تم توفير)
from __future__ import unicode_literals
from __future__ import division          #تقسيم "حقيقي"
#-----------------------------------------------------------------
#
#: إستدعاء بعض عناصر مكتبة Reportlab
from reportlab.pdfgen.canvas import Canvas
from reportlab.lib.units import cm
from reportlab.lib.pagesizes import A4
#
fichier ="document_2.pdf"
can = Canvas("{0}".format(fichier), pagesize=A4)
#: تركيب عناصر المختلفة على اللوحة
largeurP, hauteurP = A4                  #عرض و إرتفاع الصفحة
```

```
18#    centreX, centreY = largeurP/2, hauteurP/2      #إحداثيات منتصف الورقة
19#    can.setStrokeColor("red")                      #لون الأسطر
20#    #تذكير : يتم حساب الموقع العمودي للصفحة من الأسفل .
21#    can.line(1*cm, 1*cm, 1*cm, 28*cm)              #خط عمودي على اليسار
22#    can.line(1*cm, 1*cm, 20*cm, 1*cm)              #خط أفقي في الأسفل
23#    can.line(1*cm, 28*cm, 20*cm, 1*cm)             #خط مائل (تنازليا)
24#    can.setLineWidth(3)                            #سمك جديد للخطوط
25#    can.setFillColorRGB(1, 1, .5)                  #لون التعبئة
26#    can.rect(2*cm, 2*cm, 18*cm, 20*cm, fill=1)     #مستطيل 18 × 20 سم
27#
28#    #يقوم بإرجاع إحداثيات الرسم dramImage رسم نقطي (محاذات الزاوية اليسرى السفلي) . لأسلوب
       النقطي (بالبيكسلات) في نفق :
29#    dX, dY =can.drawImage("cocci3.gif", 1*cm, 23*cm, mask="auto")
30#    ratio =dY/dX                                   #تقرير عرض\إرتفاع الصورة
31#    can.drawImage("cocci3.gif", 1*cm, 14*cm,
32#            width=3*cm, height=3*cm*ratio, mask="auto")
33#    can.drawImage("cocci3.gif", 1*cm, 7*cm, width=12*cm, height=5*cm,
       mask="auto")
34#
35#    can.setFillColorCMYK(.7, 0, .5, 0)             #لون التعبئة (CMJN)
36#    can.ellipse(3*cm, 4*cm, 19*cm, 10*cm, fill=1)  #16 = محاور (! بيضوي x 6 cm)
37#    can.setLineWidth(1)                            #سمك جديد للخطوط
38#    can.ellipse(centreX -.5*cm, centreY -.5*cm,    # دائرة صغير تشير إلى
39#            centreX +.5*cm, centreY +.5*cm)        #موقع منتصف الصفحة
40#
41#    #: بعض النصوص, مع خطوط و ألوان متجهة و متحاذية مختلفة
42#    can.setFillColor("navy")                       #لون النصوص
43#    texteC ="Petite pluie abat grand vent."        #نص في الوسط
44#    can.setFont("Times-Bold", 18)
45#    can.drawCentredString(centreX, centreY, texteC)
46#    texteG ="Qui ne risque rien, n'a rien."        #نص محاذات على اليسار
47#    can.setFont("Helvetica", 18)
48#    can.drawString(centreX, centreY -1*cm, texteG)
49#    texteD ="La nuit porte conseil."               #نص محاذات على اليمين
50#    can.setFont("Courier", 18)
51#    can.drawRightString(centreX, centreY -2*cm, texteD)
52#    texteV ="L'espoir fait  vivre."                #وضع النص عمودي
53#    can.rotate(90)
54#    can.setFont("Times-Italic", 18)
55#    can.drawString(centreY +1*cm, -centreX, texteV) #! قلب الإحداثيات !
56#    texteE ="L'exception confirme la règle"        #نص لإظهاره باللون الأبيض
57#    can.rotate(-90)                                #العودة إلى الإتجاه الأفقي
58#    can.setFont("Times-BoldItalic", 28)
59#    can.setFillColor("white")                      #لون جديد للنصوص
60#    can.drawCentredString(centreX, 7*cm, texteE)
61#
62#    can.save()                                     #حفظ النتيجة
```

تعليقات

* السطر 6 : في البايثون 2, عامل قسمة هو / ينفذ قسمة عدد صحيح بشكل إفتراضي (مقابلة لمعامل // في البايثون 3 - أنظر للصفحة 13) . سيتم فرض هنا وضع قسمة الحقيقية .

* السطور من 17 إلى 40 : هذه السطور تظهر لك بعض الأساليب التي تسمح برسم رسوم على الصفحة بإستخدام أسطر و الأشكال الأساسية . من الواضح, لا يمكننا تبسيطه أكثر هنا . فإذا أخذت عناء مراجعو المستندات المرجعية ل ReportLab (كتيبات PDF عديدة), سوف تجد الكم الهائل من الأساليب الأخرى التي تسمح بتحقيقي رسوم متجه لتحجيمها, و الرسوم البيانية و الجداول و الرسوم بيانية الدائري ... إلخ .

* السطور من 19 إلى 23 تعرف مثلث الذي هو يلخص محيط أحمر . الأسطر من 24 إلى 26 تسم محيط مستطيل أكثر سماكة و تملئه بلون أصفر شاحب . لاحظ أن ترتيب التعليمات أمر بالغ الأهمية : الرسومات التي تتداخل على التوالي . لاحظ مرة أخرى أن جميع الإحداثيات العمودية تقوم بالصعود, بداية من أسفل الصفحة .

* الأسطور 19 و 25 و 36 و 43 و 60 : في ReportLab يمكن تحديد الألوان بثلاثة طرق مختلفة . و أبسط هذه الطرق هي إستخدام أسماء الألوان باللغة الإنكليزية, لكن هذا لا يمكننا من إختيار الفروق الدقيقة الخاصة . يمكننا تحديد هذا عن طريق 3 مركبات تشير لألوان الضوء و هي الأحمر و الأخصر و الأزرق (باللغة الإنكليزية RGB) كما هو الحال لبكسلات الشاشة, أو أفضل من ذلك, لأنه صفحة مطبوعة, فهي تشير إلى 4 ألوان و هي سماوي و أرجواني و أصفر و أسود (باللغة الإنكليزية CMYK) و هي الأحبار المستخدمة لإنتاج صورة على ورق , و يجب أن تكون قيم كل مكون بين 0 و 1 .

* السطور من 28 إلى 34 : يستخدم الأسلوب **drawImage()** لإعادة إنتاج نفس صورة الخنفساء 3 مرات, في مواقع و أحجام مختلفة . هذا الإجراء ممكن بفضل دالات PIL . لاحظ أنه إذا إستخدمت نفس الصورة عدة مرات في مستند, لن يتم تحميلها سو مرة واحدة في PDF, عبر آلية تخزين مؤقت .

* السطر 30 : سوف نقوم بإعادة إنتاج صورة النقطية أول مرة دون تغيير حجمها . سوف يعتبر ReportLab أنه في هذه الحالة كل بكسل من الصوؤة يساوي (1/72 بوصة) . البرامتر الأول يمرر إلى الأسلوب **drawImage()** إسم الملف الذي يحتوي على الصورة, الصيغ المقبولة هي PNG و JPG و GIF

و TIF و BMP . البرامترات الثانية و الثالث هي إلزامية فهي تحدد إحداثيات الركن الأيسر السفلي على الصفحة . البرامتر الإختيار **mask="auto"** مطلوب إذا كنت تريد أن تجعل الصورة شفافة .[96].

* الأسطر من 31 إلى 33 : ميزة مفيدة لأسلوب **drawImage()** في تقوم بإرجاع أبعاد الصورة النقطية بالبكسل, في نفق من الأعداد الصحيحة . يمكنك إذا إستخدام هذه المعلومات في سكريبت الخاص بك, و كما هو الحال هنا لتحديد طول و إرتفاع لصورة, لإعادة رسمها بالإضافة إلى مستوى آخر, و ذلك بفضل البرامترات الإختيارية **width** و **height** .

* الأسطر من 34 إلى 40 : نعيد رسم الصورة مرة أخرة, و هذه المرة مع أي أبعاد, ثم نغطيها جزئيا بقوس بيضوي . لاحظ إختلاف بين الأسلوب **rect()** في السطر 26, الأسلوب **ellipse()** يحتاج إلى 4 برامترات و التي هي إحداثيات **x** و **y** للزوايا اليسرى السفلية و اليمنى العلوية للمستطيل الذي سيتم وضع به القوس البيضوي, وللأسلوب **rect()**, هذه 4 برامترات هي أحداثيات **x** و **y** للزاوية اليسرى للمستطيل, ثم عرضه و إرتفاعه . و الأسطر من 38 إلى 40 ترسم دائرة صغيرة في وسط الصفحة, و التي ستكون بمثابة مؤشر لتحديد مواقع لفهم بعض الأسطر من النص التي ترسم بتعليماتنا الأخيرة .

* الأسطر من 42 إلى 61 : يرجى النظر إلى هذه الخطوط . سوف نظهر لكم كيف يمكنكم محاذات إلى اليسار, و إلى اليمين و في وسط خط النص, بإستخدام أساليب **drawString()** و **()drawRightString** و **drawCentredString()** . يمكنك بالطبع إستخدام خطوط و ألوان و أحجام مختلفة للأحرف, و حتى تدور النصك في أي زاوية !

96يمكن لهذا "القناع" أن يحتوي على قيم أخرى, لكن من الواضح أن هذا لا يمكن وضعه في مقدمة عن Reportlab المحدودة جدا, نحن لا يمكننا أن نسمح بتفاصل أكثر لكل خيار أو برامتر مقدم .

مستندات متعدد الصفحات و إدارة الفقرات

Gestion des paragraphes avec ReportLab

La **programmation** est l'art d'apprendre à une machine comment accomplir de nouvelles tâches, qu'elle n'avait jamais été capable d'effectuer auparavant.

C'est par la programmation que vous pourrez acquérir le plus de contrôle, non seulement sur votre machine, mais aussi peut-être sur celles des autres par l'intermédiaire des réseaux. D'une certaine façon, cette activité peut donc être assimilée à une forme particulière de magie.

Elle donne effectivement à celui qui l'exerce un certain pouvoir, mystérieux pour le plus grand nombre, voire inquiétant quand on se rend compte qu'il peut être utilisé à des fins malhonnêtes.

Dans le monde de la programmation, on désigne par le terme ***hacker*** les programmeurs chevronnés qui ont perfectionné les systèmes d'exploitation de type Unix et mis au point les techniques de communication qui sont à la base du développement extraordinaire de l'Internet.

Ce sont eux également qui continuent inlassablement à produire et à améliorer les logiciels libres (*Open Source*).

Selon notre analogie, les hackers sont donc des maîtres-sorciers, qui pratiquent la magie blanche.

Mais il existe aussi un autre groupe de gens que les journalistes mal informés désignent erronément sous le nom de *hackers*, alors qu'ils devraient plutôt les appeler *crackers*.

Ces personnes se prétendent *hackers* parce qu'ils veulent faire croire qu'ils sont très compétents, alors qu'en général ils ne le sont guère.

Ils sont cependant très nuisibles, parce qu'ils utilisent leurs quelques connaissances pour rechercher les moindres failles des systèmes informatiques construits par d'autres, afin d'y effectuer toutes sortes d'opérations illicites : vol d'informations confidentielles, escroquerie, diffusion de spam, de virus, de propagande haineuse, de pornographie et de contrefaçons, destruction de sites web, etc.

Ces sorciers dépravés s'adonnent bien sûr à une forme grave de magie noire.

Mais il y en a une autre.

Les vrais *hackers* cherchent à promouvoir dans leur domaine une certaine éthique, basée principalement sur l'émulation et le partage des connaissances. La plupart d'entre eux sont des perfectionnistes, qui veillent non seulement à ce que leurs constructions logiques soient efficaces, mais aussi à ce qu'elles soient élégantes, avec une structure parfaitement lisible et documentée.

Vous découvrirez rapidement qu'il est aisé de produire à la va-vite des programmes qui fonctionnent, certes, mais qui sont obscurs et confus, indéchiffrables pour toute autre personne que leur auteur (et encore !).

Cette forme de programmation absconse et ingérable est souvent aussi qualifiée de « magie noire » par les *hackers*.

La démarche du programmeur

Comme le sorcier, le programmeur compétent semble doté d'un pouvoir étrange qui lui permet de transformer une machine en une autre, une machine à calculer en une machine à écrire ou à dessiner, par exemple, un peu à la manière d'un sorcier qui transformerait un prince charmant en grenouille, à l'aide de quelques incantations mystérieuses entrées au clavier.

Comme le sorcier, il est capable de guérir une application apparemment malade, ou de jeter des sorts à d'autres, via l'Internet.

Mais comment cela est-il possible ?

في هذا الفصل المقدم لوظائف ReportLab, لا يمكننا لمس إلا موضوع واسع جدا . الذي قمنا بشرحه بإيجاز في الصفحات السابقة للطبقة الأدنى من هذه المكتبات, مستوى "اللوحة" . فوق الطبقة الأولى, يوجد أربعة طيقات أخرى :

* العناصر " ***fluables***[97] " : و التي هي أهم الفقرات (أجزاء من النص المنسق), لكن يمكن أن تكون صور و مسافات و جداول ...إلخ, و التي تشترك في أنها يمكن أن تكون "الإلقاء" بعضها واحد تلو الأخر في مناطق محددة مسبقا من المستند .

* الإطارات, التي تحدد المناطق المستطيلة على الصفحة, حيث يمكنك "تدفق" العناصر المختلفة fluables التي تم شرحها فوق, في تدفق مستمر .

* أنماط الصفحة, التي هي نماذج مختلفة من الصفحات مع رؤوس و تذييلات, و إطارات و ترقيم .

* أنماط المستندات (أو النماذج), التي تقدم مخططات مختلفة محددة مسبقا .

عند قراءة ما سبق, عليك أن تعرف أن تصرفك مع ReportLab هي أداة محترفة لمستوى عال, الذي يقدم لك كل ما هو متوقع من نظام معالج نصوص حديث . بدلا من ذلك نحن نفتقر إلى الوصف كاملا, و لكن سنحاول شرح آليات تنفيذها على المستويات الرئيسية 2 و 3, و fluables و الإطاران .

> يجب على تفسيراتنا أن تكون كافية لإعداد مستنداتك الأول, و لكننا نشجع و بقوة أن تقرأ الوثائق واسعة النطاق على شبكة الإنترنت على موقع الرسمي ل *Reportlab*, من أجل تحقيق أقصى فائدة .

مثال عن سكريبت تخطيط ملف نصي

السكريبت التالي تحمل أسطر لملف نصي بسيط في قائمة . كل سطر يتم تحويله إي كائن fluable من نوع "فقرة" . يتم إنشاء fluable أخرى و يتم إدراجها بين الفقرات : تباعد بين كل واحدة منهم, و من وقت لأخر تنسيق الصورة . يتم وضع كل هذه الكائنات في قائمة واحدة, و التي تستخدم لعد

[97] Fluable : « qui coule, qui est liquide » (du latin *fluere*). Adjectif très peu utilisé dans le langage courant, utilisé ici comme traduction approximative du néologisme anglais *flowable* formé à partir du verbe *to flow* (s'écouler, fluer) et qui signifie ici « élément d'un flux d'entités imprimables ».

1لك في تدفق مصادر الطاقة لملئ 5 إطارات (frames) . يتم التعامل مع الفائض المتاح في نهاية مختلفة لتسليط الضوء على بعض الأساليب الأساسية جدا لكائنات-الفقرات .

```
1#    #=== توليد وثيقة PDF مع معالج fluables (الفقرات) ===
2#
3#    #: تكييف السكريبت لجعله يعمل مع البايثون 2.6 أو 2.7
4#    #(للبايثون 3 Reportlab يمكنك حذف هذه الأسطر إذا تم توفير)
5#    from __future__ import unicode_literals
6#    from __future__ import division          #تقسيم "حقيقي"
7#    from codecs import open                  #ترميز ملفات النصية
8#    #-----------------------------------------------------------------
9#
10#   #إستدعاء بعض عناصر مكتبة Reportlab :
11#   from reportlab.pdfgen.canvas import Canvas
12#   from reportlab.lib.units import cm
13#   from reportlab.lib.pagesizes import A4
14#   from reportlab.platypus import Paragraph, Frame, Spacer
15#   from reportlab.platypus.flowables import Image as rlImage
16#   from reportlab.lib.styles import getSampleStyleSheet
17#
18#   #إنشاء قائمة لسلاسل نصية لتحويلها + + loin en paragraphes :
19#   ofi =open("document.txt", "r", encoding="Utf8")
20#   txtList =[]
21#   while 1:
22#      ligne =ofi.readline()
23#      if not ligne:
24#         break
25#      txtList.append(ligne)
26#   ofi.close()
27#
28#   #=== صنع وثيقة PDF :
29#   fichier ="document_3.pdf"
30#   can = Canvas("{0}".format(fichier), pagesize=A4)
31#   styles = getSampleStyleSheet()          #قاموس أنماط معرفة مسبقا
32#   styleN =styles["Normal"]                #كائن صنف ParagraphStyle()
33#
34#   #. الفقرات و الأرقام و المباعدات تسمى عناصر "fluables"
35#   #إضافة هذه عناصر fluables في قائمة <story> ("تاريخ") :
36#   n, f, story = 0, 0, []
37#   for txt in txtList:
38#      story.append(Paragraph(txt, styleN))    #إضافة فقرة
39#      n +=1                                   #عد الفقرات المولدة
40#      story.append(Spacer(1, .2*cm))          #إضافة مساحة (تباعدة) (2مم)
41#      f +=2                                   #عد ال fluables المولدة
42#      if n in (3,5,10,18,27,31):              #إضافة صورة نقطية
43#         story.append(rlImage("cocci3.gif", 3*cm, 3*cm, kind="proportional"))
44#         f +=1
45#
46#   #=== إعداد الصفحة الأولى :
47#   can.setFont("Times-Bold", 18)
48#   can.drawString(5*cm, 28*cm, "Gestion des paragraphes avec ReportLab")
49#   #وضع ثلاثة إطارات (2 "أعمدة" و واحد "أسفل الصفحة") :
50#   cG =Frame(1*cm, 11*cm, 9*cm, 16*cm, showBoundary =1)
51#   cD =Frame(11*cm, 11*cm, 9*cm, 16*cm, showBoundary =1)
52#   cI =Frame(1*cm, 3*cm, 19*cm, 7*cm,  showBoundary =1)
53#   #وضع عناصر ال fluables في هذه الأطر الثلاثة
54#   cG.addFromList(story, can)              #ملئ الإطار على اليسار
```

```
55#     cD.addFromList(story, can)              #ملئ الإطار على اليمين
56#     cI.addFromList(story, can)              #ملئ الإطار السفلي
57#
58#     can.showPage()                          #الإنتقال إلى الصفحة التالية
59#
60#     #=== إعداد الصفحة الثانية :
61#     cG =Frame(1*cm, 12*cm, 9*cm, 15*cm, showBoundary =1)      #إطارات
62#     cD =Frame(11*cm, 12*cm, 9*cm, 15*cm, showBoundary =1)     #(= 2 عمودان)
63#     cG.addFromList(story, can)              #ملئ الإطار على اليسار
64#     cD.addFromList(story, can)              #ملئ الإطار على اليمين
65#
66#     #: المتبقية fluables معالجة فردية لعناصر ال
67#     xPos, yPos = 6*cm, 11.5*cm              #موقع البدء
68#     lDisp, hDisp = 14*cm, 14*cm             #عرض و الإرتفاع المتاح
69#     for flua in story:
70#        f += 1
71#        l, h =flua.wrap(lDisp, hDisp)        #عرض و الإرتفاع الفعلي
72#        if flua.identity()[1:10] =="Paragraph":
73#           can.drawString(2*cm, yPos-12, "Fluable n° {0}".format(f))
74#        flua.drawOn(can, xPos, yPos-h)       #تثبيت fluable
75#        yPos -=h                             #موقع التالي
76#
77#     can.save()                              #إنهاء الوثيقة
```

تعليقات

* السطر 14 : يوفر ReportLab صنف الإطارات (frames), العديد من أصنف كيانات fluables يمكنهم "يلقو" في تدفق مستمر . نحن لا نستخدم هنا إلا fluables من نوع فقرة و مسافات و صورة . بالمناسبة إستخدام **as** لإعادة تسمية الصنف **Image()** المستدعي : هذا الوقائي مفيد لتفادي الخلط مع صنف **Image()** من مكتبة تصوير البايثون (PIL) التي يمكن أن تستخدم أيضا ربما في السكريبت أيضا .

* السطر 16 : الصنف **Paragraph()** متطور جدا : مستندات ReportLab يخصص فصلا كاملا . كائنات المثيل من هذا الصنف تسمح بتخطيط دقيق لك جزء من أجزاء أي النص, ينسق وفقا لرغباتك في نمط معين . بعض الأنكاط الأساسية متوفرة في وحدة **reportlab.lib.styles**, سمة **getSampleStyleSheet** يتم تنظيمها مثل القاموس, و لكن الشيئ المهم هو أنه يمكنك بسهولة تعديل أي منها لتناسب إحتياجاتك عن طريق تعديل خصائصه المختلفة الإفتراضية (الخط و اللون, قم السحب أو/و المسافات, تبويتات, إضافة تعداد بوصي أو رقمي, المحاذات, إلخ) . و يقترح مثال على تعديل نمط الفقرة في وقت لاحق, في التمرينات في نهاية هذا الفصل .

* الأسطر من 18 إلى 26 : مصدر فقراتنا هو ملف نص بسيط, نحن سنقوم بإستخراج سلاسل النصية بالطريقة العادية (أنظر صفحة 133) . يرجى ملاحظة أن هذا النص يحتوي على عدد من العلامات

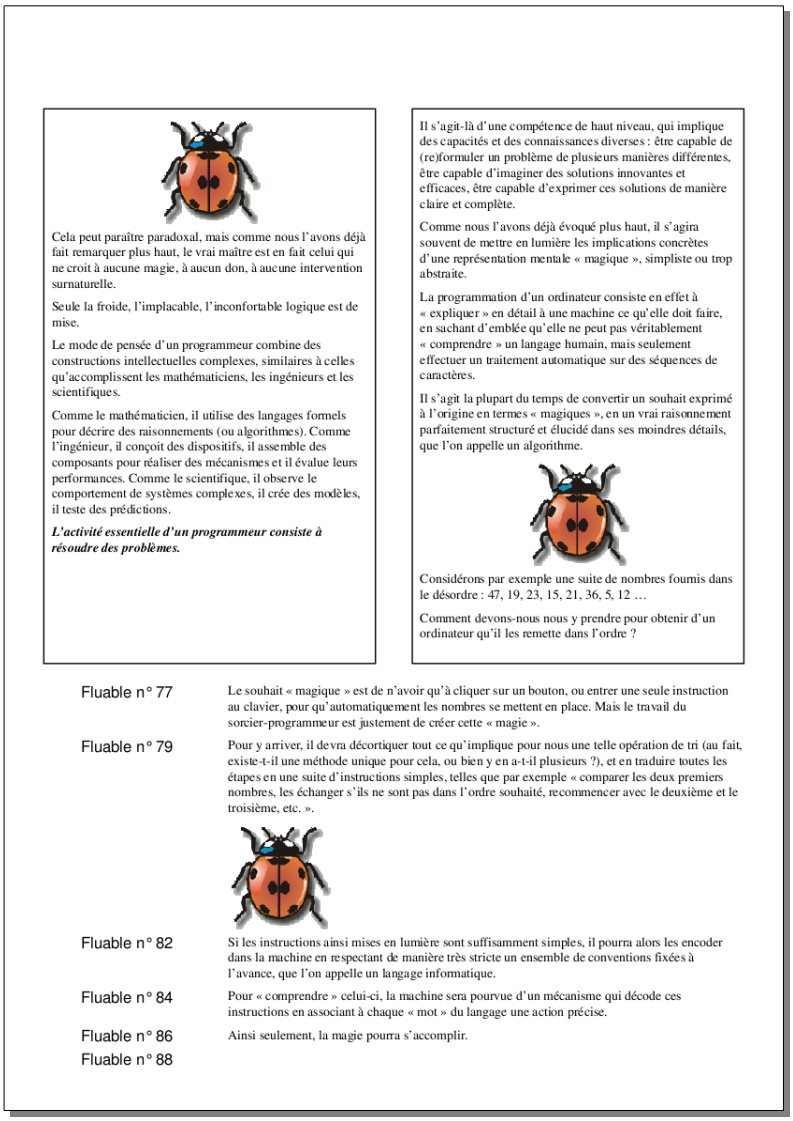
التنسيق على الإنترنات من نوع XML, مثل على سبيل المثال <u> و </u> لجعل جزء من النص تحته سطر, أو و لجعله مائلا, إلخ) . نحن لا يمكننا أن نقدم هنا قائمة جميع العلامات . لاحظ أنه لا يمكنك إستخدام رموز محجوزة مثل < و & في تص الذي سيتحول إلى فقرة ReportLab .

* السطر 32 . هذه التعليمة تضع في المتغير **styleN** كائن من صنف **ParagraphStyle()**, الذي يقوم بتعريف النمط الذي تم إختياره لكافة الفقرات, و في هذه الحالة نمط النص الحالي بخط Times-Roman . كما سبق ذكره أعلاه, و يمكننا بسهولة تغيير نمط بتغيير القيمة الإفتراضية من سمات المختلفة (أنظر التمارين في نهاية الفصل) .

* الأسطر من 34 إلى 44 : هذا هو مكان الذي سنبني قائمة كائنات fluables التي سوق تتلقى المزيد, من التدفق المستمر, في مختلف الأطر التي أعددناها . و سوف تكون هذه القائمة نوع ما "قصة" و سوف نقوم بوضع أجزاء نصوص, و صور, إلخ, في أماكن مختلفة مخصصة لصفحاتنا . و هذا ما يفسر إختيار إسم المتغير **story** عادة للإشارة إلى هذه القائمة . لكل سطر مستخرج من الملف النصي, سوف نصيف إليه كائن fluable من نوع فقرة, تم تمثيله في السطر 38 مع نمط معرفة في **styleN**, تليها مباشرة fluable من نوع مسافات, يتم إنشاء المثيل في السطر 40 . بعد بضعة فقرات, سوف نستدعي أيضا بعض fluables من نوع صورة (السطور 42 و 43) . هذه الصور يمكن إعادة تحجيمها حسب رغبتك (البرامتر الإختياري **kind="proportional"** لحفظ قوة نسبة جانب(و هذا يعني نيبة الطول/العرض, و الأبعاد التي تسبقها هي الحد الأقصى)) . و بالمناسبة, نحن قمنا بعد الفقرات المتوازية و fluables في المتغيرات **n** و **f**, لكن من الواضح أنه ليس ضروريا .

* الأسطر من 49 إلى 52 : في الصفحة الأولى من وثيقتنا, قمنا بتمثيل 3 أطر (كائنات من صنف **()Frame** ل ReportLab) : عمودان عموديان فوق مستطيل أفقي . هذه الأطر هي من المساحات "التي تتقضي" fluablesتنا . لكل إطار, يجب توفير بالترتيب : إحداثيات الزاوية اليسرى السفلية للمستطيل (يبدأ الحساب من أسفل الصفحة), و عرضه و إرتفاعه . و ليبقى الكود قابلا للقراءة, قمنا بالإشارة إلى الأبعاد بالسنتمتر, و قمنا بتحويلها إلى نقاط بيكا مع إستخدام الثابت **cm** . البرامتر الإختياري **showBoundary=1** يسمح بتصور الإطارات المستطيلة بشكل فعال خلال مراحل

تطوير السكريبتاتك . عند الإنتهاء منها, يمكنك ببساطة حذف هذه الحجة(أو إعطائها قيمة 0) لجعلها تختفي .



Cela peut paraître paradoxal, mais comme nous l'avons déjà fait remarquer plus haut, le vrai maître est en fait celui qui ne croit à aucune magie, à aucun don, à aucune intervention surnaturelle.

Seule la froide, l'implacable, l'inconfortable logique est de mise.

Le mode de pensée d'un programmeur combine des constructions intellectuelles complexes, similaires à celles qu'accomplissent les mathématiciens, les ingénieurs et les scientifiques.

Comme le mathématicien, il utilise des langages formels pour décrire des raisonnements (ou algorithmes). Comme l'ingénieur, il conçoit des dispositifs, il assemble des composants pour réaliser des mécanismes et il évalue leurs performances. Comme le scientifique, il observe le comportement de systèmes complexes, il crée des modèles, il teste des prédictions.

L'activité essentielle d'un programmeur consiste à résoudre des problèmes.

Il s'agit-là d'une compétence de haut niveau, qui implique des capacités et des connaissances diverses : être capable de (re)formuler un problème de plusieurs manières différentes, être capable d'imaginer des solutions innovantes et efficaces, être capable d'exprimer ces solutions de manière claire et complète.

Comme nous l'avons déjà évoqué plus haut, il s'agira souvent de mettre en lumière les implications concrètes d'une représentation mentale « magique », simpliste ou trop abstraite.

La programmation d'un ordinateur consiste en effet à « expliquer » en détail à une machine ce qu'elle doit faire, en sachant d'emblée qu'elle ne peut pas véritablement « comprendre » un langage humain, mais seulement effectuer un traitement automatique sur des séquences de caractères.

Il s'agit la plupart du temps de convertir un souhait exprimé à l'origine en termes « magiques », en un vrai raisonnement parfaitement structuré et élucidé dans ses moindres détails, que l'on appelle un algorithme.

Considérons par exemple une suite de nombres fournis dans le désordre : 47, 19, 23, 15, 21, 36, 5, 12 …

Comment devons-nous nous y prendre pour obtenir d'un ordinateur qu'il les remette dans l'ordre ?

Fluable n° 77 — Le souhait « magique » est de n'avoir qu'à cliquer sur un bouton, ou entrer une seule instruction au clavier, pour qu'automatiquement les nombres se mettent en place. Mais le travail du sorcier-programmeur est justement de créer cette « magie ».

Fluable n° 79 — Pour y arriver, il devra décortiquer tout ce qu'implique pour nous une telle opération de tri (au fait, existe-t-il une méthode unique pour cela, ou bien y en a-t-il plusieurs ?), et en traduire toutes les étapes en une suite d'instructions simples, telles que par exemple « comparer les deux premiers nombres, les échanger s'ils ne sont pas dans l'ordre souhaité, recommencer avec le deuxième et le troisième, etc. ».

Fluable n° 82 — Si les instructions ainsi mises en lumière sont suffisamment simples, il pourra alors les encoder dans la machine en respectant de manière très stricte un ensemble de conventions fixées à l'avance, que l'on appelle un langage informatique.

Fluable n° 84 — Pour « comprendre » celui-ci, la machine sera pourvue d'un mécanisme qui décode ces instructions en associant à chaque « mot » du langage une action précise.

Fluable n° 86 — Ainsi seulement, la magie pourra s'accomplir.

Fluable n° 88

* الأسطر من 54 إلى 56 : الأسلوب **addFromList()** هي كائنات إطارات تم صنعها بالخطوة السابقة تقوم بالملئ, بداية من القائمة **story** . فهي تقوم بإستخراج ال fluables واحدا واحدا و تثبيتها واحد تحت الأخر في الإطار, حتى يمتلئ, و من ثم يقفز تلقائيا إلى السطر الضروري

لتجنب إنقسام الكلمات, و يطبق جميع مؤشرات الأنماط التي تم إختيارها . عندما يمتلئ الإطار, يمكن إستخدام القائمة المتبقية في **story** لملئ أطر أخرى, و هكذا .

> لاحظ أن إذا كانت الفقراة كبيرة جدا ليتم تثبيتها في الإطار, يتم صنع إستثناء . و الذي يمكن أن يتسبب في الكشف عن تقسيم تلقائي لفقرة في فقرات أصغر . راجع وثائق *Reportlab* لمزيد من المعلومات .

* السطر 58 يقوم بإغلاق الصفحة الحالية و يقوم بإدراج صفحة جديدة في المستند(مستند) .

* السطور 60 إلى 64 : الصفحة المدرجة حديثا فارغة . لأنه يثبت أطر أخرة, و نملئه ب fluables التي نستمر بالإستخراجها من قائمة **story**, لكن في نهاية العملية لن تكون فارغة : سوف نقوم بخدمة ال fluables المتبقية لشرح أكثر الأساسية. و السماح بالتحكم أكثر لوضعهم في الصفحة .

* السطور 67 و 68 : سوف نختار نقطة بداية على الصفحة (العد يبدأ دائما من الأسفل !) و أبعاده الأقصى (العرض و الإرتفاع) التي قررنا تخصيصها لكل من fluablesتنا . في هذه الحالة, إخترنا مربع من 14×14 سم . هتين القيمتين, العرض الفقط الذي سيتم حده, لأن أي من fluables المتبقية هي كبيرة بما يكفي لإرتفاع 14 سم .

* الأسطر من 69 إلى 75 : سوف نقوم بتدوير قائمة fluables المتبقية . بدلا من "تدفق" في أطر صارمة معرفة سابقا كما فعلنا حتى الأن, و هذه المرة سوف نقوم بوضعها مباشرة على اللوحة, بتحديد المساحة المطلوبة لكل واحدة . الأسلوب **wrap()** للكائن fluable أثناء معالجتها لهذا الغرض . و يأخذ برامترات الأبعاد القصوى للمساحة التي ترغب في تخصيصها ل fluable (و هذا يعني بشكل عام, العرض الذي تريد رؤيته, و الإرتفاع الواضح أكثر أهمية), و تقوم بإرجاع العرض و الإرتفاع التي سيتم إستخدامها . في مثالنا إخترنا عرض ثابت, و إرتاع وجدنا أنه يمكننا وضع ال fluables تحت الأخريات (الأسلوب **drawOn()**, السطر 74) بالحفاظ على معرفة الإحداثيات الفعلية في أي وقت (و هذ ما يسمح لنا على سبيل المثال وضع ملصق بجوار كل فقرة (السطر 73)) . الأسلوب **identity()** يستخدم في السطر 72 و الذي يسمح بالمناسبة بتحديد أي نوع (فقرة, صورة, فراغات) ل fluable أثناء المعالجة .

في الختام

إعلم أن المذكور هنا هو لمحة صغيرة جدا من إمكانيات الموارد الضخمة التي تقدمها مكتبة ReportLab . من الواضح أننا تغاضينا على الكثر من المفاهيم و الأساليب . على سبيل المثال نحن لم نشرح ماذا نفعل لخفض فقرة كبيرة جدا أو إنشاء جداول أو رسوم بيانية و قوالب للمستندات, وصلات و ما إلى ذلك ... و نحن لم نشرح حتى كيفية تشفير ال PDF الخاصة بنا أو أن نضيف نماذج أو تأثيرات الإنتقال في الصفحة إلخ . مرة أخرى نحن نوصي بشدة أن وقم بالبحث على الإنترنات عن وثائق المتوفر لهذه المكتبة, فضلا عن مكتبة Python Imaging Library .

تمارين

18.1 قم بتعديل السكريبت السابق و ذلك بإزالة الإطارات المستطيلة التي تركناها ليعرض أغراض العرض التوضيحي, و عدل نمط الفقرات المستخدم بتعديل سمات الكائن المناسب, مع مثالا :

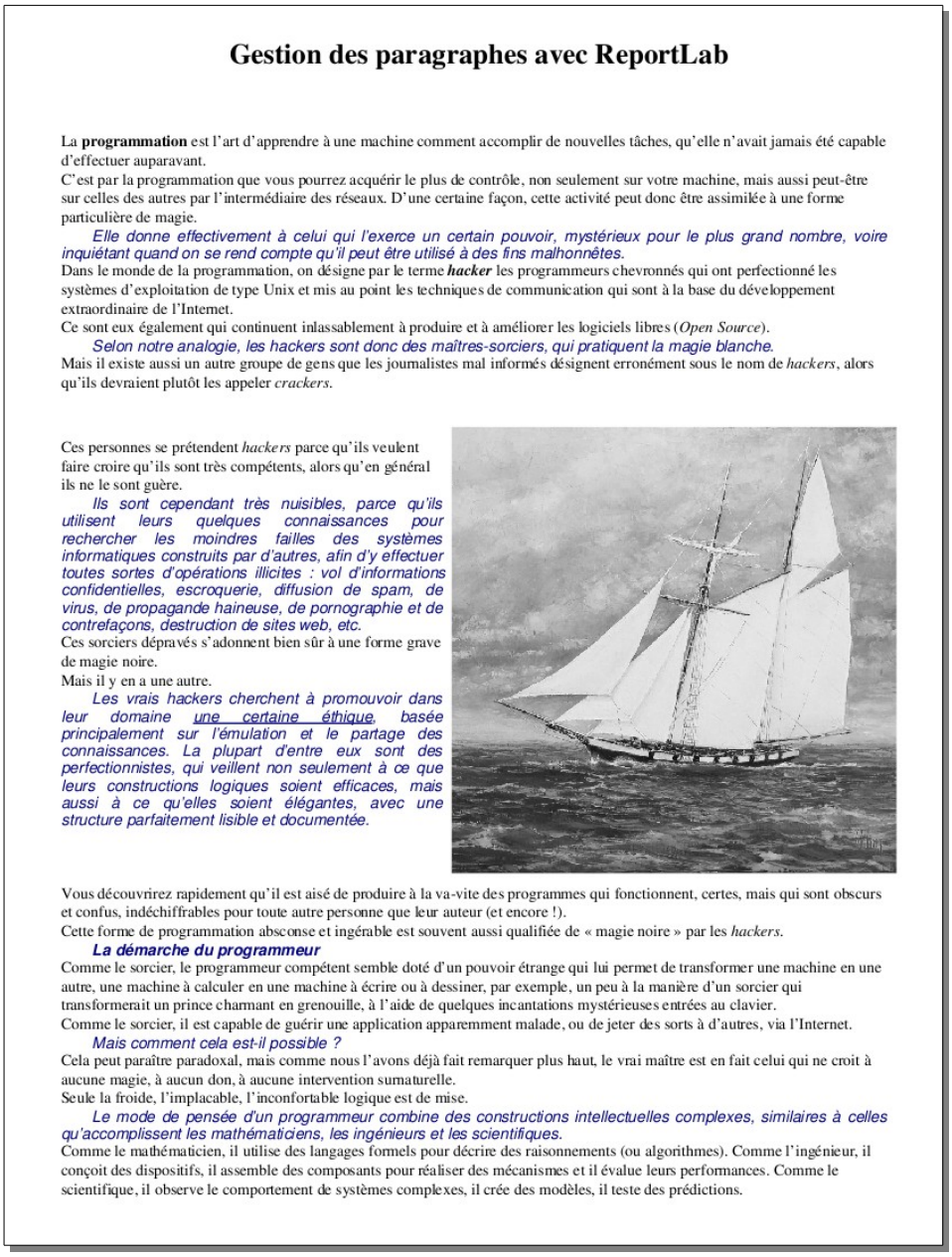
```
styleN.fontName ='Helvetica-oblique'
styleN.fontSize =10
styleN.leading =11                    # تباعد
styleN.alignment =TA_JUSTIFY          # أو TA_LEFT, TA_CENTER, TA_RIGHT
styleN.firstLineIndent =20            # بادئة للسطر الأول
styleN.textColor ='navy'
```

> لاحظ أن الثوابت ***TA_LEFT*** و***TA_RIGHT*** و ***TA_CENTER*** و ***TA_JUSTIFY*** يمكن إستدعائهم من وحدة ***reportlab.lib.enums***. و هي على التوالي 0, 2, 1 و 4.

18.2 عدل السكريبت السابق لتقوم بتغييرات في النمط, على سبيل المثال, تقسم فقرة على ثلاثة . لفعل هذا, أنت تعرف أنه لا يمكنك نقل نمط تعبين بسيط مباشرة في متغير أخر, وذلك بسبب الأسماء المستعارة (أنظر للصفحة 147, "نسخ قائمة") . لعمل نسخة "حقيقية" من كائن بايثون, و يمكنك إستخدام الدالة **()deepcopy**, بإستدعاء الوحدة **copy** :

```
from copy import deepcopy
styleB = deepcopy(styleN)
```

18.3 عدل السكريبت السابق لطلب صفحة تحتوي على نص "مغلف" صورة . يمكن للصورة أن تكون بأي حجم, لكن سوف ترمز دائما تلقائيا على وسط العمودي, و محاذات على الهامش الأيمن . حوله يتم تلقائيا تعريف ثلاثة أطر, ترتب تلقائيا الفقرات المختلفة التي أدرجت في هذه الأطر التي تجاوز



Gestion des paragraphes avec ReportLab

La **programmation** est l'art d'apprendre à une machine comment accomplir de nouvelles tâches, qu'elle n'avait jamais été capable d'effectuer auparavant.
C'est par la programmation que vous pourrez acquérir le plus de contrôle, non seulement sur votre machine, mais aussi peut-être sur celles des autres par l'intermédiaire des réseaux. D'une certaine façon, cette activité peut donc être assimilée à une forme particulière de magie.

Elle donne effectivement à celui qui l'exerce un certain pouvoir, mystérieux pour le plus grand nombre, voire inquiétant quand on se rend compte qu'il peut être utilisé à des fins malhonnêtes.

Dans le monde de la programmation, on désigne par le terme ***hacker*** les programmeurs chevronnés qui ont perfectionné les systèmes d'exploitation de type Unix et mis au point les techniques de communication qui sont à la base du développement extraordinaire de l'Internet.
Ce sont eux également qui continuent inlassablement à produire et à améliorer les logiciels libres (*Open Source*).

Selon notre analogie, les hackers sont donc des maîtres-sorciers, qui pratiquent la magie blanche.

Mais il existe aussi un autre groupe de gens que les journalistes mal informés désignent erronément sous le nom de *hackers*, alors qu'ils devraient plutôt les appeler *crackers*.

Ces personnes se prétendent *hackers* parce qu'ils veulent faire croire qu'ils sont très compétents, alors qu'en général ils ne le sont guère.

Ils sont cependant très nuisibles, parce qu'ils utilisent leurs quelques connaissances pour rechercher les moindres failles des systèmes informatiques construits par d'autres, afin d'y effectuer toutes sortes d'opérations illicites : vol d'informations confidentielles, escroquerie, diffusion de spam, de virus, de propagande haineuse, de pornographie et de contrefaçons, destruction de sites web, etc.

Ces sorciers dépravés s'adonnent bien sûr à une forme grave de magie noire.
Mais il y en a une autre.

Les vrais hackers cherchent à promouvoir dans leur domaine une certaine éthique, basée principalement sur l'émulation et le partage des connaissances. La plupart d'entre eux sont des perfectionnistes, qui veillent non seulement à ce que leurs constructions logiques soient efficaces, mais aussi à ce qu'elles soient élégantes, avec une structure parfaitement lisible et documentée.

Vous découvrirez rapidement qu'il est aisé de produire à la va-vite des programmes qui fonctionnent, certes, mais qui sont obscurs et confus, indéchiffrables pour toute autre personne que leur auteur (et encore !).
Cette forme de programmation absconse et ingérable est souvent aussi qualifiée de « magie noire » par les *hackers*.

La démarche du programmeur

Comme le sorcier, le programmeur compétent semble doté d'un pouvoir étrange qui lui permet de transformer une machine en une autre, une machine à calculer en une machine à écrire ou à dessiner, par exemple, un peu à la manière d'un sorcier qui transformerait un prince charmant en grenouille, à l'aide de quelques incantations mystérieuses entrées au clavier.
Comme le sorcier, il est capable de guérir une application apparemment malade, ou de jeter des sorts à d'autres, via l'Internet.

Mais comment cela est-il possible ?

Cela peut paraître paradoxal, mais comme nous l'avons déjà fait remarquer plus haut, le vrai maître est en fait celui qui ne croit à aucune magie, à aucun don, à aucune intervention surnaturelle.
Seule la froide, l'implacable, l'inconfortable logique est de mise.

Le mode de pensée d'un programmeur combine des constructions intellectuelles complexes, similaires à celles qu'accomplissent les mathématiciens, les ingénieurs et les scientifiques.

Comme le mathématicien, il utilise des langages formels pour décrire des raisonnements (ou algorithmes). Comme l'ingénieur, il conçoit des dispositifs, il assemble des composants pour réaliser des mécanismes et il évalue leurs performances. Comme le scientifique, il observe le comportement de systèmes complexes, il crée des modèles, il teste des prédictions.

تلقائيا الصورة .

18.4 في سكريبت **spectacles.py** في الفصل السابق, أضف ما يلزم لجعل موضوع الإدارة تكون قائمة الحجوزات التي تقترحها الزائر بالحصول عليها على شكل وثيقة PDF :

19

الإتصال عبر الشبكة و خاصية التعدد (multithreading)

أثبت التطور الغير عادي للإنترنات بوضوح أن أجهزة الحاسوب يمكن أن تكون أدوات إتصال فعالة للغاية . في هذا الفصل, سوف نستكشف أساسيات هذه التكنلوجيا, عن طريق إجراء بعض التجارب مع أساليب بسيطة لربط الإتصال بين البرامج, لتسليط الضوء على لربط الترابط المعلومات بين العديد من الشركاء .

في ما يلي, نحن نفترض أنك تتعاون مع أشخاص أخرين, و محطة عملك البايقون متصل لشبكة محلية بإستخدام الإتصالات البروتوكول TCP/IP . إن نظام التشغيل غير مهم : على سبيل المثال, يمكنك تثبيت سكريبتات البايثون التي تم وصفها أدناه على محطة عمل على نظام تشغيل لينكس, و التواصل مع سكريبت أخر يتم تنفيذه على محطة عمل بنظام تشغيل مختلف, مثل ماك أو ويندوز .

يمكنك أيضا محاولة ما يلي على جهاز واحد, بوضع السكريبتات المختلفة في نوافذ منفصلة .

ال sockets

التمرين الأول الذي سوف يتم تقدمه هو إقامة إتصال بين جهازين فقط . يمكن لكل واحدة منهم تبادل الرسائل بالدور, لكن سوف تجد أن التكوينات ليست متماثلة . سيقوم السكريبت المثبت على واحد من هذه الأجهزة بدور برنامج الخادم, في حين أن الأخر سيكون بمثابة برنامج العميل .

برنامج الخادم يعمل بشكل مستمر على جهاز هويته معرفة جيدا على الشبكة عن طريق عنوان IP معين [98].و هو ينتظر وصول طلبات المرسلة من قبل العملاء المحتملين لهذا العنوان, من خلال خاصية منفذ الإتصالات المحدد . للقيام بذلك, يجب على السكريبت أن ينفذ كائن برمجي مرتبط بهذا المنفذ و الذي يسمى socket .

و من خلال جهاز آخر, يحاول برنامج العميل الإتصال عن طريق طلب مناسب . هذا الطلب هو رسالة التي ترسل للشبكة, و تماما كما تكلف الرسالة إلى مكتب البريد . يمكن للشبكة توجيه الطلب إلى أي آلة أخرى . و لكن بشرط : ليمكن الوصول إلى الجهة المقصودة, الطلب يجب أن يحتوي في رأيه إشارة إلى عنوان IP و منفذ الإتصال للمتلقين .

عندما يتم إنشاء إتصال مع الخادم, يعين العميل بنفسه أحد منافذه للإتصالات الخاصة . و من هذه اللحظة, يمكننا أن نعتبر هذه قناة متميزة تربط بين جهازين, كما لو كانت متصلت مع بعضها البعض عن طريق الأسلاك(المنفذي الإتصال يعملان دور طرفي السلك) . و يمكن بدأ تبادل المعلومات .
لإستخدام منافذ إتصال الشبكة, تقوم البرامج بإستدعاء مجموعة من إجراءات و دالات نظام التشغيل, من خلال كائنات الواجهة تسمى sockets . و هذه يمكن أن تنفذ تقنيتين الإتصالات مختلفتين و متكاملتين : و هي الحزم (و تسمى أيضا حزم البيانات), تستخدم على نظاق واسع على شبكة الإنترنات, ويستمر الإتصال, أو تدفق socket, و الذي هو أبسط قليلا .

بناء خادم بدائي

لتجاربنا الأولى, سوف نستخدم تقنية تدفق ال sockets .

هذا في الواقع مناسب جدا عندما يتعلق الأمر بالتواصل بين أجهزة الحاسوب المتصبة عبر شبكة محلية . و هذا الأسلوب تنفيذه سهل للغاية, و إنه يسمح بإنتاجية عالية لتبادل البيانات .

و أما عن التقنية الأخر (الحزم) من الأفضل أن تستخدم للإتصالات المرسلة عبر شبكة الإنترنات, وذلك بسبب موثوقيتها العالية(الحزم نفسها يمكن أن تصل إلى وجهتها من خلال مسارات مختلفة,

98و يمكن لجهاز خاص أن يقوم بتعيينه بأكثر وضوح, و لكن شرط أنه تم وضع الية على شبكة (DNS) لترجمة تلقائيا عنوان ال IP . يرجى الرجوع إلى كتاب عن الشبكات لمزيد من المعلومات .

أو أن تصدر أو تعيد إصدار نسخ متعددة إذا كان ذلك ضروريا لتصحيح أخطاء الإرسال), لكن تنفيذها هو قليلا أكثر تعقيدا . ونحن لن ندرسها في هذا الكتاب .

السكريبت الأول أدناه هو خادم قادر على التواصل مع عميل واحد . سوف نرى لاحقا ماذا يجب علينا أن نضيف حتى نتمكن من دعم الإتصالات المتوازية من عدة عملاء .

```
1#    #تعريف خادم ويب بدائي
2#    #هذا الخادم ينتظر إتصال عميل
3#
4#    import socket, sys
5#
6#    HOST = '192.168.1.168'
7#    PORT = 50000
8#    counter =0        #عداد الإتصالات النشطة
9#
10#   #1) صنع socket :
11#   mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
12#
13#   #2) ربط socket إلى عنوان محدد :
14#   try:
15#     mySocket.bind((HOST, PORT))
16#   except socket.error:
17#     print("La liaison du socket à l'adresse choisie a échoué.")
18#     sys.exit
19#
20#   while 1:
21#     #3) إنتظار إستعلام إتصال لعميل :
22#     print("Serveur prêt, en attente de requêtes ...")
23#     mySocket.listen(2)
24#
25#     #4) : إجراء الإتصال
26#     connexion, adresse = mySocket.accept()
27#     counter +=1
28#     print("Client connecté, adresse IP %s, port %s" % (adresse[0], adresse[1]))
29#
30#     #5) التحاور مع العميل
31#     msgServeur ="Vous êtes connecté au serveur Marcel. Envoyez vos
     messages."
32#     connexion.send(msgServeur.encode("Utf8"))
33#     msgClient = connexion.recv(1024).decode("Utf8")
34#     while 1:
35#       print("C>", msgClient)
36#       if msgClient.upper() == "FIN" or msgClient =="":
37#         break
38#       msgServeur = input("S> ")
39#       connexion.send(msgServeur.encode("Utf8"))
40#       msgClient = connexion.recv(1024).decode("Utf8")
41#
42#     #6) : إغلاق الإتصال
43#     connexion.send("fin".encode("Utf8"))
44#     print("Connexion interrompue.")
45#     connexion.close()
46#
47#     ch = input("<R>ecommencer <T>erminer ? ")
48#     if ch.upper() =='T':
49#       break
```

تعليقات

* السطر 4 : وحدة socket تحتوي على جميع دالات و أصناف اللازمة لبناء برنامج تواصل . كما سنرى في السطور التالية, و إنشاء الإتصال يشمل 6 خطوات .

* السطران 6 و 7 : هذان المتغيرات يعرفان هوية الخادم, لدمجه ل socket . يحتوي **HOST** على سلسلة نصية تشير إلى عنوان IP للخادم في شكل عشري المعتاد, أو إسم **DNS** لنفس الخادم (و لكن بشرط أنه تم تنفيذ آلية تحليل الإسم على الشبكة) . و يجب على المتغير **PORT** أن يحتوي على عدد صحيح, أي رقم منفذ الذي لا يستخدم لغرض آخر, و يفضل أن تعطي قيمة أكبر من 1024 .

* السطور من 10 إلى 11 : الخطوة الأولى لآلية الربط البيني . و لقد قمنا بتمثيل كائن من صنف **() socket**, و نحدد خيارين للإشارة إلى نوع العنوان الذي تم إختياره (نحن إستخدمنا عناوين من نوع "Internet") فضلا تكنلوجيا النقل (حزم بيانات أو إتصال مستمر (تدفق) : قررنا إستخدام هذا الأخير) .

* السطور من 13 إلى 18 : الخطوة الثانية . نحاول تأسيس إتصال بين socket و منفذ الإتصال . إذا لا يمكن إنشاء إتصال (على سبيل المثال, إذا كان منفذ الإتصال مشغول أو إسم الجهاز خاطئ), يتم إغلاق البرنامج مع رسالة خطأ . في السطر 15, لاحظ أن الأسلوب **bind()** ل socket ينتظر برامترا من نوع نفق, لذا يجب علينا أن نحصر متغيرتنا في زوج من الأقواس المزدوجة .

* السطر 20 : تم تصميم برنامج الخادم الخاص بنا ليعمل بإستمرار في إنتظار طلبات العملاء المحتملين, لقد وضعنا حلقة لانهائية .

* السطور من 21 إلى 23 : الخطوة الثالثة . سيتم ربط socket إلى منفذ الإتصال, و يمكن الأن أن يكون على إستعداد لتلقي الطلبات المرسلة من قبل العملاء . و هذا هو دور الأسلوب **listen()** . البرامتر الذي ينقل يشير إلى عدد الأقسى من الإتصالات التي يجب عليه قبولها في وقت لاحق . و سوف نرى لاحقا كيفية إدارتها .

* السطور من 25 إلى 28 : الخطوة الرابعة . عندما نقوم بإستدعاء الأسلوب **accept()**, ينتظر socket إلى مالانهاية أحد الطلبات . و يتم قطع السكريبت في هذه المرحلة, تماما مثل عندما نستدعي الدالة **input()** لإنتظار إدخال من لوحة المفاتيح . فإذا تم إستلام طلب, الأسلوب **accept()**, يقوم بإرجاع نفق من عنصرين : الأول هو مرجع كائن جديد للصنف **socket()**, و الذي سيكون واجهة الإتصال الحقيقي بين الخادم و العميل, و الثاني نفق أخر يحتوي على إحداثيات هذا العميل (عنوان ال IP الخاص بم و رقم المنفذ الذي يستخدمه) .

* السطور من 30 إلى 33 : الخطوة الخامسة . يتم تأسيس الإتصال الفعلي . الأساليب **send()** و **recv()** ل socket و التي تقوم بنقل و إستقبال الرسائل . و التي يجب أن تكون سلاسل بيتات . عند الإرسال, فمن الضروري أن يتم إدراج سلاسلا نصية من البيانات من نوع byte, و تقوم بالعكس عند الإستقبال .

> يقوم الأسلوب *send*() بإرجاع عدد البايتات المرسلة . و إن إستدعاء الأسلوب *recv*() يجب أن يحتوي على برامتر عدد صحيح يشير إلى العدد الأقصى من وحدات البايت التي يتم إرسالها دفعة واحدة . و البايتات الفائضة يتم وضعها في مخزن مؤقت, و يتم إرسالها عندما يتم إستدعاء الأسلوب *recv*() من جديد .

* السطور من 34 إلى 40 : هذه الحلقة الجديدة لانهائية حتى تحافظ على التواصل إذا قرر العميل إرسال كلمة "fin" أو سلسلة بسيطة فارغة . شاشات عرض للجهازين تقوم بعرض كل تطور في هذا الحوار .

* السطور من 42 إلى 45 : الخطوة السادسة . إغلاق الإتصال .

بناء عميل بدائي

السكريبت بالأسفل يعرف برنامج عميل متكامل مع الخادم الذي تم شرحه في الصفحات السابقة . لقد قمنا بتبسيطه .

```
1#      #تعريف عميل شبكة بدائي
2#      #هذا العميل يتحاور مع خادم مخصص
3#
4#      import socket, sys
5#
6#      HOST = '192.168.1.168'
7#      PORT = 50000
```

```
8#
9#      #1) صنع socket :
10#     mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11#
12#     #2) : إرسال إستعلام إتصال إلى الخادم
13#     try:
14#        mySocket.connect((HOST, PORT))
15#     except socket.error:
16#        print("La connexion a échoué.")
17#        sys.exit()
18#     print("Connexion établie avec le serveur.")
19#
20#     #3) : التحاور مع الخادم
21#     msgServeur = mySocket.recv(1024).decode("Utf8")
22#
23#     while 1:
24#        if msgServeur.upper() == "FIN" or msgServeur =="":
25#           break
26#        print("S>", msgServeur)
27#        msgClient = input("C> ")
28#        mySocket.send(msgClient.encode("Utf8"))
29#        msgServeur = mySocket.recv(1024).decode("Utf8")
30#
31#     #4) : إغلاق الإتصال
32#     print("Connexion interrompue.")
33#     mySocket.close()
```

تعليقات

* بداية هذا السكريبت تشيه الخادم . يجب على عنوان ال IP و المنفذ أن يكونا مطابقة للخادم .

* السطور من 12 إلى 18 : نحن لا نصنع هذه المرة كائن socket واحد, و الذي يستخدم الأسلوب **()connect** لإرسال طلب إتصال .

* السطور من 20 إلى 33 : بمجرد تأسيس الإتصال, يمكنك التواصل مع خادم يستخدم الأساليب **()send** و **()recv** التي سبق و تم شرحها أعلاه .

إدارة مهام متعددة في نفس الوقت بإستخدام المواضيع (theards)

نظام الإتصالات الذي قمنا بتطويره في الصفحات السابقة بدائي جدا : من ناحية فإنه يربط جهازين, و من ناحية أخرى يحد من حرية التعبير عن متحاورين . هذه في الحقيقة لا يمكنها إرسال رسائل بالدور . على سبيل المثال, عندما يرسل أحدهم رسالة, يقوم النظام بمنعه لأن شريكه الأخر لم يرسل رد . عندما يتعلق الأمر إلى تلقي الرد, يبقى النظام غير قادر على الإستقبال الأخر, لأنه لم يدخل بنفسه رسالة جديدة و هكذا ...

كل هذه المشاكل تنبع من حقيقة أن سكريبتاتنا معتادة على التعامل مع شيئ واحد فقط في كل مرة . على سبيل المثال, عندما يواجه تدفق التعليمات الدالة **input()** لا يحدث أي شيئ حتى يقوم المستخدم بإدخال البيانات المتوقعة , و حتى لو كان هذا يأخذ وقت طويلا جدا, فإن من العادة يكون من غير الممكت أن يقوم البرنامج بتنفيذ مهام أخرى في هذا الوقت . و مع ذلك, هذا صحيح فقط في نفس البرنامج الواحد : ربما كنت تعلم أنه يمكنك تشغل تطبيقات أخرى على جهاز الحاسوبك لأن أنظمة التشغيل الحديثة متعددة المهام .

الصفحات التالية سوف نشرحها بها كيف يمكنك تقديم ميزة تعدد المهام في برامجك, حتى تتمكن من تطوير تطبيقات شبكة الحقيقية, التي يمكنها الإتصال في وقت واحد مع عدة شركاء .

يرجى الأن النظر إلى السكريبت في الصفحة السابقة . و تتمثل ميزته الرئيسية في حلقة while في السطور من 23 إلى 29 . و مع ذلك, يتم مقاطعة هذه الحلقة في مكانين :

* في السطر 27, في إنتظار مدخلات من لوحة المفاتيح من المستخدم (الدالة **input()**) .

* في السطر 29, في إنتظار وصول رسالة الشبكة .

هذان المتوقعات متعاقبان, فإنه سيكون أكثر إثارة للإهتمام إذا كانا في الوقت نفسه . إذا كانت هذه هي الحالة, يمكن للمستخدم إرسال رسائل في أي وقت, دون الحاجة إلى إنتظار في كل مرة ردة فعل الشريك . و قد يظهر أي عدد من الرسائل, دون وجود إلزام للرد على كل واحدة منها لإستقابل الأخريات .

يمكننا تحقيق هذا إذا تعلمنا إدارة المتسلسلات المتعددة للتعليمات بالتوازي ضمن برنامج واحد . و لكن كيف يمكن أن يكون هذا ممكنا ؟

على مدى تاريخ الحاسوب, وضعت العديد من التقنيات لتقاسم وقت عمل المعالج بين المهام المختلفة, بحيث تبدو و كأنها صنعت في نفس الوقت (في حين أن المعالج سعطي لك واحد دوره) . يتم تطبيق هذه التقنية في نظام التشغيل, و أنه ليس ضروري أن نقوم بشرحها بالتفصيل هنا, على الرغم من أننا نستطيع الوصول إلى كل واحدة منهم عن طريق البايثون .

في الصفحات التالية, سوف تتعلم كيفية إستخدام واحدة من هذه التقنيات و التي هي سهلة و محمولة حقا فقط (و هي في الحقيقة تدعم أنظمة التشغيل الرئيسية) و نسمي هذه التقنية عمليات الخفيفة أو الخيوط (threads)[99].

في برنامج الحاسوب, المواضيع هي تيارات من التعليمات التي تنفذ بالتوازي (في وقت واحد تقريبا), في حين تتشارك في نفس مساحة الأسماء العامة .

في الحقيقة, إن أي تدفق عمليات لأي برنامج بايثون سوف يحتوي على الأقل على موضوع : الموضوع الرئيسي . و من هذا, يمكن لمواضيع الأطفال أن يبداؤ, التي سيتم تنفيذهم في نفس الوقت . كل موضوع طفل ينتهي و يختفي دون مزيد من اللغط عندما يحتوي يتم تنفيذ كافة تعليماته . و عندما ينتهي الموضوع الرئيسي, فمن الضروري في بعض الأحيان ضمان أن جميع مواضيعه الأطفال قد تم "قتلهم" معه .

عميل شبكة لإدارة الإرسال و الإستقبال المتزامن

سوف نطبق الأن تقنية المواضيع لبناء نظام دردشة[100] مبسط . هذا النظام يتكون من خادم واحد و أي عدد من العملاء . على عكس ما حدث في تمريننا الأول, لا يستخدم أحد خادم نفسه للإتصال, لكن عندما يتم تشغيلها, يمكن للعديد من العملاء الإتصال به و البدء في تبادل الرسائل .

كل عميل سيقوم بإرسال كافة الرسائل إلى الخادم, و لكن سوف يتم إحالة الرسائل على الفور على جميع العملاء الأخرين المتصلين, بحيث روئية كل واحد حركة المرور . يستطيع كل واحد منهم إرسال رسائل و يتلقاها الأخرين في أي وقف, في أي ترتيب, و التلقي و الإرسال سوف تدار في مواضيع منفصلة .

السكريبت النصي أدناه يقوم بتعريف برنامج عملي . و سيتم شرح برنامج الخادم لاحقا . و سوف تجد أن الجزء الرئيسي من البرنامج النصي (السطر 38 و ما يليه) مشابه للمثال السابق . فقط جزء

99في نظام تشغيل من نوع يونكس (مثل لينكس), الخيوط المختلفة لبرنامج نفسه هي جزء من عملية واحدة . و من الممكن أيضا إدارة العمليات المختلفة بإستخدام سكريبت بايثون(عملية متفرقة), و لكن تفسير هذه التقنية هي خارج نطاق هذا الكتاب .

100"الشات" (باللغة الفرنسية) هب عملية دردشة عبر أجهزة الحاسوب . و لقد تم إقتراح هذا المصطلح من قبل فرانكفونيين كنديين و يشير إلى "الدردشة عبر لوحة المقاتيح" .

"الحوار مع الخادم" سوف يتم إستبداله . بدلا من حلقة while, ستجد الأن تعليمات لصنع كائنين (في السطرين 49 و 50), لنبدأ مميزات هذين السطرين التاليين . هذين الكائنين موضعين تم صنعه عن طريق إشتقاق من صنف **Thread()** من وحدة threading . الذين يشغلونها بغض النظر عنن إستقبال و إرسال الرسائل . و يتم تغليف الموضوعين الطفلين في كائنات منفصلة, مما يسهل فهم الآلية .

```
1#    #تعرف عميل مدير شبكة يعمل بالتوازي في نقل و إستقبال الرسائل (بإستخدام خيطين) .
2#
3#    host = '192.168.1.168'
4#    port = 46000
5#
6#    import socket, sys, threading
7#
8#    class ThreadReception(threading.Thread):
9#      """objet thread gérant la réception des messages"""
10#     def __init__(self, conn):
11#       threading.Thread.__init__(self)
12#       self.connexion = conn        # مرجع socket الإتصال
13#
14#     def run(self):
15#       while 1:
16#         message_recu = self.connexion.recv(1024).decode("Utf8")
17#         print("*" + message_recu + "*")
18#         if not message_recu or message_recu.upper() =="FIN":
19#           break
20#       # خيط <réception> ينتهي هنا
21#       # نفرض إغلاق الخيط <émission> :
22#       th_E._stop()
23#       print("Client arrêté. Connexion interrompue.")
24#       self.connexion.close()
25#
26#   class ThreadEmission(threading.Thread):
27#     """objet thread gérant l'émission des messages"""
28#     def __init__(self, conn):
29#       threading.Thread.__init__(self)
30#       self.connexion = conn        # مرجع socket لبإتصال
31#
32#     def run(self):
33#       while 1:
34#         message_emis = input()
35#         self.connexion.send(message_emis.encode("Utf8"))
36#
37#   # البرنامج الرئيسي – تأسيس الإتصال :
38#   connexion = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
39#   try:
40#     connexion.connect((host, port))
41#   except socket.error:
42#     print("La connexion a échoué.")
43#     sys.exit()
44#   print("Connexion établie avec le serveur.")
45#
```

```
46#    #: الحوار مع الخادم : بتشغيل خيطين لإدارة مستقلة لعمليات إرسال و إستقبال الرسائل
47#    th_E = ThreadEmission(connexion)
48#    th_R = ThreadReception(connexion)
49#    th_E.start()
50#    th_R.start()
```

تعليقات

* ملاحظة هامة : في هذا المثال, قررنا إنشاء كائنين موضوعين مستقلين في الموضوع الرئيسي, لتسليط الضوء بوضوح على الآليات . يستخدم برنامجنا 3 مواضيع في كل شيئ, في حين أن القارئ اليقظ أن إثنين تكفي . في الواقع, إن الموضوع الرئيسي هو في نهاية المطاف مجرد مشغل 2 الأخريين ! لكن لا يوجد الحد الأدنى من عدد المواضيع . على العكس, منذ اللحظة التي قررت فيها إستخدام هذه التقنية, يجب أن تأخذ ميزة لتقسيم التطبيق إلى وحدات متميزة .

* السطر 7 : تحتوي وحدة threading على تعريف مجموعة متنوعة من الأصناف المثيرة للإهتمام لإدارة المواضيع . سوف نستخدم هنا الموضوع الصنف الوحيد **Thread()**, لكن في وقت لاحق سوف نستغل (الصنف **Lock()**), عندما يكون لدينا ما يدعو للقلق حول قضايا التزامن بين المواضيع المتزامنة المختلفة .

* الأسطر من 9 إلى 25 : الصنف المشتق من صنف **Thread()** يحتوي على أساسيات الأسلوب **run()** . وهو في هذا المكان الذي هو جزء من برنامج عمل خصيصا في الموضوع . غالبا ما سيكون حلقة متكررة, مثل هنا . يمكنك أن تنظر بشكل كامل محتويات هذا الأسلوب كسكريبت مستقل, الذي يعمل بالتوازي مع مكونات الأخرى من تطبيقك . عندما يتم تنفيذ هذا الكود تماما, يتم إغلاق الموضوع .

* الأسطر من 16 إلى 20 : تدير هذه الحلقة إستقبال الرسائل . في كل تكرار, تدفق التعليمات يتوقف عند السطر 17 في إنتظار رسالة جديدة, لكن لم يتم تجميد بقية البرنامج حتى الأن : مواضيع الأخرى تحتوي على عملهم بشكل المستقل .

* السطر 19 : تسبب الرسالة 'fin'(كبيرة أو صغيرة) أو رسالة فارغة(هذا صحيح لاسيما إذا تم قطع الإتصال من قبل شريك) بخروج حلقة التلقي . فيتم تنفيذ بعض تعليمات "التنظيف", و ينتهي الموضوع .

* السطر 23 : عندما يتم إنهاء تلقي الرسائل, نحن نأمل أن يتم إنهاء بقية البرنامج أيضا . لذلك نحن بحاجة إلى فرض إغلاق الكائنات المواضيع الأخرى, و نحن وضعناها في مكان مناسب لإدارة نقل الرسائل . و يمكن الحصول على هذا الإغلاق الإجباري بإستخدام الأسلوب **stop_()**[101].

* الأسطر من 27 إلى 36 : هذا الصنف يعرف كائن موضوع آخر, الذي يحتوي هذه المرة على حلقة تكرار أبدية . لذا لا يمكن أن تنتهي إلا إذا أجبرنا إغلاقها من الأسلوب الذي تم شرحه في الفقرة السابقة . في كل حلقة من هذا التكرار, تدفق التعليمات يتوقف في السطر 35 في إنتظار إدخال مدخلات من لوحة المفاتيح, و لكن هذا لا يمنع بأي حال من الأحوال المواضيع الأخرى من القيام بعلهم .

* السطور من 38 إلى 45 : هذه الأسطر تسرد مماثلة للسكريبتات السابقة .

* السطور من 47 إلى 52 : يتم هنا تمثيل و بدء كائني موضوعين أطفال . يرجى ملاحظة أنه من المستحسن أن تبدأ هذه النتيجة من خلال إستدعاء الأسلوب المدمج **start()**, بدلا من إستدعاء مباشرة الأسلوب **run()** الذي قمت بتعريفه بنفسك . لاحظ أيضا أنه يمكنك إستدعاء **start()** مرة واحدة فقط (إذا توقف مرة, لن تستطيع إعادة تشغيل كائن الموضوع) .

خادم مدير شبكة إستصالات المتعدد للعملاء في نفس الوقت

السكريبت النصي التالي يقوم بصنع خادم قادر على التعامل مع إتصالات لعدد من العملاء من نفس النوع الذي شرحناه في الصفحات السابقة .

لا سيستخدم هذا الخادم للإتصال بنفسه : فهم العملاء الذين سيتواصلون مع بعضهم البعض من خلال الخادم . فهي تلعب دور التتابع : فهو يقبل إتصالات العملاء, ثم ينتظر وصول رسائلهم . عندما تصل رسالة من عميل معين, سيقوم الخادم بإعادة توجيهها إلى العملاء الآخرين (يمكننا أن نضيف إليه سلسلة تحديد الإرسال إلى عميل محدد), بحيث يمكن للجميع رؤية كافة الرسائل, و معرفة من أين جائت .

```
1#    #. تعريف خادم شبكة يدير نظام دردشة مبسط
2#    #. إستخدام الخيوط لصنع إتصالات عميل بالتوازي
```

101أرجو أن يسامحون المبرمجون المتمرسوم : و أنا أعترف أن هذه الحيلة لفرض وقف للخيط ليست مستحسنة . و لكن وضعت هذا الإختصار لتجنب إثقال هذا النص, و هو مقدمة بدائية فقط . و يمكن للقارئ أن يستكشف ذه المسألة من خلال البحث في الكتب المرجعية المدرجة في قائمة المراجع .

```
3#
4#      HOST = '192.168.1.168'
5#      PORT = 46000
6#
7#      import socket, sys, threading
8#
9#      class ThreadClient(threading.Thread):
10#        '''dérivation d'un objet thread pour gérer la connexion avec un client'''
11#        def __init__(self, conn):
12#          threading.Thread.__init__(self)
13#          self.connexion = conn
14#
15#        def run(self):
16#          #التحاور مع العميل
17#          nom = self.getName()          #لكل خيط إسم
18#          while 1:
19#            msgClient = self.connexion.recv(1024).decode("Utf8")
20#            if not msgClient or msgClient.upper() =="FIN":
21#              break
22#            message = "%s> %s" % (nom, msgClient)
23#            print(message)
24#            #: إرسال رسالة إلى جميع العملاء
25#            for cle in conn_client:
26#              if cle != nom:        #لا يتم إعادة إرسالها إلى المرسل
27#                conn_client[cle].send(message.encode("Utf8"))
28#
29#          #: إغلاق الإتصال
30#          self.connexion.close()     #قطع الإتصال من جانب الخادم
31#          del conn_client[nom]       #حذف المدخلات في القاموس
32#          print("Client %s déconnecté." % nom)
33#          #ينتهي الخيط هنا
34#
35#      #تهيئة الخادم - وضع socket :
36#      mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
37#      try:
38#        mySocket.bind((HOST, PORT))
39#      except socket.error:
40#        print("La liaison du socket à l'adresse choisie a échoué.")
41#        sys.exit()
42#      print("Serveur prêt, en attente de requêtes ...")
43#      mySocket.listen(5)
44#
45#      #: إنتظار و دعم الإتصالات المطلوبة من العملاء
46#      conn_client = {}              #قاموس إتصالات العملاء
47#      while 1:
48#        connexion, adresse = mySocket.accept()
49#        #صنع كائن خيط جديد لتوليد الإتصال
50#        th = ThreadClient(connexion)
51#        th.start()
52#        #حفظ الإتصال في قاموس :
53#        it = th.getName()       #رقم الخيط
54#        conn_client[it] = connexion
55#        print("Client %s connecté, adresse IP %s, port %s." %\
56#            (it, adresse[0], adresse[1]))
57#        #: التحاور مع العميل
58#        msg ="Vous êtes connecté. Envoyez vos messages."
59#        connexion.send(msg.encode("Utf8"))
```

تعليقات

* السطور من 35 إلى 43 : تهيئة الخادم هي تعريف نفسه للخادم البدائي الذي تم شرحه في بداية الفصل السابق .

* السطر 46 : مراجع المختلفة للإتصالات يجب أن يتم تخزينهم . يمكننا وضعها في قائمة, لكن من الأفضل وضعها في قاموس, و ذلك لسببين : الأول هو أننا بحاجة إلى إضافة أو إزالة هذه المراجع في أي ترتيب, لأن العملاء سيتصلون و يقطعون الإتصال بإرادتهم . و السبب الثاني هو أننا يمكن أن نقوم بسهولة بتعريف معرف فريد لكل إتصال, و الذي يمكن أن يكون بمثابة مفتاح وصول في القاموس . هذا المعرف سوف يقدم تلقائيا من قبل الصنف **Thread()** .

* السطور من 47 إلى 51 : يبدأ البرنامج هنا من خلال تكرار حلقة دائمة, و التي من شأنها أن تنتظر بإستمرار عن الإتصالات الجديدة . لكل واحدة من هذه الإتصالات, يتم إنشاء كائن جديد **()ThreadClient** و التي يمكن العناية بها بشكل مستقل عن الآخرين .

* السطور من 52 إلى 54 : الحصول على معرف فريد يتم بإستخدام الأسلوب **getName()** . يمكننا هنا أن نتمتع لأن البايثون يقوم تلقائيا بتعيين إسم فريد لكل موضوع جديد : هذا الإسم كمعرف (أو مفتاح) للعثور على إتصال مناظر في قاموسنا . سوف ترى أنه عبارة عن سلسلة, من نموذج "Thread-N" (الحرف N يدل على ترتيب الموضوع) .

* السطور من 15 إلى 17 : ضع في إعتبارك أن الكائنات **ThreadClient()** كإتصالا, و جميع هذه الكائنات تعمل بشكل مواز . الأسلوب **getName()** يمكن إستخدامه داخل أي من هذه العناصر للعثور على هويته . و نحن نستخدم هذه المعلومات للتمييز بين الإتصال الحال من الجميع الآخرين (أنظر للسطر 26) .

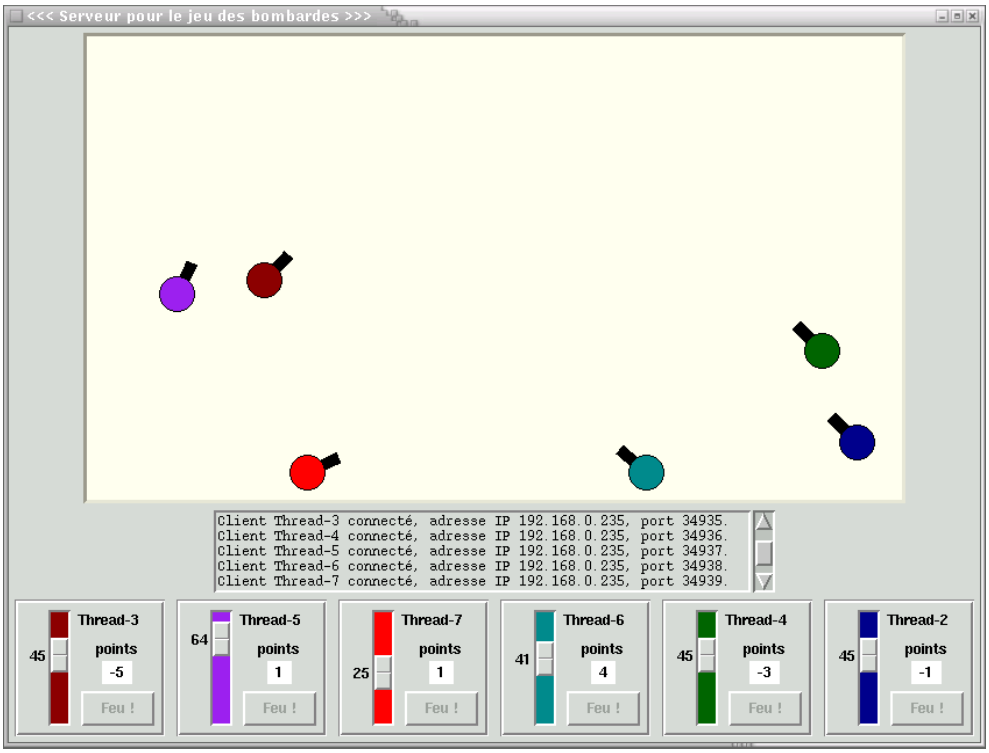
* السطور من 18 إلى 23 : فائدة الموضوع هو الحصول على كل الرسائل من عميل معين . لذلك هي حلقة دائمة التكرار, و التي سوف تتوقف عندما تستلم رسالة معينة "fin", أو عند إستلام رسالة فارغة (عند قطع الإتصال من شريك) .

* السطور من 24 إلى 27 : كل رسالة مستقبلة من عميل يجب إعادة توجيهها إلى كل الآخرين . نحن نستخدم هما حلقة for لتكرار على(التدوير) مفاتيح القاموس و الإتصالات, و التي تسمح لنا بعد ذلك بإيجاد الإتصالات نفسها . وهنالك إختبار بسيط (السطر 26) يمنعنا من إعادة إرسال الرسالة إلى العميل الذي من أين يأتي .

* السطر 31 : عندما نغلق socket إتصال, فمن الأفضل إزالة مرجعها في القاموي, لأنه هذا المرجع لم يعد يستخدم . و يمكننا القيام بذلك دو إتخاذ إحتياطات خاصو, حيث أن عناصر القاموس غير مرتبة (و نحن يمكننا إضافة أو إزالة في أي ترتيب) .

لعبة القصف, نسخة الشبكة

في الفصل 15 علقنا في تطوير لعبة قتال صغيرو التي تواجه اللاعبين بالقصف . إن فائدة هذه اللعبة



لا تزال محدوة للغاية, كما أنها تلعب على حاسوب واحد . سوف نقوم الأن, بإضافة تقنيات اللتي تعلمناها الأن . مثل نظام "الدردشة" الذي شرحناه في الصفحات السابقة, فإن التطبيق الكامل سوف يتكون الأن من برنامجين متفصلين : برنامج خادم الذي سيتم تشغيله على جهاز واحد, و تطبيق

العميل الذي يمكن تشغيله من أي جهاز .و نظر لطبيعة البايثون المحمولة, سوف تكون قادر على تنظيم معارك بين أجهزة الحاسوب التي تديرها أنظمة تشغيل مختلفة (لينكس ↔ ويندوز ↔ ماك) .

برنامج الخادم : فكرة عامة

برامج الخادم و العميل يشغلون نفس قاعدة البرنامج, في حد ذاتها تعافى إلى حد كيبر عن ماسبق التي وضعت في جميع أنحاء الفصل 15 . لذا نفترض هذا لما تبقى من هذه الدراسة التي تم حفظ نسختين سابقا من اللعبة في ملفات وحدات **canon03.py** و **canon04.py**, المثبتة في الدليل الحالي . في الواقع يمكننا إستخدام الكثير من التعليمات البرمجية التي تحتويها, و سنستخدم بحكم إستدعاء و وراثة الأصناف .

من وحدة **canon04**, سوف نعيد إستخدام الصنف **Canon** على هذا النحو, فإنه لكلا من البرناج الخادم للبرنامج العميل . من هذه الوحدة, سوف نستدعي الصنف **AppBombardes()**, والذي سوف نشتق منه الصنف الرئيسي من تطبيق الخادمنا : **AppServeur()** . سوف تجد أيضا أنها تنتج بنفسها الصنف الفرعي **AppClient()**, ودائما عن طريق الميراث .

من وحدة **canon3**, سوف نستدعي صنف **Pupitre()** التي من شأنها أن تؤدي إلى نسخة أكثر ملائمة ل "التحكم عن بعد" .

و أخيرا, سيتم إضافة صنفين جديدين إلى ما سبق, كل واحدة متخصصة في إنشاء كائن موضوع : الصنف **ThreadClients()**, منها مثيل الذي يراقب بإستمرار لتلقي طلبات socket من العملاء الجدد و الصنف **ThreadConnexion()**, الذي سينشئ كائنات socket اللازمة لإجراء حوار مع كل عميل متصل بالفعل .

هذه الأصناف الجديدة ستكون مصدر إلهام من تلك التي قمنا بتطويرها لخادم الدردشة في الصفحات السابقة . و الفرق الرئيسي هو أن لدينا ترابط محدد لتفعيل مواضيع خاصة للكود التي الذي ينتظر إتصالات العملاء, بحيث يمكن للتطبيق الرئيسي أن يفعل شيئا أخر خلال ذلك الوقت .

من هنا, عملنا الكبير الذي نواجه هو تطوير بروتوكول إتصال للحاور بين الخادم و العملاء . ما هو الجديد ؟ ببساطة نقوم بتعريف مصمون الرسائل التي سيتم تبادلها الآلات المتصلة . لا تقلق : تطوير

هذه "اللغة" يمكن أن يكون متقدما . نبدأ من خلال إنشاء قاعدة حوار, ثم نضيف تدريجيا "مفردات" .

و يمكن تحقيق الكثير من هذا العمل من خلال مساعدة برنامج العميل الذي سبق و وضعناه لنظام الدردشة . و يستخدم لإرسال "الأوامر" إلى الخادم الذي نطوره, و يتم تصحيح الإطاعة : يوضوح, الإجراءات الذي سوف نضعها تدريجيا على خادم سوف يتم إختبارخا تدريجبا, إستجابة لرسائل التي أصدر "باليد" العميل .

بروتوكول الإتصالات

و غني عن القول أن البروتوكول هو الموضح أدناه هو إجراء تعسفي تماما . و سيكون من الممكن تماما إختيار إتفاقيات أخرى مختلفة تماما . يمكنك بالطبع إنتقاد الإختيارات, و ربما ترغب في إستبدالها بأخرى أكثر فاعلية أو أبسط .

أنت تعرف مسبقا أن الرسائل المتبادلة هي سلاسل بسيطة من وحدات البايتات . توقع أن هذه الرسائل سوف تنقل العديد من المعلومات في وقت واحد, و لقد قررنا أن كل واحد منهم يمكن أن يحمل العديد من المجالات, التي ستفصل بفواصل . عند إستلام أي من هذه الرسائل, يمكننا بعد ذلك بسهولة إستعادة جميع مكونات في القائمة, و 1لك بإستخدام أسلوب المدمج **split()** .

هذا مثال عن نوع تحاور, كما يمكن إتباعها على جانب العميل . الرسائل النجمية هي التي وردة من الخادم , و الأخرة هي تلك الصادرة من قبل العميل نفسه :

```
1#    *serveur OK*
2#    client OK
3#    *canons,Thread-3;104;228;1;dark red,Thread-2;454;166;-1;dark blue,*
4#    OK
5#    *nouveau_canon,Thread-4,481,245,-1,dark green,le_vôtre*
6#    orienter,25,
7#    feu
8#    *mouvement_de,Thread-4,549,280,*
9#    feu
10#   *mouvement_de,Thread-4,504,278,*
11#   *scores,Thread-4;1,Thread-3;-1,Thread-2;0,*
12#   *angle,Thread-2,23,*
13#   *angle,Thread-2,20,*
14#   *tir_de,Thread-2,*
15#   *mouvement_de,Thread-2,407,191,*
16#   *départ_de,Thread-2*
17#   *nouveau_canon,Thread-5,502,276,-1,dark green*
```

عندما يبدأ عميل جديد,يرسل طلب إتصال إلى الخادم, الذي يرسل له رسالة عودة : "serveur OK" . عند إستلام هذا الأخير, سييتجب العميل من خلال إرسال "client OK" . هذا الأول تبادل

مجملات و التي هي ليست ضرورية, لكنه يضمن أن البلاغ على ما يرام في كلا الإتاجهين . ذلك يحذر أن العميل على إستعداد ااعمل, سيقوم الخادم بإرسال وصف للمدافع بالفعل في اللعبة (إن وجدت) : إسم المستخدم, الموقع على اللوحة. التوجه, و اللون (السطر 3) .

* ردا على إستلام العميل (السطر 4), سيقوم الخادم بتثبيت مدفع حديد في فضاء اللعبة, فهو يشير إلى خصائص التثبيت, و ليس فقط للعميل الذي تسبب بذلك, و لكن حتى جميع العملاء الآخرين المتصلين . و الرسالة المرسلة للعميل الجديد تحمل فرق (لأنه هو صاحب مدفع الجديد) : بالإضاف إلى الممبزات الموجودة في المدفع, و التي يتم توفيرها للجميع, و لديه حقل إضافي يحتوي ببساطة على "le_vôtre" (قارن على سبيل المثال بين السطر 5 مع السطر 17, مما يدل على إتصاله من لاعب أخر) . هذه الإشارة الإضافية تسمح للعميل بتمييز بين مدفع, في رسائل عدة متماثلة, و التي تحتوي على تعريف موحد المسند إلى الخادم .

الرسائل في السطور 6 و 7 هي أوامر مرسلة من قبل العميل (إنشاء و التحكم في إطلاق النار) . في النسخة السابقة من اللعبة, كنا قد وافقنا على أن المدافع ستتحرك قليلا (و بشكل عشوائي) بعد كل طلقة . بالتالي فإن الخادم هو الذي سيقوم بهذه العملية, ثم يقوم بإرسال النتائج إلى كافة المستخدمين المتصلين . الرسالة الواردة إلى الخادم في السطر 8 هي مؤشر على هذه الخطوة (الإحداثيات المقدمة هي إحداثيات الناتجة عن المدفع المعني) .

* السطر 11 يقوم بنسخ نوع الرسالة المرسلة من قبل الخادم عندما يتم ضرب الهدف . و ترسل أيضا نتائج اللاعبين الجديدة إلى جميع اللاعبين لجميع العملاء .

رسائل الخادم في الأسطر 12 و 13 و 14 تشير إلى الإجراءات التي إتخذها اللاعب الأخر (إعداد تتبع إطلاق النار) . مرة أخرى, يتم نقل المدفع عشوائيا بعد كل إطلاق نار (السطر 15) .

* السطران 16 و 17 : عندما يقطع أحد العملاء الإتصلا, يقوم الخادم بإعلام العملاء الآخرين, بإختفاء المدفع في فضاء اللعب على جميع الأماكن . و على العكس, يمكن للعملاء الجدد أن يقوموا بإتصال جديد في أي وقت للعب اللعبة .

ملاحظات إضافية

الحقل الأول من كل رسالة يشير إلى مضمونها . الرسائل المرسلة من العميل بسيطة جدا : فهي تتوافق مع الإجراءات التي إتخذها معظم اللاعبين (تغيير في زاوية الإطلاق و السيطرة على النار) . و التي تم إرسالها من قبل الخادم هي قليلا أكثر تعقيدا . يتم إرسال معظمهم إلى كافة المستخدمين المتصلين للحفاظ على التقدم المحرز . و نتيجة ذلك, يجب أن تكون لهذه الرسائل معرف اللاعب الذي يسيطر على العمل أو يتأثر أو يغير . لقد رأينا أعلاه أن هذه المعرفات هي أسماء تم صنعها تلقائيا من قبل خادم المواضيع, في كل مرة يتصل بها عميل جديد .

بعض رسائل اللعبة تحتوي على معلومات بالمجال . في هذه الحالة, التغييرات "المجالات-الفرعية" تكون مفصولة بفواصل منقوطة (السطران 3 و 11) .

برنامج خادم : الجزء الأول

سوف نجد في الصفحات التالية سكريبت كامل لبرنامج خادم . سوف نقدم إليك في ثلاثة قطع على التوالي تقليقات الكود, و لكن ترقيم الأسطر مستمر . على الرغم من أنه بالفعل طويل جدا و معقد, سوف تشغر أنك تستحق ربما المزيد من التحسين, لا سيما من حيث العرض العام . نترك لك الأمر لإضافط بنفسك جميع الملاحق التي قد تبدو مفيدة (على سبيل المثال, إقتراح لإختيار إحداثيات الجهاز المصيف عند بدء التشغيل, و شريط قوائم و إلخ) :

```
1#      ##########################################################
2#      # لعبة القصف - نسخة الخادم                              #
3#      # (C) Gérard Swinnen, Verviers (Belgique) - July 2004 #
4#      # رخصة : GPL                          rév. 2010 #
5#      # قبل تشغيل هذا السكريبت, تأكد من أن عنوان  #
6#      # IP في الأسفل نفس في جهازك  #
7#      # يمكنك إختيار رقم منفذ آخر, أو #
8#      # تغيير إحداثيات فضاء اللعبة .  #
9#      # في جميع الأحوال, تأكد من أن نفس الإختيارات #
10#     # تم وضعها في كل سكريبت عميل  #
11#     ##########################################################
12#
13#     host, port = '192.168.1.168', 36000
14#     largeur, hauteur = 700, 400          # إحداثيات فضاء اللعبة
15#
16#     from tkinter import *
17#     import socket, sys, threading, time
18#     import canon03
19#     from canon04 import Canon, AppBombardes
20#
21#     class Pupitre(canon03.Pupitre):
22#        """Pupitre de pointage amélioré"""
23#        def __init__(self, boss, canon):
24#           canon03.Pupitre.__init__(self, boss, canon)
25#
```

```
26#     def tirer(self):
27#        "déclencher le tir du canon associé"
28#        self.appli.tir_canon(self.canon.id)
29#
30#     def orienter(self, angle):
31#        "ajuster la hausse du canon associé"
32#        self.appli.orienter_canon(self.canon.id, angle)
33#
34#     def valeur_score(self, sc =None):
35#        "imposer un nouveau score <sc>, ou lire le score existant"
36#        if sc == None:
37#           return self.score
38#        else:
39#           self.score =sc
40#           self.points.config(text = ' %s ' % self.score)
41#
42#     def inactiver(self):
43#        "désactiver le bouton de tir et le système de réglage d'angle"
44#        self.bTir.config(state =DISABLED)
45#        self.regl.config(state =DISABLED)
46#
47#     def activer(self):
48#        "activer le bouton de tir et le système de réglage d'angle"
49#        self.bTir.config(state =NORMAL)
50#        self.regl.config(state =NORMAL)
51#
52#     def reglage(self, angle):
53#        "changer la position du curseur de réglage"
54#        self.regl.config(state =NORMAL)
55#        self.regl.set(angle)
56#        self.regl.config(state =DISABLED)
57#
```

* الصنف **Pupitre()** تم بنائه عن طريق الإشتقاق من صنف لديه نفس الإسم تم إستدعائه من وحدة **canon03** . و لذلك فإنه[102] يرث جميع خصائصه, لكن نحن بحاجة بتجاوز أساليبه **tirer()** و **orien()** **ter** .

* في إصدار الفردي للبرنامج, في الواقع, يمكنك التحكم بكل مدفع من خلال لوحة التحكم . في نسخة الشبكة, العميل هو الذي سيقوم بالتحكم عن بعد بتشغيل المدافع . و لذلك, العميل هو الذي يتحكم عن بعد بالمدفع . و لا يمكن للواحات التحكم التي تظهر في النافذة الخادم بتكرار المناورات التي يقوم بها الاعبون من خلال كل عميل . زر الإطلاق و مؤشر الإرتفاع معطلان (غير مفعلان), لكن المؤشرات تطيع الأوامر التي تم إرسالها من قبل التطبيق الرئيسي .

102تذكير : في الصنف المشتق, يمكنك تعريف أسلوب جديد مع نفس إسم أسلوب الصنف الأصل, لتغيير وظائفها في الصنف المشتق . و هذا يسمى تجاوز هذا الأسلوب (أنظر أيضا إلى صفحة Error: Reference source not found).

* هذا الصنف الجديد **Pupitre()** سوف يستخدم في كل نسخة من برنامج العميل . في النافذته مثل التي في الخادم, جميع لوحات التحكم تظهر كمكررات, لكن واحد منهم سيكون جاهزا تماما : الذي يتوافق مع مدفع اللاعب .

* كل هذه الأسباب تؤدي إلى ظهور أساليب جديدة **activer()** و **desactiver()** و **()** **reglage** و **valeur_score()**, الذين سيتم إستدعائهم من قبل التطبيق الرئيسي, ردا على رسائل المتبادلة بين الخادم و عملائه .

* يتم إستخدام الصنف **ThreadConnexion()** أدناه لإنشاء مثيل لمجموعة من الكائنات المواضيع التي تتسلم بالتوازي جميع الإتصالات من العملاء . يحتوي أسلوبه **run()** على الوظائف الأساسية للخادم, حلقة التعليمات التي تدير إستقبال الرسائل من عميل معين, و التي لكل واحدة منها سلسلة من ردود الفعل . سوف تجد تنفيذ ملموس للبروتوكول الإتصال الموضح في الصفحات السابقة (بعض الرسائل تم صنعها بواسطة الأساليب **depl_aleat_canon()** و **goal()** من صنف **AppServeur()** الذي سيتم شرحه لاحقا) .

```
58#   class ThreadConnexion(threading.Thread):
59#     """objet thread gestionnaire d'une connexion client"""
60#     def __init__(self, boss, conn):
61#       threading.Thread.__init__(self)
62#       self.connexion = conn        #مرجع socket الإتصال
63#       self.app = boss              #مرجع نافذة التطبيق
64#
65#     def run(self):
66#       "actions entreprises en réponse aux messages reçus du client"
67#       nom = self.getName()         #معرف العميل = إسم الخيط
68#       while 1:
69#         msgClient = self.connexion.recv(1024).decode("Utf8")
70#         print("**{0}** de {1}".format(msgClient, nom))
71#         deb = msgClient.split(',')[0]
72#         if deb == "fin" or deb =="":
73#           self.app.enlever_canon(nom)
74#           #: علامة بداية هذا المدفع للعملاء الاخرين
75#           self.app.verrou.acquire()
76#           for cli in self.app.conn_client:
77#             if cli != nom:
78#               message = "départ_de,{0}".format(nom)
79#               self.app.conn_client[cli].send(message.encode("Utf8"))
80#           self.app.verrou.release()
81#           #: إغلاق هذا الموضوع
82#           break
83#         elif deb =="client OK":
84#           #: علامة إلى عميل جديد أن المدافع مسجلة بالفعل
85#           msg ="canons,"
86#           for g in self.app.guns:
87#             gun = self.app.guns[g]
```

```
88#             msg =msg +"{0};{1};{2};{3};{4},".\
89#                format(gun.id, gun.x1, gun.y1, gun.sens, gun.coul)
90#          self.app.verrou.acquire()
91#          self.connexion.send(msg.encode("Utf8"))
92#          #إنتظار الحصول على إعتراف ('OK') :
93#          self.connexion.recv(100).decode("Utf8")
94#          self.app.verrou.release()
95#          #إضافة مدفع إلى فضاء لعب الخادم
96#          #: الأسلوب الذي تم إستدعاؤه يرجع صفات للمدفع الذي تم صنعه :
97#          x, y, sens, coul = self.app.ajouter_canon(nom)
98#          #: إرسال الصفات لهذا المدفع الجديد إلى كل العملاء المتصلة
99#          self.app.verrou.acquire()
100#          for cli in self.app.conn_client:
101#             msg ="nouveau_canon,{0},{1},{2},{3},{4}".\
102#                format(nom, x, y, sens, coul)
103#            #: للعميل الجديد, أضف حقل مشير إلى أن الرسالة تتعلق بالمدفع الخاص بها
104#             if cli == nom:
105#                msg =msg +",le_vôtre"
106#             self.app.conn_client[cli].send(msg.encode("Utf8"))
107#          self.app.verrou.release()
108#        elif deb =='feu':
109#          self.app.tir_canon(nom)
110#          #: الإشارة إلى هذا الإطلاق النار لجميع العملاء الآخرين
111#          self.app.verrou.acquire()
112#          for cli in self.app.conn_client:
113#             if cli != nom:
114#                message = "tir_de,{0},".format(nom)
115#                self.app.conn_client[cli].send(message.encode("Utf8"))
116#          self.app.verrou.release()
117#        elif deb =="orienter":
118#          t =msgClient.split(',')
119#          #: يمكن أن نستقبل العديد من الزوايا . نقوم بإستخدام الأخيرة
120#          self.app.orienter_canon(nom, t[-1])
121#          #الإشارة لهذه التغييرات لجميع العملاء الآخرين
122#          self.app.verrou.acquire()
123#          for cli in self.app.conn_client:
124#             if cli != nom:
125#               #: نقطة نهاية لأن, الرسائل قد تكون مجمعة في بعض الأحيان
126#                message = "angle,{0},{1},".format(nom, t[-1])
127#                self.app.conn_client[cli].send(message.encode("Utf8"))
128#          self.app.verrou.release()
129#
130#      #إغلاق الإتصال
131#      self.connexion.close()        #قطع الإتصال
132#      del self.app.conn_client[nom]   #حذف مرجعه في القاموس
133#       self.app.afficher("Client %s déconnecté.\n" % nom)
134#      #الخيط ينتهي هنا
```

تزامن الخيوط بإستخدام الأقفال (thread locks)

من خلال فحص الكود أعلاه, سوف تلاحظة بنية معينة من كتل التعليمات التي يرسل الخادم نفس لرسالة إلى جميع عملائه . على سبيل المثال أنظر إلى الأسطر من 74 إلى 80 .

السطر 75 يقوم ينشيط الأسلوب **acquire()** لكائن "مغلق" الذي تم إنشاءه عن طريق المنشئ للتطبيق الرئيسي (أنظر أدناه) . هذا الكائن هو مثيل الصنف **Lock()**, الذس هو جزء من وحدة **threading** التي قمنا بإستدعائها في بداية السكريبت . الأسطر التالية (من 76 إلى 79) تسبب إرسال رسالة إلى كافة العملاء المتصلين (بإستثناء واحد) . ثم, يكون لكائن-القفل مسعى جديد, هذه المرة لأسلوبه **release()** .

ماذا يخدم هذا كائن-القفل إذا ؟ بما أنه يتم صنعه من صنف من وحدة **threading**, يمكنك تخمين أن له فائدة للمواضيع . في الواقع, تستخدم هذه كائنات-القفل لمزامنة المواضيع المتزامنة . ما هو ؟ هل تعلم أن الخادم يبدء بموضوع مختلف لكل عميل متصل . ثم, تصبح كل هذه المواضيع تعمل بالتوازي . و بالتالي هنالك خطر في بعض الأحيان, و هو ربما أن موضوعين أو أكثر يمكن أن يستخدموا مورد مشترك في نفس الوقت .

في الأسطر البرمجية التي ناقشناها, على سبيل المثال, نحن نتعامل مع موضوع الذي يريد إستخدام تقريبا جميع الإتصالات الموجود لنشر الرسالة . و لذلك من الممكن أنه خلال هذا الوقت, موضوع آخر يريد أن يستخدم أيضا واحدة أو أخرى من هذه الإتصالات, و التي قد يتسبب في عطب (أي تطابق بشكل فوضوي عدة رسائل) .

يمكن حل هذه المشكلة بإستخدام كائن-القفل (thread lock) . هذا الكائن لا يتم إنشائه إلا في نسخة واحدة, في مساحة الأسماء التي في متناول جميع المواضيع المتزامنة . و يتميز أساسا في أنه دائما في حالة أو أخرى : مغلق أو غير مغلق . حالته الأولية هي غير مغلق .

إستخدام

عندما يكون أي موضوع على وشك الوصول إلى مورد مشترك, يتك تفعيل أولا الأسلوب **() acquire** للقفل . فإذا كانت الحالة غير مغلق, يتم غلقه, و يمكن للموضوع الذي طلبه أن يستخدم مصادر مشتركة بأمان . عند الإنتهاء من إستخدام المورد, فإنه سوف يقوم بتفعيل الأسلوب **() release** للقفل, الذي سيقون بفتح قفله (يكون غير مغلق) .

في الواقع, إذا كان موضوع أخر يحاول تفعيل هو أيضا الأسلوب **acquire()** للقفل, عندما يكون في حالة مقفل, يكون الأسلوب "ليس في متناول يده", مما يتسبب في منع هذا الموضوع, و الذي

يقوم بتعليق نشاطه حتى يعود إلى حالت غير مقفل . و هذا بالتالي يمنع الوصول إلى مورد مشترك خلال الوقت الذي يستخدم فيه موضوع آخر . عندما يتم فتح القفل, واحدة من المواضيع الإنتظار (قد يكون في الواقع أكثر من واحد) يستأنف نشاطه أثناء إغلاق القفل, و هكذا .

يقوم كائن-القفل بحفظ مراجع المواضيع الذي قد منعها, بحيث أنه يتم إزالة المنع لواحد فقط عندما يتم إستدعاء الأسلوب **release()** . و يجب دائما لكل موضوع يتم تفعيله بإستخدام الأسلوب **acquire()** قبل الوصول إلى الموارد, و يجب عليه نفعيل أسلوبه **release()** بعد ذلك .

شرط أن تكون جميع المواضيع المتزامنة تتبع نفس الإجراء, هذه التقنية بسيطة تمنع إمكانية إستخدام مورد مشترك في وقت واحد من قبل العديد منهم . نقول قي هذه الحالة المواضيع قد تم مزامنتها .

برنامج الخادم : إنهائه

الصنفين بالأسفل تكمل السكريبت الخادم . يتم تنفيذ التعليمات البرمجية في الصنف **ThreadClients()** مماثلة لتلك التي طورناها سابقا لجسم تطبيق برنامج الدردشة . في هذه الحالة, و مع ذلك, وضعنا في صنف مشتق من **Thread()**, لأنه يحتاج إلى تشغيل هذا الكود في موضوع مستقل عن التطبيق الرئيسي . و بالفعل تم القبض عليه عن طريق حلقة **mainloop()** للواجهة الرسومية[103].

يشتق الصنف **AppServeur()** من صنف **AppBombardes()** لوحدة **canon04** . و لقد أضفنا مجموعة من الأساليب المكملة للقيام بتنفيذ جميع العمليات التي تنتج عن حوار مع العملاء . لاحظنا سابقا أن العملاء يمثلون نسخة مشتقة من هذا الصنف (تمتع بنفس التعاريق الأساسية للنافذة, اللوحة, إلخ) .

```
135#  class ThreadClients(threading.Thread):
136#     """objet thread gérant la connexion de nouveaux clients"""
137#     def __init__(self, boss, connex):
138#        threading.Thread.__init__(self)
139#        self.boss = boss          # مرجع نافذة التطبيق
140#        self.connex = connex      # مرجع socket المبدئي
141#
142#     def run(self):
143#        "attente et prise en charge de nouvelles connexions clientes"
```

103سوف نناقش هذا السؤال في الصفحات القادمة, لأنه يفتح بعض الوجهات المثيرة للإهتمام : أنظر إلى : تحسين الرسوم المتحركة بإستخدام الخيوط, صفحة Error: Reference source not found.

```
144#        txt ="Serveur prêt, en attente de requêtes ...\n"
145#        self.boss.afficher(txt)
146#        self.connex.listen(5)
147#       #: إدارة الإتصالات المطلوبة من قبل العملاء
148#        while 1:
149#          nouv_conn, adresse = self.connex.accept()
150#          # صنع كائن خيط جديد لصنع الإتصال :
151#          th = ThreadConnexion(self.boss, nouv_conn)
152#          th.start()
153#         it = th.getName()       # معرف فريد للخيط
154#          # حفظ الإتصال في قاموس :
155#          self.boss.enregistrer_connexion(nouv_conn, it)
156#          #: إظهاره
157#          txt = "Client %s connecté, adresse IP %s, port %s.\n" %\
158#                (it, adresse[0], adresse[1])
159#          self.boss.afficher(txt)
160#          #: بدء التواصل مع العميل
161#          nouv_conn.send("serveur OK".encode("Utf8"))
162#
163#  class AppServeur(AppBombardes):
164#    """fenêtre principale de l'application (serveur ou client)"""
165#    def __init__(self, host, port, larg_c, haut_c):
166#        self.host, self.port = host, port
167#        AppBombardes.__init__(self, larg_c, haut_c)
168#       self.active =1        # مؤشر النشاط
169#       #: تأكد من خروجك بشكل صحيح عند إغلاق النافذة
170#        self.bind('<Destroy>',self.fermer_threads)
171#
172#    def specificites(self):
173#        "préparer les objets spécifiques de la partie serveur"
174#        self.master.title('<<< Serveur pour le jeu des bombardes >>>')
175#
176#       #: ويدجت نص, مرتبط مع شريط تمرير
177#        st =Frame(self)
178#        self.avis =Text(st, width =65, height =5)
179#        self.avis.pack(side =LEFT)
180#        scroll =Scrollbar(st, command =self.avis.yview)
181#        self.avis.configure(yscrollcommand =scroll.set)
182#        scroll.pack(side =RIGHT, fill =Y)
183#        st.pack()
184#
185#       # جزء خادم الإتصال
186#       self.conn_client = {}         # قاموس إتصالات العميل
187#       self.verrou =threading.Lock()   # قفل مزامنة الخيوط
188#       # تهيئة الخادم - وضع socket :
189#        connexion = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
190#        try:
191#          connexion.bind((self.host, self.port))
192#        except socket.error:
193#          txt ="La liaison du socket à l'hôte %s, port %s a échoué.\n" %\
194#               (self.host, self.port)
195#          self.avis.insert(END, txt)
196#          self.accueil =None
197#        else:
198#          #: بدء خيط لمراقبة إتصال العملاء
199#          self.accueil = ThreadClients(self, connexion)
200#          self.accueil.start()
201#
202#    def depl_aleat_canon(self, id):
203#        "déplacer aléatoirement le canon <id>"
```

```
204#        x, y = AppBombardes.depl_aleat_canon(self, id)
205#        #: الإشارة لهذه الإحداثيات الجديدة لكافة العملاء
206#        self.verrou.acquire()
207#        for cli in self.conn_client:
208#            message = "mouvement_de,%s,%s,%s," % (id, x, y)
209#            self.conn_client[cli].send(message.encode("Utf8"))
210#        self.verrou.release()
211#
212#    def goal(self, i, j):
213#        "le canon <i> signale qu'il a atteint l'adversaire <j>"
214#        AppBombardes.goal(self, i, j)
215#        #: الإشارة إلى النتائج الجديدة لكافة العملاء
216#        self.verrou.acquire()
217#        for cli in self.conn_client:
218#            msg ='scores,'
219#            for id in self.pupi:
220#                sc = self.pupi[id].valeur_score()
221#                msg = msg +"%s;%s," % (id, sc)
222#            self.conn_client[cli].send(msg.encode("Utf8"))
223#        time.sleep(.5)          # للتحسين نفصل الرسائل
224#        self.verrou.release()
225#
226#    def ajouter_canon(self, id):
227#        "instancier un canon et un pupitre de nom <id> dans 2 dictionnaires"
228#        # on alternera ceux des 2 camps :
229#        n = len(self.guns)
230#        if n %2 ==0:
231#            sens = -1
232#        else:
233#            sens = 1
234#        x, y = self.coord_aleat(sens)
235#        coul =('dark blue', 'dark red', 'dark green', 'purple',
236#               'dark cyan', 'red', 'cyan', 'orange', 'blue', 'violet')[n]
237#        self.guns[id] = Canon(self.jeu, id, x, y, sens, coul)
238#        self.pupi[id] = Pupitre(self, self.guns[id])
239#        self.pupi[id].inactiver()
240#        return (x, y, sens, coul)
241#
242#    def enlever_canon(self, id):
243#        "retirer le canon et le pupitre dont l'identifiant est <id>"
244#        if self.active == 0:      # تم إغلاق النافذة
245#            return
246#        self.guns[id].effacer()
247#        del self.guns[id]
248#        self.pupi[id].destroy()
249#        del self.pupi[id]
250#
251#    def orienter_canon(self, id, angle):
252#        "régler la hausse du canon <id> à la valeur <angle>"
253#        self.guns[id].orienter(angle)
254#        self.pupi[id].reglage(angle)
255#
256#    def tir_canon(self, id):
257#        "déclencher le tir du canon <id>"
258#        self.guns[id].feu()
259#
260#    def enregistrer_connexion(self, conn, it):
261#        "Mémoriser la connexion dans un dictionnaire"
262#        self.conn_client[it] = conn
263#
```

```
264#     def afficher(self, txt):
265#       "afficher un message dans la zone de texte"
266#       self.avis.insert(END, txt)
267#
268#     def fermer_threads(self, evt):
269#       "couper les connexions existantes et fermer les threads"
270#       #: قطع الإتصالات مع جميع العملاء
271#       for id in self.conn_client:
272#         self.conn_client[id].send('fin'.encode("Utf8"))
273#       #: فرض إنهاء خيط خادم إنتظار الإستعلامات (الطلبات)
274#       if self.accueil != None:
275#         self.accueil._stop()
276#       self.active =0            #منع وصول إلى لاحقة Tk
277#
278#  if __name__ =='__main__':
279#     AppServeur(host, port, largeur, hauteur).mainloop()
```

تعليقات

* السطر 173 : سيحدث من وقت لأخر يريد "الإعتراض " على إغلاق التطبيق الذي قام المستخدم بالخروج من برنامجك, على سبيل المثال لأنك ترير إجباره بحفظ نسهة من البيانات المهمة في ملف, أو غلق نوافذ أخرى, إلخ . فقط إجعله يكشف عن العنصر **<Destroy>**, كما فعلنا هنا لإجبار إنهاء جميع المواضيع النشطة .

* السطور من 179 إلى 186 : هنا عليك مراجعة تقنية ربط شريط التمرير مع ويدجت "نص - **Text**" (أنظر للصفحة 228) .

* السطر 190 : إنشاء مثيل لكائن-قفل يسمح بمزامنة المواضيع .

* السطران 202 و 203 : هنا يتم تمثيل كائن الموضوع الذي ينتظر بشكل مسمتر طلبات إتصال العملاء المحتملين .

* السطور من 205 إلى 213 و من 215 إلى 227 : هذه الأساليب تجاوز الأساليب من نفس الإسم التي ورثت من صنف الأصل . فهي تبدأ بإستدعائهم للقيام بنفس العمل (السطور من 207 إلى 217), و من ثم إضافة وظائفهم الخاصة, و هو الذي يقدم الجميع ما حدض الأن .

* السطور من 229 إلى 243 : هذا الأسلوب يمثل موقف جديد للإطلاق في كل مرة يتصل فيها عميل جديد . يتم وضع المدافع بالتناوب في الجانب الأيمن و الأيسر, و هو إجراء من الواضح أنه يمكن أن يتحسن . قائمة الألوان الموجود تحد عدد العملاء إلى 10, و الذي ينبغي أن يكون كافيا .

برنامج العميل

سكريبت برنامج العميل أدناه . كما في الخادم, فهو قصير نسبيا, لأنه يستخدم أيضا إستدعاء الوحدات و يرث الأصناف . يجب على سكريبت الخادم أن يتم حفظه في ملف-وحدة تسمى **canon_serveur.py** . و هذا الملف يجب وضعه في الدليل الحالي, و كذلك ملفات-الوحدات **canon03.py** و **canon04.py** الذي يستخدمهم .

هذه الوحدات تم إستدعائها, هذا السكريبت يستخدم الأصناف **Canon()** و **Pupitre()** بشكل مطابق, و المشتقة من صنف **AppServeur()** . في هذه الأخيرة, العديد من الأساليب الثقيلة للتكيف مع وظائفها . على سبيل المثال الأساليب **goal()** و **depl_aleat_canon()** , لا تفعل شيئا (العبارة pass), لأنه لا يمكن حساب النتائج و إعادة تموضع المدافع بعد كل إطلاق للنار لأن هذا يتم من الخادم فقط .

سوف تجد في الأسلوب **run()** للصنف **ThreadSocket()** (الأسطر من 86 إلى 126) كود الذي يعالج الرسائل المتبادلة مع الخادم . و نحن أبقينا تعليمة print (في السطر 88) ذلك لأن الرسائل المتلقاة من الخادم تظهر على الإخراج القياسي . فإذا قمت بنفسك شكل أكثر تحديدا لهذه اللعبة, يمكنك بالطبع حذف هذه التعليمات .

```
1#      ##########################################################
2#      #لعبة القصف – نسخة العميل                              #
3#      #(C) Gérard Swinnen, Liège (Belgique) - Juillet 2004 #
4#      #Licence : GPL                    Révis.  2010 #
5#      #قبل تشغيل هذا السكريبت, تأكد من عنوان, #
6#      #رقم المنفذ و إحداثيات فضاء  #
7#      #اللعبة التي تم وضعها في الأسفل هي نفسها بالضبط   #
8#      #التي تم تعريفها للخادم .          #
9#      ##########################################################
10#
11#   from tkinter import *
12#   import socket, sys, threading, time
13#   from canon_serveur import Canon, Pupitre, AppServeur
14#
15#   host, port = '192.168.1.168', 36000
16#   largeur, hauteur = 700, 400        #إحداثيات فضاء اللعب
17#
18#   class AppClient(AppServeur):
19#      def __init__(self, host, port, larg_c, haut_c):
20#         AppServeur.__init__(self, host, port, larg_c, haut_c)
21#
22#      def specificites(self):
23#         "préparer les objets spécifiques de la partie client"
24#         self.master.title('<<< Jeu des bombardes >>>')
25#         self.connex =ThreadSocket(self, self.host, self.port)
```

```
26#         self.connex.start()
27#         self.id =None
28#
29#      def ajouter_canon(self, id, x, y, sens, coul):
30#         "instancier un canon et un pupitre de nom <id> dans 2 dictionnaires"
31#         self.guns[id] = Canon(self.jeu, id, int(x), int(y), int(sens), coul)
32#         self.pupi[id] = Pupitre(self, self.guns[id])
33#         self.pupi[id].inactiver()
34#
35#      def activer_pupitre_personnel(self, id):
36#         self.id =id                    #تلقي معرف من الخادم
37#         self.pupi[id].activer()
38#
39#      def tir_canon(self, id):
40#         r = self.guns[id].feu()          #إرسال False إذا توقف
41#         if r and id == self.id:
42#            self.connex.signaler_tir()
43#
44#      def imposer_score(self, id, sc):
45#         self.pupi[id].valeur_score(int(sc))
46#
47#      def deplacer_canon(self, id, x, y):
48#         "note: les valeurs de x et y sont reçues en tant que chaînes"
49#         self.guns[id].deplacer(int(x), int(y))
50#
51#      def orienter_canon(self, id, angle):
52#         "régler la hausse du canon <id> à la valeur <angle>"
53#         self.guns[id].orienter(angle)
54#         if id == self.id:
55#            self.connex.signaler_angle(angle)
56#         else:
57#            self.pupi[id].reglage(angle)
58#
59#      def fermer_threads(self, evt):
60#         "couper les connexions existantes et refermer les threads"
61#         self.connex.terminer()
62#         self.active =0              #منع وصول إلى لاحقة Tk
63#
64#      def depl_aleat_canon(self, id):
65#         pass                    #=> أسلوب غير فعال
66#
67#      def goal(self, a, b):
68#         pass                    #=> أسلوب غير فعال
69#
70#
71#   class ThreadSocket(threading.Thread):
72#      """objet thread gérant l'échange de messages avec le serveur"""
73#      def __init__(self, boss, host, port):
74#         threading.Thread.__init__(self)
75#         self.app = boss          #مرجع نافذة التطبيق
76#         #إتصال بالخادم : - socket وضع
77#         self.connexion = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
78#         try:
79#            self.connexion.connect((host, port))
80#         except socket.error:
81#            print("La connexion a échoué.")
82#            sys.exit()
83#         print("Connexion établie avec le serveur.")
84#
85#      def run(self):
```

```
86#         while 1:
87#           msg_recu = self.connexion.recv(1024).decode("Utf8")
88#           print("*%s*" % msg_recu)
89#           # le message reçu est d'abord converti en une liste :
90#           t =msg_recu.split(',')
91#           if t[0] =="" or t[0] =="fin":
92#             #: حذف هذا الخيط
93#             break
94#           elif t[0] =="serveur OK":
95#             self.connexion.send("client OK".encode("Utf8"))
96#           elif t[0] =="canons":
97#             self.connexion.send("OK".encode("Utf8"))   #إعتراف وصول
98#             #. مسح أول و اخر عنصر في القائمة
99#             #: تبقي قوائم
100#             lc = t[1:-1]
101#             #: كل واحد هو وصف كامل للمدفع
102#             for g in lc:
103#               s = g.split(';')
104#               self.app.ajouter_canon(s[0], s[1], s[2], s[3], s[4])
105#           elif t[0] =="nouveau_canon":
106#             self.app.ajouter_canon(t[1], t[2], t[3], t[4], t[5])
107#             if len(t) >6:
108#               self.app.activer_pupitre_personnel(t[1])
109#           elif t[0] =='angle':
110#             #. من الممكن أن نستقبل مجموعة من المعلومات المجمعة
111#             #: لن نأخذ سوى الأولى
112#             self.app.orienter_canon(t[1], t[2])
113#           elif t[0] =="tir_de":
114#             self.app.tir_canon(t[1])
115#           elif t[0] =="scores":
116#             #. مسح أول و اخر عنصر في القائمة
117#             #: تبقي قوائم
118#             lc = t[1:-1]
119#             #: كل عنصر هو وصف نتيجة
120#             for g in lc:
121#               s = g.split(';')
122#               self.app.imposer_score(s[0], s[1])
123#           elif t[0] =="mouvement_de":
124#             self.app.deplacer_canon(t[1],t[2],t[3])
125#           elif t[0] =="départ_de":
126#             self.app.enlever_canon(t[1])
127#
128#         #خيط <réception> ينتهي هنا
129#         print("Client arrêté. Connexion interrompue.")
130#         self.connexion.close()
131#
132#       def signaler_tir(self):
133#         self.connexion.send("feu".encode("Utf8"))
134#
135#       def signaler_angle(self, angle):
136#         msg ="orienter,{0}".format(angle)
137#         self.connexion.send(msg.encode("Utf8"))
138#
139#       def terminer(self):
140#         self.connexion.send("fin".encode("Utf8"))
141#
142#     # Programme principal :
143#     if __name__ =='__main__':
144#       AppClient(host, port, largeur, hauteur).mainloop()
```

تعليقات

* السطران 15 و 16 : يمكنك تحسين هذا السكريبت بإضافة نموذج لطلب هذه القيم من المستخدم أثناء بدء التشغيل .

* السطور من 19 إلى 27 : ينتهي منشئ صنف الأصل بإستدعاء الأسلوب **specificites()** . يمكننا وضع فيها الذي يجب أن يبني بشكل مختلف في الخادم و العملاء . يقوم الخادم خاصة بتمثيل ويدجت "نص - text" الذي لم تظهر في العملاء, واحدة أو أخرى من الكائنات الموضوع المختلفة تبدأ للإدارة الإتصالات .

* السطور من 39 إلى 42 : يتم إستدعاء هذا الأسلوب في كل مرة يقوم فيها المستخدم بالضغط على زر إطلاق النار . لا يمكن للمدفع إلا لقطات بشكل مستمر . لذلك, لا يمكن لأي رصاصة جديدة أن تقبل إذا لم تكمل الطلقة التي قبلها مسارها , القيم "صحيح" أو "خطأ" تقوم بإرجاع بإستخدام الأسلوب **feu()** لكائن المدفع الذي يشير إلى ما إذا تم قبول إطلاق النار أو لا . يتم إستخدام هذه القيمة للإشارة إلى الخادم (و عملاء آخرين) أن إطلاق النار قد حصل فعلا .

* السطور من 105 إلى 108 : يتم إضافة مدفع جديد في فضاء لعبة لكل واحد (و هذا معناه في لوحة الخادم و في لوحات جميع العملاء المتصلين), في كل مرة يتصل فيها عميل جديد . يرسل الخادم في هذا الوقت رسالة إلى جميع العملاء لإعلامهم بهذا الشريك الجديد . لكن الرسالة ترسل على وجه الخصوص تحمل حقل إضافي (التي تحتوي ببساطة على سلسلة "le_vôtre") بحيث أن هذا الشريك يعلم أن هذه الرسالة لمدفع خاص به, و يمكن تفعيل لوحة التحكم أثناء تخزين المعرف الذي تم تعيينه من قبل الخادم (أنظر أيضا إلى السطور من 35 إلى 37) .

إستنتاجات و وجهات نظر

تم تقدين هذا التطبيق في هدف تعليمي . لقد تعمدنا تبسيط العديد من المشاكل . على سبيل المثال, إذا إختبرت بنفسك هذا البرنامج, سوف تجد أنه في كثير من الأحيان تم تجميع الرسائل المتبادلة في "الحزم", و هذا يتطلب تحسين خوارزمية تنفيذ لتفسيرها .

و بالمثل, فإننا بالكاد رسمنا آلية الأساسية للعبة : توزيع اللاعبين على معسكرين, و تدمير المدافع المعنية, و مختلف العقبات الأخرى, إلخ . و لدينا العديد من الطرق لإستكشافها !

تمارين

19.1 قم بتبسيط السكريت لعميل الدردشة الذي تم شرحه في الصفحة 356, عن طريق إزالة كائن من كائني المواضيع . و قم بترتيب على سبيل المثال معالجة إرسال الرسائل في الموضوع الرئيسي .

19.2 قم بتعديل لعبة القصف (الإصدار المستقل) في الفصل 15 (أنظر إلى صفحات 255 و ما يليها), بحفظ لوحة تحكم واحدة و مدفع واحد . و أضف هدف متحرك, و حركات سيت إدراجها من كائن موضوع مستقل (من أجل فصل أجزاء من تعليمات البرمجية التي تحكم في حركة الهدف و المقذوف) .

إستخدام المواضيح لتحسين الرسوم المتحركة

التمرين الأخير في نهاية المقطع السابق يشير إلى وجود منهجية لتطوير التطبيقات التي يمكن أن تكون مفيدة بشكل خاص في حالة ألعاب الفيديو التي تشمل العديد من الرسوم المتحركة المتزامنة .

في الواقع, إذا قمت ببرمجة مختلف عناصر الرسوم المتحركة للعبة ككائنات مستقاة تعمل كل واحدة منها على موضوع خاص بها, فأنت لا تبسط فقط هذه المهم بل حتى حسنت من إنمانية قراءة السكريبت الخاص بك, و لكن يمكنك زيادة سرعة التنفذ و بالتالي سيولة الرسومالمتحركة . لتحقيق هذه النتيجة, يجب عليك ترك مهلة للتقنية التي نستغلها حتى الأن, لكن الذي سوف تستخدم في مكانها أسهل في نهاية المطاف .

تأخير الرسوم المتحركة بإستخدام *after*()

في جميع الرسوم المتحركة التي تحدثنا عنها حتى الأن, يتم صنع "المحرك" في كل مرة من خلال دالة تحتوي على أسلوب **after()**, التي تربط تلقائيا لجميع ويدجات tkinter , أنت تعلم أن هذا الأسلوب يسمح بإدخال تأخير وقت في سلوك برنامجك : يتم تنشيط مرقت داخلي, بحيث بعد فترة من الوقت المتفق عليه, ستدعي النظام تلقائيا أي دالة . بشكل عام, الدالة التي تحتوي على **after()** يتم إستدعاؤها : فهي توفر التالي في حلقة عودية, و يبقى برمجة مواقع للكائن الرسومية المختلفة .

إعلم أنه خلال تدفق الفاصل الزمني للبرنامج بإستخدام الأسلوب **after()**, تطبيقك ليس "مجمدا" . يمكنك على سبيل المثال, خلال هذا الوقت, الضغط على زر, إعادة تحجيم النافذة, إدخال مدخلات من خلال لوحة المفاتيح, إلخ . لكن كيف يكون هذا ممكنا ؟

و لقد سبق و أن ذكرنا مرات عديدة أن التطبيقات تشمل دائما محرك رسوم حديث "يعمل" بشكل مستمر في الخلفية : هذا الجهاز يبدأ عندما تشغل الأسلوب **mainloop()** لنافذتك الرئيسية . كما يشير إسمه بوضوح, هذا الأسلوب يطبق دائما حلقة تكرار, نفس نوع خلفة while التي تعرفها جيدا . يتم تضمين العديد من آليات في هذا "المحرك" . واحدة من هذه آليات هي الحصول على جميع الأحداث التي تحدث, و بعد صنع تقرير لهم بإستخدام الرسائل الملائمة للبرامج التي تطلب ذلك (أنظر إلى : برامج تتحكم بواسطة الأحداث, صفحة 83), على أن تتخذ إجراءات تحكم أخرى في العرض, إلخ . عندما تقوم بإستدعاء أسلوب **after()**, لويدجت, سوف تستخدم في الواقع آلية توقيت و التي تتكامل مع **mainloop()** أيضا, و هذا إذا مدير المركزي الذي يقوم بإستدعاء الدالة التي تريدها, بعد فترة زمنية معينة .

تقنية تحريك الرسوم المتحركة تستخدم الأسلوب **after()** هي الوحيدة الممكن تطبيق عملها على موضوع واحد, لأن الحلقة **mainloop()** تجوجه السلوك العام مثل هذا الطلب إلى ذلك . هذا خاصة سيتولى إعادة رسم جزء أو كل النافذةكلما كان ذلك ضروريا . لهذا السبب لا يمكنك تخيل بناء محرك رسوم متحركة التي من شأنها أن تعيد تحديد إحداثيات الكائن الرسومي داخل حلقة while بسيطة, على سبيل المثال, لأن في أثناء تشغيل **mainloop()** سوف تبقى معلقة, الأمر الذي يعني أنه خلال هذه الفترة الزمنية سوف يتم إعادة تصميم أي كائن رسومي (خاصة إذا كنت ترغب في الحصول على هذه الخطوة) . في الواقع, سوف يتم تجميد التطبيق, طالما لا يتم مقاطعة حلقة while .

هو الوحيد الممكن, هذه التقنية التي إستخدمناها حتى الأن في أمثلة تطبيق الموضوع-الواحد الخاص بنا . لديها عيب واحد مزعج : نظر لوجود عدد كبير من عمليات الدعم في كل تكرار لحلقة **() mainloop**, جهاز ضبط الوقت الذي يمكن برمجته بإستخدام **after()** لا يمكن أن يكون قصيرا جدا . على سبيل المثال, لا يمكنها أن تقع أقل من 15 مللي ثانية بالكاد على حاسوب نموذجي (سنة

2004, بروسيسور من نوع Pentium IV, f = 1,5 GHz) . يجب أن تنظر إلى هذا القيد إذا كنت ترغب في تطوير رسوم متحركة سريعة .

عيب أخر عند إستخدام الأسلوب **after()** يكمن في هيكل حلقة الرسوم المتحركة (أي دالة أو أي أسلوب "متكرر", هذا يعني أن يطلق على نفسه) : ليس من السهل في الواقع إتقان هذا النوع من البناء المنطقي, خاصة إذا كنت تريد تعيين رسوم متحركة من عدة كائنات رسم مستقلة, بما في ذلك عدد الحركات التي ينبغي أن تختلف مع مرور الوقت .

تأخير الرسوم المتحركة بإستخدام time.sleep()

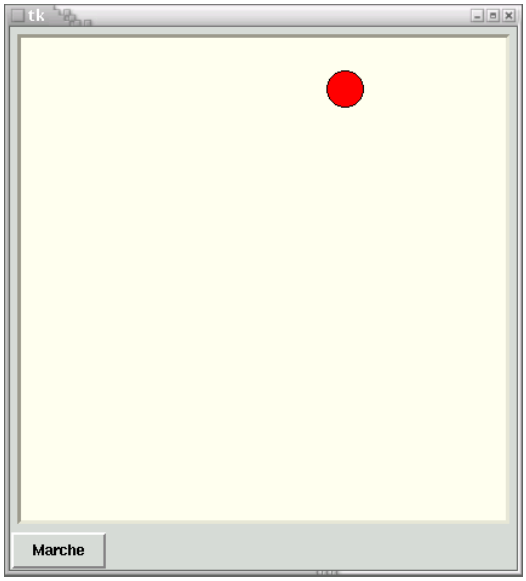
يمكنك تجاهل هذه القيود المفروضة على أسلوب **after()** المذكور أعلاه, إذا كنت تعطي رسوم المتحركة الخاصة بكائنات الرسومية للمواضيع المستقلة . بذلك, تقوم بتحرر وصاية **mainloop()**, و من ثم يسمح لك ببناء إجراءات رسوم متحركة على أساس هياكل حلقات أكثر "كلاسيكية", على سبيل المثال بإستخدام عبارة while أو عبارة for .

في قلب كل من هذه الحلقات, يجب عليك أن تكون دائما على يقن من أن إدراج تأخير أثناء "السير جانب" نظام التشغيل (حتى تتمكن من التعامل مع المواضيع الأخرى) . للقيام بذلك, سوف تستخدك الدالة **sleep()** من وحدة time . هذه الدالة تسمح لك بتعليق تنفيذ الموضوع الحالي لفترة معينة من الزمن, أثنائها المواضيع و التطبيقات الأخرى تستمر في العمل . إن التأخير المنتج على هذا النحو لا يعتمد على **mainloop()**, و بالتالي, يمكن أن يكون أقصر بكثير مما كنت تأذن أسلوب **after()** .

إنتباه : هذا لا يعني أن الشاشة سوف تجدد نفسها بشكل أسرع, لأن هذا التجديد لايزال يقدم من خلال **mainloop()** . يمكنك تسريعها إلى حد كبير من آليات المختلفة التي تثيتعا بنفسك في إجرائات الرسوم المتحركة الخاصة بك . على سبيل المثال, في برنامج لعبة, فإنه من الشائع أن تكون المقارنة بين مواقع إثنين من الجوالات بشكل دوري (مثل قذيفة و الهدف), من أجل إتخاذ إجراءات عند إنضمامهم (إنفجار, إضافة نقاط إلى النتيجة, إلخ) . بإستخدام تقنية الرسوم المتحركة التي تم شرحها هنا, يمكنك أن تفعل الكثير في الكثير من الأحيان من هذه المقارنات, و نتوقع بالتالي نتائج أكثر دقة . و بالمثل, يمكنك زيادة عدد النقاط التي تؤخذ بعين الإعتبار لحساب المسار في الوقت الحقيقي, و بالتالي فإنه يتحسن .

> تنبيه
>
> عندما تستخدم الأسلوب *after*(), يجب عليك أن تقوم بتحديد التأخير بالميلي ثانية, كبرامتر صحيح . عندما تستدعي الدالة *sleep*(), البرامتر التي تقوم بتمريرها يجب أن تكون بالثواني, بشكل عدد حقيقي (*float*) , و يمكنك إستخدام أعداد صغيرة جدا (على سبيل المثال : 0.0003) .

مثال ملموس



السكريبت الصغير المستنسخ أدناه يوضح تنفيذ هذه التقنية في مثال بسيط عمدا . هو تطبيق رسومي صغير فيه تتحرك في دائرة داخل اللوحة. يتم تشغيل "محركه" **mainloop()** كالعادة على الموضوع الرئيسي . منشئ تطبيق المثيل لوحة تحتوي على رسم دائرة, و زر و كائن موضوع . هذا كائن الموضوع هو رسوم متحركة للرسم, لكن دون اللجوء إلى إستدعاء الأسلوب **after()** للويدجت . فهو يستخدم حلقة بسطة while كلاسيكية جدا, مثبة في أسلوبه **run()** .

```
1#    from tkinter import *
2#    from math import sin, cos
3#    import time, threading
4#
5#    class App(Frame):
6#      def __init__(self):
7#        Frame.__init__(self)
8#        self.pack()
9#        can =Canvas(self, width =400, height =400,
10#                bg ='ivory', bd =3, relief =SUNKEN)
11#       can.pack(padx =5, pady =5)
12#       cercle = can.create_oval(185, 355, 215, 385, fill ='red')
13#       tb = Thread_balle(can, cercle)
14#       Button(self, text ='Marche', command =tb.start).pack(side =LEFT)
15#       #Button(self, text ='Arrêt', command =tb.stop).pack(side =RIGHT)
16#       #: إيقاف الخيط الاخر إذا قمنا بإغلاق النافذة
17#       self.bind('<Destroy>', tb.stop)
18#
19#   class Thread_balle(threading.Thread):
20#     def __init__(self, canevas, dessin):
21#       threading.Thread.__init__(self)
22#       self.can, self.dessin = canevas, dessin
23#       self.anim =1
24#
25#     def run(self):
26#       a = 0.0
27#       while self.anim == 1:
28#         a += .01
29#         x, y = 200 + 170*sin(a), 200 +170*cos(a)
30#         self.can.coords(self.dessin, x-15, y-15, x+15, y+15)
31#         time.sleep(0.010)
32#
33#     def stop(self, evt =0):
34#       self.anim =0
35#
36#   App().mainloop()
```

تعليقات

* السطران 13 و 14 : لتبسيط المثال إلى الحد الأقصى, قمنا بإنشاء كائن موضوع المسؤول عن حركة الرسوم المتحركة, مباشرة في منشئ التطبيق الرئيسي . هذا كائن الموضوع سوف يتم تشغيله عندما يقوم المستخدم بالضغط على **<Marche>**, التي تقوم بتفعيل أسلوبه **start()** (تذكر هنا أن هذا الأسلوب المدمج الذي سيتم تشغيله هو نفس **run()** حين أنشأنا حلقة الرسوم المتحركة الخاصة بنا) .

* السطر 15 : لا يمكن إعادة تشغيل موضوع إنتهى . و لذلك, لا يمكنك تشغيل هذه الرسوم المتحركة سوى مرة واحدة فقط (على الأقل في شكل مقدمة هنا) . لإقناعك لذلك, فعل السطر 15 عن طريق إزالة الرمز **#** في البداية (و الذي تعتبره بايثون مجرد تعليق) : عند بدء الرسوم

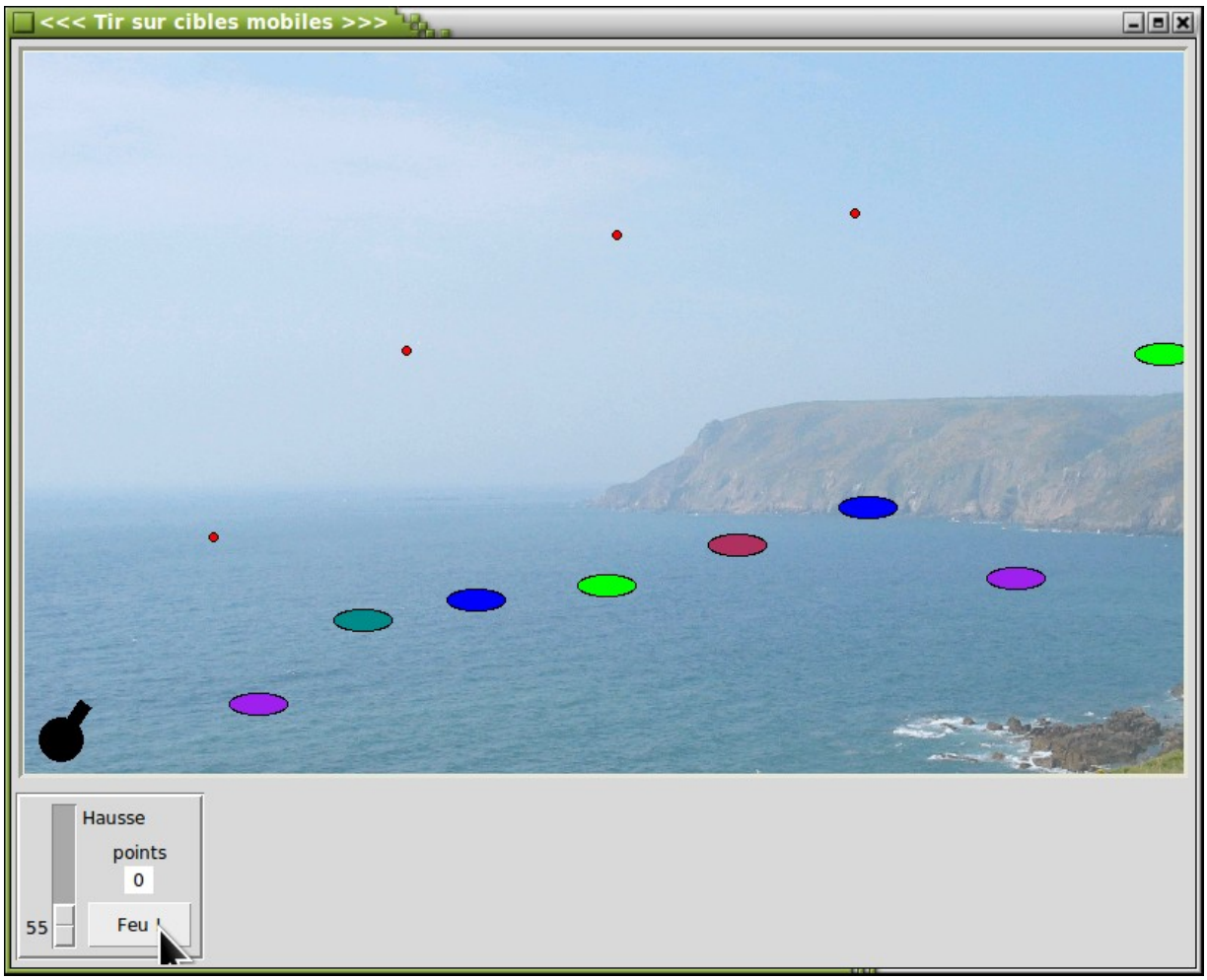
المتحركة, النقر بواسطة زر الفأرة يتسبب في إستدعاء الخروج من حلقة while في الأسطر من 27 إلى 31, سوف ينهيها الأسلوب **run()** . ستتوقف الرسوم المتحركة, لكن الموضوع سوف ينتهي أيضا . إذا حاولت إادة تشغيله بإستخدام الزر **<Marche>**, لن تحصل سوى على رسالة خطأ .

* الأسطر من 26 إلى 31 : لمحاكات حركة دائربة موحدة, يكفي أن تغير بإستمرار قيمة الزاوية a , الجيب و الجيب التمام لهذه الزاوية يمكنك حساب إحداثيات **x** و **y** من نقطة محيط الدائرة التي تتطابق مع هذه الزاوية[104].

في كل تكرار, سوف تتغير الزاوية بمئة راديان فقط (حوالي 0.6 درجة), و سوف يتطلب ذلك 638 تكرار ليقوم بدورة كاملة . التوقيت الذي تم إختياره لهذه التكرارات هو في السطر 31 : 10 ميلي ثانية . يمكنك تسريع هذه العملية عن طريق التقليل هذه القيمة, و لكنك لا يمكنك أن تقلل أقل من 1 ميلي ثانية (0.001 ثانية), و التي هي ليست سيئة للغاية .

> تذكير
>
> يمكنك الحصزل على شفرة جميع الأمثلة من موقع :
>
> *http://inforef.be/swi/python.htm*
>
> سوف تجد في ملف يسمى *cibles_multiples.py*, لعبة صغيرة التي يجب على السمتخدم إطلاق النار على مجموعة من الأهداف المتحركة التي تزداد أعدادها و سرعتها مع مرور الوقت . و هذه اللعبة تستخدم تقنيات الرسوم المتحركة الموضحة أعلاه .

104يمكنك أن تجد بعض التفسيرات حول هذا الموضوع في صفحة Error: Reference source not found.


<<< Tir sur cibles mobiles >>>
Hausse
points
0
55
Feu !

الملحق أ

تثبيت البايثون

إذا أردت تجربة البايثون على حاسوبك الخاص, لا تتردد : عملية التثبيت سهلة للغاية, بدون أي خطر على حاسوبك .

في نظام تشغيل ويندوز

على الموقع الرسمي للبايثون : **http://www.python.org** سوف تجد قسم تحميل برامج التثبيت التلقائي لمختلف إصدارات البايثون . يمكنك إختيار الإختيار أخر "منتج" .

على سبيل المثال, في 2 ديسمبر (كانون الأول) 2011, تم إصدار الإصدار 3.2.2 . ملف التثبيت يسمى Python 3.2.2 Windows x86 MSI Installer (أو النسخة المطابقة لأنظمة 64 بت) . تتضمن هذه الإصدارة على المكتبة الرسومية tkinter .

أنسخ هذا الملف إلى دليل مؤقت على جهاز الحاسوب الخاص بك. و قم بتشغيله . البايثون 3 سيتم تثبيته إفتراضيا في دليل بإسم **Python** (حيث ** أول رقمين من رقم الإصدار), و سيتم تحديث أيقونات البايثون تلقائيا .

إذا كنت تريد الإستفادة من موارد مكتبات الطرف ثالث التي لم تتوفر بعد للبايثون 3, مثل ReportLab أو Python Imaging Library, يمكنك تثبيت مثبت الإصدار الأخير من البايثون 2, بتحميل الملف Python 2.7.2 Windows Installer (أو النسخة المخصصة لأجهزة 64بت) . يتم تثبيت النسختين في دلائل مختلفة دون أن يعوق أحدهم الأخر بأي شكل من الأشكال .

عند إكتمال التثبيت, يمكنك حذف الدليل المؤقت .

في نظام تشغيل لينكس

ربما قد قمت بتثبيت نظام تشغيل لينكي بمساعدة إحدى التوزيعات مثل أبنتو, سوزي, ريدهات ... قم ببساطة بتثبيت البايثون و التي هي جزء منه, و لا تستغنى عن tkinter (أيضا قم بتثبيته في نفس الوقت مع مكتبة Python Imaging Library) . إذا قمت بتثبيت البايثون 2 أو البايثون 3, يجب عليك أيضا تثبيت إصدارين مختلفين من tkinter .

في نظام تشغيل ماك

على الموقع الرسمي للبايثون, سوف تجد مجموعة تركيب لنظام ماك مثل التي لدى نظام تشغيل ويندوز .

تثبيت *Cherrypy*

مكتبة Cherrypy هي منتج مستقل له موقعه الرسمي على الأنترنات :

قم بزيارة قسم التحميل : **http://cherrypy.org**

/http://download.cherrypy.org/cherrypy/3.2.2

في هذا الدليل, سوف تجد روابط تحميل الإصدار الحالي من Cherrypy . في وقت كتابة هذه السطور (02\12\2011), توجد نسخة 3.2.2 (هذا الرقم ليس له أي علاقة بإصدارات البايثون) . الملفات تتضمن نسخ Cherrypy للبايثون 2 و البايثون 3 في نفس المجموعة .

- إذا كنت تستخدم نظام تشغيل ويندوز, يجب عليك تحميل الملف CherryPy-3.2.2.win32.exe, ثم قم بتشغيل الملف الذي قمت بتحميله (المثبت التلقائي) . إذا كنت قد ثبت نسختين بايثون أي الإصدار 2 و 3, فيجب عليك في هذه الحالة إعادة تشغيل المثبت مرتين لكي يثبت المكتبات في كلا الإصدارين .
- إذا كنت تعمل على نظام تشغيل لينكس أو أي نظام تشغيل أخر, فيجب عليك نقل ملف الأرشيف الذي حملته (CherryPy-3.2.2.tar.gz أو CherryPy-3.2.2.zip) إلى أي دليل مؤقت, ثم قم بفك ضغطه بمساعدة برنامج المناسب (unzip أو tar) . الفك يكشف عن دليل فرعي يسمى CherryPy-3.2.2 . قم بالدخول إلى هذا الدليل و أنت في وضع مستخدم الجذر (روت), و قم بتنفيذ هذا الأمر **python3 setup.py install** لتثبيت نسخة Cherrypy الخاصة بالبايثون 3, أو\و الأمر : **python setup.py install** لتثبيت نسخة Cherrypy الخاصة بالبايثون 2 .

تثبيت pg8000

مكتبة pg8000 هي واحدة من وحدات الواجهة التي تمكنك من الوصول إلى الخادم PostgreSQL من البايثون . و هي ليست الأكثر كفاءة, لكنها لديها ميزة أنها متاحة للبايثون 2 و 3 في وقت كتابة هذه الأسطر, و هذه الميزة لا توجد في كل المكتبات . بالإضافة إلى ذلك, هذه الوحدة مكتوبة بالبايثون و لا تحتاج إلى أية مكتبة أخرى, بحيث تطبيقات البايثون التي تستخدمها تكون محمولة .

عندما تقرأ هذه الأسطر, إن الوحدات الأكثر كفاءة بالتأكيد متاحة, مثل psycopg2 . أرجو منك الإطلاع على مواقع الويب التي تتعامل مع تفاعل بايثون-PostgreSQL للمزيد إذا كنت ترغب في تطوير تطبيق مهم .

لتثبيت pg8000 على نظامك, قم بزيارة موقع **http://pybrary.net/pg8000/** , وقم بتحميل ملف المناسب لأحدث نسخة متاحة, و هي مخصصة للبايثون 2 أو البايثون 3 (يمكنك مرة أخرى تثبيت النسختين) . ملف التحميل هو نفسه, سواء أن كنت تعمل على ويندوز أو لينكس أو ماك أو أي نظام نشغيل أخر, و يمكنك تحميله على شكل **.zip** أو **.tar.gz** (على سبيل المثال pour-8000 py3-1.08.tar.gz للبايثون 3 أو pg8000-1.08.tar.gz للبايثون 2 في وقت كتابة هذه الأسطر) . ثم قم بنسخ ملف الأرشيف الذي حملته إلى أي دليل مؤقت, ثم قم بفك ضغطه بمساعدة أي برنامج مناسب (unzip أو tar) . عملية الفك تكشف عن دليل فرعي pg8000-1.08 أو pg8000-py3-1.08 . قم بالدخول إلى هذا الدليل كمدير, و أكتب الأمر **python3 setup.py install** لتثبيت pg8000 المخصصة للبايثون 3, وأو الأمر : **python setup.py install** لتثبيت النسخة المخصصة للبايثون 2 .

تثبيت ReportLab و Python Imaging Library

في وقت كتابة هذه السطور, هذه المكتبة غير متاحة للأسف لبايثون 3 (أنظر للفصل 18 للإطلاع على المناقشة لهذه المشكلة) . لذا يجب عليك تثبيت واحدة من الإصدارات المناسبة للبايثون 2.6 و البايثون 2.7 .

لتثبيت ReportLab :

- في لينكس, يكفي أن تقوم بتثبيت حزمة **python-reportlab** المناسبة لتوزيعتك (أبنتو, ديبان ...) . للحصول على أخر إصدار, قم بالحصول عليها من شبكة الإنترنات, حزم ReportLab موجودة على العنوان التالي : **http://www.reportlab.com/ftp** .

- في نظام تشغيل ويندوز, إعتماد على نسخة البايثون التي لديك (2.6 أو 2.7), قم بتحميل الملف **reportlab-2.5.win32-py2.6.exe** أو **reportlab-2.5.win32-py2.7.exe**, ثم قم بتشغيله (التثبيت التلقائي) .
- بالنسبة للأنظمة التشغيل الأخرى, قم بتحميل إحدى ملفات الأرشيف **reportlab-2.5.tar.gz** أو **reportlab-2.5.zip**, و ثم بفك ضغطها في دليل مؤقت . سيتم إنشاء دليل فرعي تلقائيا, قم بتنفيذ هذا الأمر **python setup.py install** و أنت مدير .

لتثبيت Python Imaging :

- في لينكس, يكفي أن تقوم بتثبيت حزمة **python-imaging** المناسبة لتوزيعتك (أبنتو, ديبان ...) . للحصول على أخر إصدار, قم بالحصول عليها من شبكة الإنترنات, حزم Python Imaging Library موجودة على العنوان التالي : **/http://www.pythonware.com/products/pil** .
- في نظام تشغيل ويندوز, قم بتحميل الملف **Python Imaging Library 1.1.7** للبايثون 2.6 أو للبايثون 2.7, ثم قم بتشغيله (التثبيت التلقائي) .
- بالنسبة للأنظمة التشغيل الأخرى, قم بتحميل **Python Imaging Library** , و ثم بفك ضغطها في دليل مؤقت . سيتم إنشاء دليل فرعي تلقائيا, قم بتنفيذ هذا الأمر **python setup.py install** و أنت مدير .