خطوة على طريق بيثون

ترجمـــه : أشرف علي خلف

راجعه : خالد حسني

Byte of Python

Swaroop C H

الإصدارة 1. 20

ملخص

سيساعدك هذا الكتاب على تعلم لغة البرمجة بيثون، سواء كنت جديدا على الحواسيب أو مبرمجا متمرسا.

قائمة المحتويات

7. الدوال

8. الوحدات

9. هياكل البيانات

تمهيـــــد

قائمة المحتويات

بيثون هي واحدة من تلك اللغات القليلة التي يمكننا الادعاء أنها تجمع بين البساطة والقوة. إنها لغة جيدة للمبتدئين وللمحترفين على حد سواء، والأمر الأهم المتعة في البرمجة بها. يهدف الكتاب إلى مساعدتك في تعلم هذه اللغة الرائعه ويريك كيفية إنجاز الأمور بسرعة وسهولة – وفى الواقع هو 'الترياق المثالي لمشاكلك البرمجية'.

لمن هذا الكتاب؟

هذا الكتاب بمثابة دليل تعليمي للغة البرمجة بيثون. وهي تستهدف أساسا المبتدئين. وهي مفيدة للمبرمجين ذوي الخبرة كذلك.

والهدف من ذلك عموما إذا كان كل ما تعرفه عن الحواسيب هو كيفية حفظ الملفات النصية فيمكنك إذن أن تتعلم بيثون من هذا الكتاب. وإذا كان لديك خبرة مسبقة عن البرمجة فيمكنك أيضا أن تتعلم بيثون من هذا الكتاب.

إذا كان لك خبرة مسبقة بالبرمجة، فستكون مهتما بأوجه الاختلاف بين بيثون ولغة برمجتك المفضلة - لقد ألقيت الضوء على الكثير من هذه الاختلافات. على الرغم من ذلك فلي تنبيه بسيط، بيثون عما قريب سوف تصبح لغة برمجتك المفضلة.

تاريخ الكتـاب

في أول الأمر بدأت مع بيثون عندما احتجت إلى برنامج تثبيت لبرمجيتي Diamond، حتى أجعل التثبيت سهلا. وكان علي الاختيار بين ارتباطات بيثون و بيرل مع مكتبة QT وبحثت في شبكة الإنترنت حتى عثرت على مقالة لإريك إس. رايموند - ذلك الهاكر المبجل والمشهور - يتكلم فيها عن كيف أصبحت بيثون هي لغة برمجته المفضلة. وكذلك اكتشفت أن ارتباطات PyQt جيدة جدا بالمقارنة مع Perl-Qt. لذلك قررت أن بيثون هي لغتي.

بعدها بدأت البحث عن كتاب جيد في لغة بيثون. ولكني لم أجد أيا منها!!، وقد وجدت بعض كتب O'Reilly ولكنها كانت إما باهظة الثمن للغاية، أو أقرب إلى الكتيب المرجعي منها إلى دليل. وبالتالي اتجهت إلى الوثائق التي جاءت مع بيثون، لكنها كانت مختصرة جدا وصغيرة، وقد أعطتني فكرة جيدة عن بيثون، ولكنها لم تكن مكتملة. وقد أمكنني التعامل معها حيث كان لدي خبرة مسبقة بالبرمجة، ولكنها غير ملائمة للمبتدئين.

بعد ستة أشهر من أول لقاء لي مع بيثون قمت بتثبيت آخر توزيعة (في وقتها) من ردهات Red Hat 9، وبدأت ألعب ببرنامج KWord، وكنت أزداد إثارة وفجأة خطرت لي فكرة كتابة مادة عن بيثون. وقد بدأت الكتابة بقليل من الصفحات، ولكنها سريعا أصبحت ثلاثين صفحة طويلة، بعدها صبحت جادا في جعلها أكثر فائدة على شكل كتاب. وبعد *الكثير* من إعادة الكتابة، أصبح في مرحلة كونه مرجعا مفيدا في تعلم لغة بيثون، وأنا أعتبر هذا الكتاب مساهمتي وتحيتي لمجتمع المصادر المفتوحة.

بدأ هذا الكتاب كملاحظات شخصية عن بيثون وما زلت أنظر إليه نفس النظرة، بالرغم من أني بذلت فيه الكثير من الجهد ليكون أكثر قبولا عند الآخرين :)

ومن خلال الروح الحقيقية لمجتمع المصادر المفتوحة، تلقيت الكثير من الاقتراحات البناءة، والانتقادات وردود فعل متحمسة من القراء مما ساعدني كثيرا على تحسين هذا الكتاب.

حالة الكتـاب

ما يزال هذا الكتاب **قيد العمل**. حيث الكثير من الفصول تتغير باستمرار وتتحسن، ومع ذلك فالكتاب قد نضج كثيرا، وستكون مستعدا لتعلم بيثون بسهولة من هذا الكتاب. من فضلك أخبرني إن وجدت أي جزء في هذا الكتاب غير صحيح أو غير مفهوم.

هناك خطط لمزيد من الفصول في المستقبل، مثلا فصل عن wxPython Twisted وربما حتى Boa Constructor.

الموقع الرسمي

الموقع الرسمي لهذا الكتاب هو http://www.byteofpython.info ومن خلال الموقع يمكنك

قراءة الكتاب كاملا بشكل مباشر أو تنزيل آخر إصدار من الكتاب، وكذلك إرسال الملاحظات لي.

بنود الترخيص

الاقتراحات

لقد بذلت الكثير من الجهد لجعل هذا الكتاب مفيدا ومحكما على قدر الإمكان. ولكن، إذا وجدت بعض المواد غير متسقة أو غير صحيحة، أو ببساطة بحاجة إلى تحسين، رجاء أبلغني، بحيث أتمكن من عمل الإصلاحات المناسبة. يمكنك الوصول إليّ عبر <swaroop@byteofpython.info>.

مسائل يجب التفكير فيها

هناك طريقتان لتصميم برمجية أحدها هو جعلها بسيطة جدا بحيث يظهر بوضوح أنها بلا قصور، وإما جعلها معقدة بحيث لا يظهر بها أوجه القصور.

C. A. R. Hoare--

النجاح في الحياة ليس مسألة ذكاء وموهبة بقدر ما هو تركيز ومثابرة.

C. W. Wendte--

فصل 1.مقدمـــة

قائمة المحتويات

مقدمـــة

بيثون هي واحدة من تلك اللغات القليلة التي يمكن الادعاء أنها **بسيطة** و**قوية** على حد سواء. ستتفاجأ كم هي سهلة وتركز على حل المشكلة وليس تراكيب وأساسيات لغة البرمجة التي تبرمج بها.

المقدمة الرسمية لبايثون هي

بيثون هي لغة برمجة سهلة التعلم، قوية. لها هيكلة بيانات عالية المستوى كفؤة، وتوجه بسيط لكن فعال نحو البرمجية الكائنية. حذاقة قواعد بيثون وديناميه تحديد الأنواع، جنبا إلى جنب مع طبيعتها التفسيرية، تجعل من بيثون لغة مثالية لبرمجة السكربتات وتطوير التطبيقات السريع في العديد من المجالات على معظم المنصات.

سأناقش معظم هذه الميزات بمزيد من التفصيل في الباب التالي.

ملاحظة

أطلق جويدو ڤان روسام (Guido van Rossum) -مؤلف لغة بيثون- عليها هذا الاسم على اسم عرض هيئة الإذاعة البريطانية "سيرك مونتي للثعابين الطائرة (Monty Python's Flying Circus)". فهو ليس معجبا بالثعابين التي تقتل الحيوانات لتتغذى عليها عن طريق تصفية جسدها بالالتفاف حولها ثم سحقها.

مميزات بيثون

بسيطة

بيثون لغة بسيطة لأبعد الحدود. قراءة برنامج بيثون جيد يكاد يشبه قراءة اللغة الإنكليزية على الرغم من أنها إنجليزية صارمة!هذه الطبيعة الشبه رمزية (pseudo-code) لبايثون أحد

أعظم أسرار قوتها. فتتيح لك التركيز على حل المشكلة لا اللغة نفسها.

سهلة التعلم

كما سترون، بيثون سهلة للغاية لتبدأ بها في تعلم البرمجة. بيثون تحتوي تراكيب سهلة بشكل غير معتاد، كما سبق ذكره.

حرة ومفتوحة المصدر

بيثون هي مثال على البرمجيات الحرة مفتوحة المصدر. بعبارات بسيطة، يمكنك بحرية توزيع نسخ من هذه البرمجيات، وقراءة كود المصدر، و القيام ببعض التغييرات عليها واستخدام أجزاء منها في برمجيات حرة جديدة، وأنت تعرف أنه يمكنك أن تفعل هذه الأشياء. البرمجيات الحرية تقوم على مبدأ المجتمع الذي يتشارك في المعرفة. هذا واحد من أسباب كون بيثون جيدة جدا - لأنه قد تم إنشاؤها وتحسينها بشكل مستمر من خلال المجتمع الذي يريد فقط أن يرى بيثون أفضل.

لغة برمجة عالية المستوى

عندما تكتب البرامج في بيثون، لا تحتاج للاهتمام بالتفاصيل دقيقة المستوى مثل إدارة الذاكرة التي يستخدمها برنامجك، إلخ.

محمولة

نظرا لطبيعتها كبرمجية مفتوحة المصدر، تم نقل بيثون إلى (أي تم جعلها تعمل على) العديد من المنصات. كل ما تكتبه من برامج بيثون يمكن أن يعمل على أي من هذه المنصات دون أن يتطلب ذلك أي تغييرات على الإطلاق إذا كنت دقيقا بما فيه الكفاية لتجنب أي خصائص تعتمد على نظام بعينه.

يمكنك استخدام بيثون على لينكس، ويندوز، فري بي‌إس‌دي، ماكينتوش، سولاريس، OS/2، Amiga، AROS، AS/400، BeOS، OS/390، z/OS، Palm OS، QNX، VMS، Psion، Acorn RISC OS، VxWorks، PlayStation، Sharp Zaurus، Windows CE وحتى الحاسوب الكفي.

مفسَّرة

هذا يتطلب شيئا من الشرح.

البرنامج المكتوب بلغة مصرّفة (compiled) مثل سي أو سي++ يتم تحويله من اللغة المصدر (سي أو سي++) إلى اللغة التي يتكلمها حاسوبك (كود ثنائي من أصفار وآحاد) باستخدام المصرّف مع مختلف الخيارات والتعليمات. عند تشغيلك البرنامج، يقوم الرابط/المحمّل (linker/loader) بنسخ البرنامج من القرص الصلب إلى الذاكرة ويبدأ في تشغيله.

بيثون -من ناحية أخرى- لا تحتاج التصريف إلى كود ثنائي. فقط *شغل* البرنامج مباشرة من الكود المصدر. داخليا، فإن بيثون يحول كود المصدر إلى شكل وسيط يسمى bytecode ثم يترجم هذا إلى اللغة الأصلية لجهازك، ثم يشغله. كل هذا يجعل من الأسهل بكثير استخدام بيثون حيث لست بحاجة للاهتمام بتصريف البرنامج، أو التأكد من صحة مكتبات الربط وتحميلها، الخ، الخ. وهذا أيضا يجعل برامج بيثون الخاصة بك أكثر محمولية، بحيث يمكنك مجرد نسخ برنامج بيثون الخاص بك إلى حاسوب آخر، وبعدها يعمل!

كائنية التوجه

تدعم بيثون البرمجة الإجرائية (procedure-oriented) وكذلك البرمجة الكائنية (object-oriented). في اللغات *إجرائية التوجه*، يتمحور البرنامج حول الإجراءات أو الدوال التي ليست سوى قطع من البرامج يمكن إعادة استخدامها. وفي اللغات *كائنية التوجه*، يتمحور البرنامج حول الكائنات (objects) التي تجمع بين البيانات والوظائف. ولبايثون طريقة قوية جدا ولكن تبسيطية لعمل البرمجة الكائنية خاصة عند مقارنتها باللغات الكبيرة مثل سي++ أو جافا.

قابلة للامتداد

إذا كنت في حاجة لجعل جزء حيوي من الكود يعمل سريعا جدا أو تريد إخفاء بعض الخوازميات، فيمكنك كتابة هذا الجزء من برنامجك بلغة سي أو سي++ وبعدها تستخدمه من برنامج بيثون الخاص بك.

قابلة للتضمين

يمكنك تضمين بيثون في برامج سي/سي++ لإعطاء قدرات ال'scripting' لمستخدمي برنامجك.

مكتبات شاملة

مكتبة بيثون القياسية مكتبة ضخمة حقا. تساعدك على عمل مختلف الأشياء العادية بما فيها

التعابير النمطية (regular expressions)، توليد التوثيق، اختبار الوحدات، الخيوط (threading)، قواعد البيانات، متصفحات وب، ftp، CGI، بريد إلكتروني، XML-RPC، HTML، ملفات WAV، التعمية، الواجهات الرسومية وغيرها من الأشياء التي تعتمد على النظام. تذكر، كل هذا متاح دائما أينما يثبت بيثون. وهذا ما يسمى فلسفة 'البطاريات مضمنة' في بيثون.

بجانب المكتبات القياسية توجد العديد من المكتبات المتنوعة الأخرى عالية الجودة مثل wxPython، و Twisted، و Python Imaging Library وغيرها الكثير.

الخلاصة

بيثون لغة مثيرة وقوية حقا. فهي مزيج من حسن الأداء والميزات التي تجعل كتابة برامج بيثون خليطا من السهولة والمتعة.

لماذا ليس بيرل؟

إذا كنت لا تعرف بعد، فبيرل تعتبر هي الأخرى لغة برمجة مفسرة مفتوحة المصدر شعبية للغاية.

إذا سبق لك وحاولت كتابة برنامج كبير في بيرل، ربما كنت قد أجبت عن هذا السؤال بنفسك! بعبارة أخرى، فإن برامج بيرل سهلة عندما تكون صغيرة، وهو يبرع في البرامج الصغيرة والسكربتات والهاكات لإنجاز العمل. لكن سرعان ما تصبح هذه البرامج صعبة المراس بمجرد البدء في كتابة برامج أكبر، وأنا أتحدث من واقع تجربة كتابة برامج كبيرة بلغة بيرل في ياهو!

وبالمقارنة مع بيرل، فإن البرامج على بيثون هي بالتأكيد أبسط، وأوضح، وأسهل في الكتابة وبالتالي أكثر قابلية للفهم وللصيانة. أنا معجب بلغة بيرل وأقوم باستخدامها بشكل أساسي يوميا لأمور متنوعة، ولكني كلما كتبت برنامجا، أبدأ التفكير في بيثون؛ حيث إنها أصبحت طبيعية جدا بالنسبة لي. خضعت لغة بيرل لعدد كبير من التغييرات والهاكات، وتشعر أنها تبدو كهاك كبير. ومن المحزن أن إصدارة بيرل 6 المقبلة لا يبدو أنها قامت بأي تحسينات تتعلق بهذا.

الميزة الوحيد الهامة جدا والتي أشعر بها في بيرل هي المكتبة الضخمة شبكة أرشيف بيرل الشاملة (CPAN, the Comprehensive Perl Archive Network)وكما يوحي الاسم، هذا جمع مزيج من وحدات (modules) بيرل وهو ببساطة مذهل للعقل نظرا للحجم والعمق؛ يمكنك عمليا القيام بأي شيء يمكنك القيام به مع الحاسوب باستخدام هذه الوحدات. أحد الأسباب التي تجعل مكتبات بيرل أكثر من بيثون كونها أقدم من بيثون بكثير. ولعلي

أقترح ورشة هاك لنقل وحدات بيرل إلى بيثون في comp.lang.python :)

كذلك؛ الآلة الافتراضية الجديدة Parrot مصممة لتشغيل كل من لغة بيرل 6 المعاد تصميمها تماما بالإضافة لبايثون واللغات المفسرة الأخرى مثل Ruby و PHP و Tcl. ما يعنيه هذا بالنسبة لك أنك *ربما* تكون قادرا على استخدام جميع وحدات بيرل من داخل بيثون في المستقبل، ولذا سيمنحك ذلك الأفضل في كل من أقوى مكتبة في العالم CPAN بالاشتراك مع لغة بيثون القوية. على أية حال؛ علينا فقط أن ننتظر ونرى ما سيحدث.

ماذا يقول المبرمجون

ربما من المهم أن تقرأ ما يقوله عظماء الهاكر من أمثال إريك س. رايموند.

- **إريك س. رايموند** مؤلف كتاب 'الكاتدرائية والسوق' (The Cathedral and the Bazaar) وهو أيضا واضع مصطلح 'المصادر المفتوحة'. يقول لقد أصبحت بيثون هي لغة برمجته المفضلة. وتعتبر هذه المقالة هي الملهم الحقيقي لي في أولى خطواتي مع بيثون.
- **بروس إيكل** مؤلف الكتب الشهيرة 'التفكير بلغة جافا' و 'التفكير بلغة سي++' يقول أن ليس هناك لغة قد جعلته أكثر إنتاجية من بيثون. ويقول أن بيثون ربما تكون اللغة الوحيدة التي تركز على جعل الأمور أسهل بالنسبة للمبرمج. اقرأ المقابلة الكاملة لمزيد من التفاصيل.
- **بيتر نورفج** مؤلف ليسب شهير ومدير جودة البحث في جوجل (شكرا لجويدو ڤان روسام لإشارته إلى ذلك) يقول أن بيثون كانت دائما جزءا أصيلا من جوجل. يمكنك التحقق من هذا التصريح في الواقع من خلال النظر في صفحة وظائف جوجل. والتي وضعت لغة بيثون في قائمة المعارف المطلوبة من مهندسي البرمجيات.
- **بروس بيرينز** أحد مؤسسي OpenSource.org ومشروع UserLinux. يهدف Userlinux لعمل توزيعة قياسية من لينكس مدعومة من منتجين متعددين. وقد ضربت بيثون المتنافسين مثل بيرل وروبي لتصبح لغة البرمجة الرئيسية التي ستكون مدعومة من قبل Userlinux.

فصل 2. تثبيت بيثون

قائمة المحتويات

لمستخدمي لينكس/بي‌إس‌دي

إذا كنت تَستعملُ توزيعة لينكس مثل فيدورا أَو ماندريك أَو {ضع اختيارَكَ هنا}، أَو نظام بي‌إس‌دي مثل فري بي‌إس‌دي، فغالبا بيثون مثبتة على نظامك.

لاختبار ما إذا كانت بيثون موجودة بالفعل عندك على توزيعة لينكس، افتح برنامج طرفية (مثل konsole أَو gnome-terminal) وأدخل الأمر python -V كما موضح بأسفل.

```
$ python -V
Python 2. 3. 4
```

ملاحظـــة

$ هي علامة المحث في صدفة النظام وذلك قد تختلف بالنسبة لك اعتمادا على إعدادات نظامك. لذا سوف أشير إلى المحث بهذا الرمز فقط $.

إذا رأيت بعض المعلومات عن الإصدارة مثل المعروض أعلاه، فبايثون مثبتة بالفعل.

ولكن إذا حصلت على رسالة مثل هذه:

```
$ python -V
          bash: python: command not found
```

فبايثون ليست مثبتة عندك. هذا من المستبعد جداً لكنه ممكن.

في هذه الحالة، هناك طريقتان لتثبيت بيثون على نظامك :

- تثبيت حزم ثنائية باستخدام برامجَ إدارةِ الحزم التي تَجيءُ مع النظام، مثل yum في لينكس فيدورا، و urpmi في لينكس ماندراك، و apt-get في لينكس دبيان، و pkg_add في نظام FreeBSD، إلخ. مُلاحظة: ستحتاج اتصالاً بالإنترنت لاستعمال هذه الطريقة.

أوبديلاً عن ذلك، يُمْكِنُكَ أَنْ تُحمّلَ الحزم الثنائية مِنْ مكان آخر وبعد ذلك انسخها إلى جهازك وقم بتثبيتها.

- يُمْكِنُ أَنْ تصرّف بيثون من الكود المصدري وتثبتها. تعليمات التصريف موجودة في موقع بيثون.

لمستخدمي ويندوز

اذهب إلى Python.org/download ونزّل آخر إصدارة بيثون مِنْ هذا الموقع. يبلغ حجمها 9. 4 م. بايت فقط وهي مضغوطةُ جداً بالمقارنة مع أكثر لغات البرمجة الأخرى. و التثبيت مثل أيّ برامج على بيئة ويندوز.

تحذير

عندما تَعطي خياراً بعدم تثبيت أيّ مكوّنات *اختيارية*، لا ترفع العلامة عن أيها ستكون بعض هذه المكوّناتِ مفيدا لَك، خصوصاً IDLE.

الحقيقة المثيرة أنّ حوالي 70% ممن قام بتحميل برامجِ بيثون مِن مستعملي ويندوز. بالطبع، هذه لا تَعطي صورةَ كاملةَ حيث أن كُلّ مستعملي لينكس تقريبا سَيكونُ عِنْدَهُمْ بيثون مثبتا على أنظمتِهم بشكل مبدئي.

استخدام بيثون في سطر أوامر ويندوز

إذا أردت أن تَكُونَ قادرا على اسْتِعْمال بيثون مِنْ سطر أوامر ويندوز، فإنك تَحتاجُ لضبط المتغير PATH بشكل صحيح.

بالنسبة لويندوز 2000K و XP و 2003، انقر على Control Panel -> System -> Advanced -> Environment Variables. اضغط على المتغيّرِ المسمّى **PATH** في قسم 'System Variables'، ثم اختر Edit وأضف **;C:\Python23** (بدون علامات التنصيص) في نهاية ما هو مكتوب هناك. بالطبع استخدم اسم المجلد المناسب.

في الإصدارات الأقدمِ مِنْ نظام ويندوز، أضف السطر **'PATH=%PATH%;C:\Python23'** في الملف C:\AUTOEXEC.BAT (بدون علامات التنصيص). بالنسبة لويندوز NT، استخدم الملف AUTOEXEC.NT.

الخلاصة

بالنسبة لنظام لينكس غالبا بيثون مثبّتة على نظامكَ. وإلا فيُمْكِنُكَ أَنْ تُثبتها باستخدام برامجَ إدارةِ الحزم التي تَجيءُ مع توزيعتك. بالنسبة لنظام ويندوز، يتم تثبيت بيثون بسهولة كذلك من خلال تَحميل ملف البرنامج وبالنقر مرتين عليه. ومن الآن فصاعداً، سنفترض أنّ بيثون مثبتة على نظامك.

الفصل القادم، سَنَكْتبُ برنامجَنا الأول على بيثون.

فصل 3. الخطوات الأولى

قائمة المحتويات

مقدمـــة

سَنَرى الآن كَيفَ نشغل البرنامج التقليدي 'Hello World' في بيثون. سَيُعلّمُك هذا كَيفَ تَكْتبُ، وتحفظ، وتشغّل برامجَ بيثون.

هناك طريقتين لاستخدام بيثون لتشغيل برنامجكَ؛ استخدام محث المفسر التفاعلي (من خلال أي طرفية) أو استخدام ملف مصدري. وسَنَرى الآن كَيفَ نَستعملُ كلتا الطريقتين.

استخدام المحث

ابدأ المفسر في سطر الأوامر بكتابة كلمة **python** في محث الصّدَفة. والآن اكتب print 'Hello World' متبوعة بزر **Enter**. يفترض أن ترى الكلمات Hello World كناتج.

لمستخدمي ويندوز، يُمْكنك أن تشغل المفسر من سطر الأوامر إذا ضبطت دليل المتغيّر PATH بشكل ملائم. أو بديلاً عن ذلك، يُمْكنُكَ أنْ تَستعملَ برنامجَ IDLE. IDLE عبارة عن بيئة تطوير متكاملة. انقر على Start -> Programs -> Python 2.3 -> IDLE (Python GUI). مستخدمي لينكس يمكنهم أن يستعملوا IDLE أيضاً.

لاحظ أن هذه العلامة >>> هي محث إدخال إفادات بيثون.

مثال 3.1.استعمال محث مفسر بيثون

```
$ python
Python 2.3.4 (#1, Oct 26 2004, 16:42:40)
```

```
[GCC 3.4.2 20041017 (Red Hat 3.4.2-6.fc3)] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>> print 'hello world'
hello world
>>>
```

لاحظْ أنّ بيثون تعطيك ناتجَ السطر فوراً. ما أدخلته هو *إفادة* بيثون واحدة. نستخدم print -ليست مفاجئة- لطبع أيّة قيمة نعطيها لها. هنا، أعطيناها النَصّ Hello World وقد طُبِع فوراً على الشاشة.

كيفية الخروج من محث بيثون

للخُروج من محث البرنامج اضغط على مفتاحي **Ctrl+d** إذا كنت تَستعملُ IDLE أَو تستخدم صددَفَة لينكس/بي‌إس‌دي. في حالة سطر أوامر ويندوز، اضغط مفتاح **Ctrl+z** متبوعة بمفتاح **Enter**.

اختيار محرر

قَبْلَ أَنْ نَنتقل لكتابة برامج بيثون في الملفات المصدرية، نَحتاجُ محرّراً لكتابَة الملفات المصدرية. اختيار المُحرّر أمر في غاية الأهمية. يَجِبُ أَنْ تختارَ المحررَ كما تَختارُ سيارة تَشتريها. المحرّر الجيد يُساعدُك على كتابة برامج بيثون بسهولة، ويَجْعلُ رحلتَك مريحةً أكثرَ ويُساعدُك لتَصلَ إلى غايتِكَ بطريقة أسرع بكثير وأكثر أماناً.

إحدى المتطلبات الأساسية جداً هي **إبراز التركيب** (syntax highlighting) حيث تلون أجزاء برنامج بيثون المختلفة حتى يتسنى لك أن تَرى برنامجَكَ وتَتصوّرُ كيفية عمله.

إذا كنت تَستعملُ ويندوز، أَقترحُ عليك استعمال IDLE. حيث إن IDLE يقوم بعمل بإبراز التركيب وأكثر مثل تمكينك من تشغيل برامجِكَ داخل IDLE ضمن أشياء أخرى. مُلاحظة خاصّة: **لا تستعملْ Notepad**، فهي اختيار سيئ لأنها لا تقوم بإبراز التركيب والأهم من ذلك أنه لا تَدْعمُ إزاحة النَصّ (indentation، أي تنظيم المسافات البادءة) وهو مهمٌ جداً في حالتنا كما سَنَرى لاحقاً. المحرّرات الجيدة مثل IDLE (وأيضاً VIM) ستساعدك آلياً على عملَ ذلك.

إذا كنت تَستعملُ لينكس/بي‌إس‌دي فلديك الكثير مِنْ المحرّرات لتختار منها. وإذا كنت مبرمجا خبيرا، فمن المؤكد أنك تستعمل VIM أو Emacs. ولا حاجة للقول بأنهما اثنان مِنْ المحرّرات الأقوى و سَتَستفيدُ من استعمالهما لكِتابَة برامج بيثون. أنا شخصياً أَستعملُ

VIM لأغلب برامجي. إذا كنت مبرمجا مبتدئا، حينئذ يُمْكِنُكَ أَنْ تَستعملَ Kate، وهي إحدى أدواتي المفضّلة. في حالة ما إذا كنت راغبا في قضاء وقت لتَعَلّم VIM أَو Emacs، فإني أَوصيك بشدة أن تَتعلّم استِعْمال أي منهما حيث سَيَكُونُ ذلك مفيدا جداً لَكَ في المدى البعيد.

إذا كنت ما تزال تُريدُ اسْتِكْشاف الخيارات الأخرى من المحررات، انظر القائمةَ الشاملةَ لمحرّرات بيثون وحدد خيارك. يُمْكِنُكَ أَنْ تَختارَ أيضاً بيئة تطوير متكاملة لبايثون، طالع القائمةَ الشاملةَ لبيئات التطوير التي تدعم بيثون للمزيد من التفاصيل.

أُكرّرُ مرةً أخرى، رجاءً اخترْ محرّرا جيدا، فهو يَجْعلَ كتابة برامج بيثون أكثرَ مرحاً وسهولة.

استخدام ملف مصدري

والآن لنعد إلى البرمجة. هناك تقليد أنه كلما تعلمت لغة برمجة جديدة، أول برنامج تكتبه وتشغله هو برنامج 'Hello World'؛ كل ما يفعله هو أن يقول 'Hello World' عند تشغيله.

شغل محررك المفضل، أدخل البرنامج التالي واحفظه باسم helloworld.py

مثال 3.2. استخدام ملف مصدري

```
#!/usr/bin/python
# Filename : helloworld.py
print 'Hello World'
```

(الملف المصدر:code/helloworld.py)

شغل هذا البرنامج عن طريق فتح الصدفة (طرفية لينكس أو محث دوس) ثم كتابة الأمر python helloworld.py. إذا كنت تستخدم IDLE فاستخدم قائمة Edit -> Run Script أو اختصار لوحة المفاتيح Ctrl+F5. الخرْج مبين أدناه.

الخرْج

```
$ python helloworld.py
Hello World
```

إذا حصلت على الناتج كما هو مبين أعلاه، فتهانينا لك، لقد نجحت في تشغيل أول برنامج لك في بيثون.

وفي حال حصلت على خطأ ما، يرجى كتابة البرنامج المذكور *بالضبط* كما هو مبين أعلاه وتشغيل البرنامج مرة أخرى. لاحظ أن بيثون حساسة لحالة الأحرف (التفريق بين الأحرف اللاتينية الكبيرة والصغيرة) مثلا كلمة print لا تساوي Print -لاحظ الحرف الصغير p في الكلمة الأولى و الحرف الكبير P في الأخيرة- أيضا، تأكد من عدم وجود مسافات قبل الحرف الأول في كل سطر -سنرى لماذا هذا أمر مهم في وقت لاحق-.

كيف يعمل

دعونا ننظر في أول سطرين من البرنامج. تسمى هذه *تعليقات* (comments)، أي شيء مكتوب على يمين الرمز # هو تعليق و هو في الأساس أمر مفيد لقارئ هذا البرنامج.

بيثون لا تستخدم التعليقات باستثناء السطر الأول هنا وهو حالة خاصة. وهي تسمى *سطر شابانغ* (shebang) عندما يكون أول حرفين من الملف المصدري عبارة عن #! متبوعا بمسار برنامج فإن هذا يخبر لينكس/يونكس أن هذا البرنامج يجب أن يعمل مع هذا المفسر (interpreter) عند *تنفيذه*. وسوف يُشرح هذا بالتفصيل في الفصل التالي. علما بأنه يمكنك دائما تشغيل البرنامج على أي منصة من خلال تحديد المفسر مباشرة على سطر الأوامر مثل الأمر **python** helloworld.py.

هام

استخدم التعليقات بحنكة في برنامجك لشرح بعض التفاصيل المهمة في البرنامج، وهذا أمر مفيد لقارئ برنامجك بحيث يمكن بسهولة فهم ما يقوم به البرنامج. تذكر، إن هذا الشخص يمكن أن يكون أنت نفسك بعد ستة أشهر!

التعليقات متبوعة *بإفادة* بيثون، تقوم فقط بطباعة النص 'Hello World'. كلمة print هي في الحقيقة عامل (operator) و 'Hello World' يشار إليها باعتبارها سلسلة (string)، لا تقلق، سنبحث أمر هذه المصطلحات بالتفصيل لاحقا.

برامج بيثون تنفيذية

لا ينطبق هذا إلا على مستخدمي لينكس/يونكس ولكن قد يكون مستخدمو ويندوز لديهم بعض الفضول عن السطر الأول من البرنامج. أولا، علينا إعطاء البرنامج صلاحية التنفيذ باستخدام الأمر **chmod** ثم *تشغيل* البرنامج المصدري.

```
$ chmod a+x helloworld.py
```

```
$ ./helloworld.py
Hello World
```

أمر chmod يستخدم هنا لتغيير حالة تصاريح الملف لجعله قابلا للتنفيذ باعطاء الصلاحية لجميع مستخدمي النظام. وبعدها فإننا ننفذ البرنامج مباشرة عن طريق تحديد موقع الملف المصدري. نستخدم . / لتشير إلى أن هذا البرنامج يقع في الدليل الحالي.

لجعل الأمور أكثر متعة، يمكنك فقط إعادة تسمية الملف إلى helloworld وتشغيله على النحو . **helloworld/** وسيعمل لأن النظام يعرف أن عليه تشغيل البرنامج باستخدام المفسر المحدد في السطر الأول من الملف المصدر.

أنت الآن قادر على تشغيل البرنامج ما دمت تعرف بالضبط مسار البرنامج، ولكن ماذا لو كنت تريد القدرة على تشغيل البرنامج من أي مكان؟ يمكنك أن تفعل ذلك من خلال تخزين البرنامج في واحدة من الأدله الواردة في مسار متغير البيئة PATH. كلما قمت بتشغيل أي برنامج، فإن النظام يبحث عن هذا البرنامج في كل من الأدلة الواردة في مسار متغير البيئة PATH ومن ثم يشغل هذا البرنامج. يمكننا أن نجعل هذا البرنامج متاحا في كل مكان وبكل بساطة نسخ هذا الملف المصدر إلى واحد من الأدله الواردة في المسار PATH.

```
$ echo $PATH
/
opt/mono/bin:/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/home/sw
aroop/bin
$ cp helloworld.py /home/swaroop/bin/helloworld
$ helloworld
Hello World
```

يمكنك عرض المتغير PATH باستخدام الأمر **echo** وإضافة $ قبل اسم المتغير لإخبار الصدفة أننا نريد قيمة هذا المتغير. ونحن نرى أن /home/swaroop/bin هو أحد الأدلة في المسار PATH حيث **swaroop** هو اسم المستخدم الذي أستخدمه على نظامي. سيكون هناك عادة دليل مماثل لاسم المستخدم الخاص بك على جهازك. أو كبديل لذلك، يمكنك أن تضيف دليلا من اختيارك للمتغير PATH، ويمكن أن يتم ذلك عن طريق كتابة الأمر

```
PATH=$PATH:/home/swaroop/mydir
```

حيث 'home/swaroop/mydir/' هو الدليل الذي أريد إضافته إلى المتغير PATH.

وهذه الطريقة مفيدة جدا إذا كنت تريد أن تكتب سكربتات مفيدة وتريد تشغيل البرنامج في أي وقت وفي أي مكان. إنه يشبه عمل أوامرك الخاصة مثلها مثل الأمر **cd** أوغيره من الأوامر

التي تستخدمها في طرفية لينكس أو دوس.

تحـذيــر

في بيثون برنامج أو سكريبت أو تطبيق جميعها تعني نفس الشيء.

الحصول على المساعدة

إذا كنت بحاجة إلى معلومات بشكل سريع عن أي دالة أو إفادة (statement) في بيثون، يمكنك استخدام وظيفة المساعدة المدمجة في البرنامج. هذا مفيد جدا وخصوصا عند استخدام محث المفسر. فعلى سبيل المثال شغل help(str) وسيعرض هذا مساعدة عن الصنف str والتي تستخدم لتخزين كل نص (سلسلة) تستخدمه في برنامجك. ستشرح الطبقات بالتفصيل في الفصل المتعلق بالبرمجة الكائنية.

ملاحظــة

اضغط q للخروج من المساعدة.

وبالمثل، يمكنك الحصول على معلومات عن أي شيء تقريبا في بيثون. استخدم help() لمعرفة المزيد حول استخدام المساعدة نفسها.

في حال كنت بحاجة إلى الحصول على مساعدة عن عامل مثل print، فأنت بحاجة إلى تحديد متغير البيئة PYTHONDOCS بشكل مناسب. ويمكن أن يتم ذلك بسهولة على لينكس/يونكس باستخدام الأمر **env**.

```
$ env PYTHONDOCS=/usr/share/doc/python-docs-2.3.4/html/ python
Python 2.3.4 (#1, Oct 26 2004, 16:42:40)
[GCC 3.4.2 20041017 (Red Hat 3.4.2-6.fc3)] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>> help('print')
```

ستلاحظ أني استخدمت علامة الاقتباس لتحديد 'print' حتى يمكن لبايثون أن تفهم أنني أريد استحضار مساعدة حول 'print' وأنني لا أطلب منه طباعة أي شيء. علما أنني استخدمت المكان المستخدم في فيدورا 3 وقد تكون مختلفة طبقا للتوزيعة أو الإصدارة.

الخلاصة

يجب أن تكون الآن قادرا على كتابة، وحفظ، وتشغيل برامج بيثون بكل سهولة. الآن أنت مستخدم بيثون، دعنا الآن نتعلم مفاهيم أكثر عن بيثون.

فصل 4. الأساسيات

قائمة المحتويات

مجرد طباعة 'Hello World' فقط لا يكفي، أليس كذلك؟ هل تريد أن تفعل أكثر من ذلك؛ تريد أخذ بعض المدخلات، و معالجتها والحصول على شيء منها. يمكننا أن نحقق هذا في بيثون باستخدام الثوابت والمتغيرات.

الثوابت الحرفية

من الأمثلة على الثابت الحرفي العدد 5، أو 1. 23، أو 9. 3-25e، أو سلسلة مثل 'هذه سلسلة' أو "It's a string!". وهذه تسمى حرفية (literal) لأنها حرفية، وأنت تستخدم قيمتها الحرفية. الرقم 2 يمثل نفسه دائما ولا شيء آخر؛ فهو ثابت (constant) لأن قيمته لا يمكن أن تتغير. لذلك، فهذه كلها يشار إليها بأنها ثوابت حرفية (lliteral constants).

الأعـــداد

الأعداد في بيثون أربعة أنواع:- أعداد صحيحة (integers)، أعداد صحيحة طويلة (long integers)، وكسور عشرية (floating point) وأعداد مركّبة (complex numbers).

- الأعداد الصحيحة مثل 2 التي هي عدد صحيح فقط.

- الأعداد الصحيحة الطويلة مثل الأعداد الصحيحة ولكنها أكبر.
- الكسور العشرية (floating point) أو *العشرية* اختصارا، مثل 3. 23 و 52. 3E-4. الرمز E يعني أس 10. في هذه الحالة 52. 3E-4 تساوي 52. 3 *10^{-4}.
- الأعداد المركّبة مثل (-5+4j) و (2. 3 - 4. 6j)

السلاسل

السلسلة (string) هي *تتابع* من *المحارف*. السلاسل ببساطة مجرد مجموعة من الكلمات.

أكاد أضمن لك أنك ستستخدم السلاسل تقريبا في كل برامج بيثون التي تكتبها، لذلك عليك الانتباه إلى الجزء التالي. وإليك كيفية استخدام السلاسل في بيثون:

•استخدام علامات التنصيص المفردة (')

يمكنك تحديد سلسلة بوضعها بين علامتي تنصيص مثل 'هذه جملة' جميع الفراغات مثل المسافات وعلامات التبويب تبقى كما هي.

استخدام علامات التنصيص المزدوجة (")

الجمل ضمن علامات التنصيص المزدوجة تعمل تماما كما في علامات التنصيص المفردة. مثلا "What's your name?"

•استخدام علامة التنصيص الثلاثية (''' أو """)

يمكنك تحديد سلاسل متعددة الأسطر باستخدام علامات التنصيص الثلاثية. ويمكنك استخدام علامة التنصيص المفردة والمزدوجة بحرية ضمن علامة التنصيص الثلاثية. مثلا

```
'''This is a multi-line string. This is the first line.
This is the second line.
"What's your name?," I asked.
He said "Bond, James Bond."
'''
```

تتابعات الخلوص (Escape Sequences)

افترض أنك تريد سلسلة تحتوي على علامة تنصيص فردية (')، كيف تحدد هذه السلسلة؟ مثلا 'What's your name?'. لا يمكنك استخدام 'What's your name?' لأن هذا سيربك بيثون حول مبدأ الجملة ومنتهاها. لذا سيتعين عليك أن تجعل علامة التنصيص المفردة هذه لا تشير إلى نهاية النص (لئلا تعتقد بيثون أن الجملة انتهت عند علامة الاقتباس

الأولى). يمكن إنجاز ذلك بمساعدة ما يسمى *تتابع الخلوص* (escape sequence). اجعل علامة الاقتباس المفردة هكذا \' -لاحظ الشرطة الخلفية (backslash)- الآن، يمكنك تحديد الجملة النصية بعلامة اقتباس مفردة هكذا 'What\'s your name?'.

هناك طريقة أخرى لتحديد هذه السلسلة باستخدام "What's your name?" أي باستخدام علامة تنصيص مزدوجة. وبالمثل، عليك استخدام تتابع خلوص من أجل استخدام علامة التنصيص المزدوجة داخل جملة محاطة بعلامة تنصيص مزدوجة. كذلك، عليك تحديد الشرطة المائلة الخلفية ذاتها باستخدام تتابع الخلوص \\.

ماذا لو أردت استخدام سلسلة من سطرين؟ يمكنك إما استخدام علامة تنصيص ثلاثية كما ذُكر سابقا أو يمكنك استخدام تتابع خلوص لمحرف السطر الجديد بكتابة \n لتشير إلى بدء سطر جديد. مثال على هذا This is the first line\nThis is the second line. تتابع خلوص مفيد آخر؛ علامة التبويب \t. هناك العديد من تتابعات الخلوص ولكني ذكرت هنا أكثرها منفعة فقط.

شيء آخر علينا ملاحظته في السلاسل، هو أن الشرطة الخلفية في نهاية السطر تشير إلى أن السلسلة مستمرة في السطر المقبل، ولكن بدون إضافة سطر جديد. فمثلا،

```
"This is the first sentence.\
This is the second sentence."
```

هو نفسه "This is the first sentence. This is the second sentence." .

السلاسل الغفل

إذا كنت في حاجة إلى تحديد سلاسل لا تريد أن يجري عليها أي معالجة خاصة مثل تتابعات الخلوص، فعليك تحديد سلسلة *غفل* (Raw String) بتقديم الحرف r أو R قبل السلسلة مثل r"Newlines are indicated by \n".

سلاسل يونيكود

يونيكود هي طريقة معيارية لكتابة النصوص العالمية. فإذا أردت كتابة نص بلغتك القومية مثل اللغة العربية أو الهندية، فستحتاج إلى محرر نصوص يدعم يونيكود. وبالمثل، فإن بيثون تتيح لك التعامل مع نصوص يونيكود، كل ما عليك فعله هو إضافة البادئة u أو U قبل النص. على سبيل المثال، u"This is a Unicode string". "

تذكر أن تستخدم يونيكود عندما تريد التعامل مع الملفات النصية، وخاصة تلك التي تحتوي نصوصا بلغة غير الإنجليزية.

السلاسل ثابتة

وهذا يعني أنه بعد إنشاء السلسلة لا يمكنك تغييرها. ورغم أن هذا قد يبدو أمرا سيئا، إلا أنه ليس كذلك في الحقيقة. سنرى لم ليس هذا قيدا في مختلف البرامج التي سنراها في وقت لاحق.

تجميع نصوص السلاسل

إذا كنت وضعت سلسلتين جنبا إلى جنب، فستقوم بيثون تلقائيا بجمعهما. على سبيل المثال، 'What\'s' 'your name?' تُحول تلقائيا إلى "What's your name?".

ملاحظة لمبرمجي سي/سي++

لا يوجد نوع بيانات منفصل للمحارف (char) في بيثون. ولا حاجة له، وأعتقد أنكم لن تفتقدوها.

ملاحظة لمبرمجي بيرل/PHP

تذكر أن السلاسل داخل علامات التنصيص المفردة أو المزدوجة أو الثلاثية بمعنى واحد ولا تختلف بأي شكل.

ملاحظة لمستخدمي التعبيرات النمطية

دائما استخدام السلاسل الغفل عند التعامل مع التعابير النمطية. وإلا فستحتاج للكثير من تتابعات الخلوص. فعلى سبيل المثال، الإحالة الخلفية (backreference) يمكن الإشارة إليها باستخدام '\\1' أو 'r'\1'.

المتغيرات

استخدام الثوابت الحرفية فقط يمكن أن يصبح سريعا أمرا مملاً، ونحن بحاجة الى طريقة ما لتخزين المعلومات ومعالجتها. هنا تظهر *المتغيرات* (Variables) في الصورة. المتغيرات -كما يظهر من اسمها- يمكن أن تتغير قيمتها، أي يمكنك أن تخزن أي شيء باستخدام متغير. المتغيرات ليست سوى أجزاء محجوزة من ذاكرة الحاسوب حيث تخزن بعض المعلومات. بعكس الثوابت الحرفية، تحتاج إلى طريقة ما للوصول إلى هذه المتغيرات، ولذا نعطيها أسماء.

تسمية المعرف

المتغيرات هي أمثلة على *المعرفات* (Identifiers)، المعرفات هي أسماء تعطى لتعريف *شيء ما*. هناك بعض القواعد عليك اتباعها لتسمية المعرفات:

- المحرف الأول من المعرف يجب أن يكون حروفا من حروف الهجاء (لا تيني كبير أو صغير) أو شرطة تحتية ('_'، underscore).
- بقية اسم المعرف يمكن أن تتكون من الحروف (كبيرة أو صغيرة)، أو الشرطة التحتية ('_')، أو الأرقام (0-9).
- أسماء المعرف حساسة لحالة الحرف (سواء لاتيني كبير أو صغير) على سبيل المثال myname و myName **لا** يتساويان. لاحظ الحرف الصغير n في الأولى والكبير N في الثانية.
- أسماء المعرفات *الصالحة* مثل i، و __my_name، و name_23 و a1b2_c3.
- أسماء المعرفات *غير الصالحة* مثل 2things، و this is spaced out و my-name.

أنواع البيانات

يمكن أن تحمل المتغيرات أنواعا مختلفة من القيم تسمى **أنواع البيانات**. الأنواع الأساسية هي الأعداد والسلاسل، التي ناقشناها من قبل. وفي الفصول القادمة سنرى كيفية إنشاء أنواع خاصة بنا باستخدام الأصناف (classes).

الكائنات

تذكر أن بيثون تشير إلى أي *شيء* مستخدم في برنامج على أنه *كائن* (object)، وهذا مقصود على وجه عام. فبدلا من قول '*الشيء*' نقول '*الكائن*'.

ملاحظة لمستخدمي البرمجة الكائنية

بيثون كائنية التوجه بقوة، بمعنى إن كل شيء عبارة عن كائن، سواء الأعداد أو النصوص أو حتى الدوال (functions).

سنرى الآن كيف نستخدم المتغيرات مع الثوابت الحرفية. احفظ المثال التالي في ملف ثم شغل البرنامج.

كيف تكتب برامج بيثون

من الآن فصاعدا، الخطوات القياسية لحفظ وتشغيل برامج بيثون هي:

1. افتح محرر النصوص المفضل.
2. أدخل كود البرنامج كما في المثال.
3. احفظ الملف بالاسم المذكور في التعليق. اتبعُ الطريقة المتعارف عليها في

حفظ برامج بيثون بالامتداد .py.

4.غلّ مفسر بيثون متبوعا باسم البرنامج (الأمر **python** program.py) أو استخدم IDLE لتشغيل البرامج. يمكنك ايضا استخدام طريقة الملفات التنفيذية المذكورة سابقا.

مثال 4.1. استخدام المتغيرات والثوابت الحرفية

```
# Filename : var.py
i = 5
print i
i = i + 1
print i

s = '''This is a multi-line string.
This is the second line.'''
print s
```

الخـــرْج

```
$ python var.py
5
6
This is a multi-line string.
This is the second line.
```

كيـف يعمـــل

إليك كيف يعمل هذا البرنامج. أولا أسندنا القيمة الثابتة الحرفية 5 إلى المتغير i باستخدام المعامل (=). هذا السطر يسمى إفادة (statement) لأنه يبين ويعين شيئا ما ينبغي القيام به، وفي حالتنا هذه، ربطنا اسم المتغير i بالقيمه 5. بعد ذلك طبعنا قيمة i باستخدام جملة print التي -كما هو متوقع- تطبع قيمة المتغير على الشاشة.

ثم أضفنا 1 إلى القيمة المخزنة في المتغير i ثم خزناه ثانية. ثم طبعناه، و -كما

نتوقع- حصلنا على القيمة 6.

وبالمثل، أسندنا سلسلة حرفية للمتغير S ثم طبعناه.

ملاحظة لمبرمجي سي/سي++

تستخدم المتغيرات بمجرد إسنادها إلى قيمة. فلاحاجة للإعلان (declaration) أو تعريف نوع البيانات.

السطور المادية والمنطقية

السطر المادي (Physical) هو ما *تراه* عندما تكتب البرنامج. والسطر المنطقي (Logical) هو ما *تراه* (تفهمه) بيثون كإفادة واحدة. بيثون تفترض ضمنا أن كل *سطر مادي* يقابل *سطرا منطقيا*.

الإفادة التالية مثال على السطر المنطقي print 'Hello World'؛ إذا كان مكتوبا على سطر وحده (كما تراه في المحرر) فهذا أيضا سطر مادي.

تشجع بيثون على استعمال إفادة واحد لكل سطر مما يجعل الكود مقروءا أكثر.

إذا أردت أن تحدد أكثر من سطر منطقي على سطر مادي واحد فعليك تحديد هذا صراحة باستخدام الفاصلة المنقوطة (;) حيث تشير إلى نهاية السطر المنطقي أو الإفادة. على سبيل المثال:

```
i = 5
print i
```

عمليا لا يختلف عن :

```
i = 5;
print i;
```

ويمكن أن يكتب هكذا :

```
i = 5; print i;
```

أو حتى :

```
i = 5; print i
```

ومع ذلك، فإنني **أوصي بقوة** أن تلتزم **كتابة سطر منطقي واحد فقط في كل سطر مادي واحد.** استخدم أكثر من سطر مادي لكل سطر منطقي إذا كان السطر المنطقي طويلا حقا. الفكرة هي تجنب الفاصلة المنقوطة قدر الإمكان لأن هذا يعني

كودا أسهل قراءة. في الحقيقة، أنا لم أستخدم *أبدا* أو حتى أرى الفاصله المنقوطه في برنامج بيثون.

فيما يلي مثال على كتابة سطر منطقي يمتد عبر عدة أسطر مادية. سمى هذا **الوصل العمدي للأسطر** (Explicit line joining).

```
s = 'This is a string. \
This continues the string.'
print s
```

وهذه تعطينا الناتج

```
This is a string. This continues the string.
```

وبالمثل:

```
print \
i
```

هي بالضبط مثل

```
print i
```

أحيانا يكون هناك افتراض ضمني بحيث لا تحتاج الى استخدام الشرطة العكسية المائلة (backslash). وهذا هو الحال عندما يستخدم السطر المنطقي الأقواس الهلالية أو المربعة أو المجعدة. وهذا يسمى **الوصل الضمني للأسطر**. يمكنك أن ترى هذا يعمل عندما نكتب برامج مستخدمين القوائم في الفصول القادمة.

الإزاحـــة

الفراغات البيضاء مهمة في بيثون وخاصة **في بداية السطر**. هذا ما يسمى **الإزاحة** (Indentation). الفراغات البيضاء (المسافات وعلامات التبويب) في بداية السطور المنطقية تستخدم لتحديد مستوى إزاحة السطر المنطقي، والذي بدوره يستخدم لتحديد تجميع من الإفادات (Statements).

وهذا يعني أن الإفادات المرتبطة ببعضها **يجب** أن يكون لها نفس الإزاحة. كل مجموعة من هذه التصريحات تسمى **لبنة** (block). وسنرى أمثلة على مدى أهمية اللبنات في

الفصول اللاحقة.

أمر واحد عليك أن تتذكره وهو أن الإزاحة الخاطئة يمكن أن تؤدي إلى أخطاء. فعلى سبيل المثال:

```
i = 5
print 'Value is', i # Error! Notice a single space at the
start of the line
print 'I repeat, the value is', i
```

عند تشغيله يعطيك هذا الخطأ:

```
File "whitespace.py", line 4
    print 'Value is', i # Error! Notice a single space at
the start of the line
    ^
SyntaxError: invalid syntax
```

لاحظ أن ثمة مسافة فارغة واحدة في بداية السطر الثاني. الخطأ الذي أشارت إليه بيثون يخبرنا أن التركيب النحوي (syntax) لهذا البرنامج خطأ، أي أن البرنامج غير مكتوب على الوجه الصحيح. معنى هذا أنك *لا تستطيع بدء لبنة إفادات جديدة* بصورة اعتباطية (باستثناء اللبنة الرئيسية التي تستخدمها دائما، بطبيعة الحال). الحالات التي يمكنك فيها استخدام اللبنات الجديدة سيتم تفصيلها في فصول لاحقة مثل فصل التحكم في التدفق.

كيفية الإزاحة

لا تستخدم خليطا من المسافات وعلامات التبويب للإزاحة لأنها لا تعمل على النحو الصحيح عبر مختلف المنصات. *وأوصي بشدة* أن تستخدم *علامة تبويب واحدة* أو *مسافتين أو أربع* لكل مستوى إزاحة.

اختر أيا من أساليب الإزاحة هذه. الأهم أن تختار واحدة وتستخدمها **باذضباط**، أي استخدم أسلوب الإزاحة هذا فقط.

الخلاصة

الآن بعد أن مررنا على كثير من التفاصيل الدقيقة، يمكننا الانتقال إلى أشياء أكثر متعة مثل التحكم في تدفق الإفادات. تأكد من أن تعتاد على ما قرأته في هذا الفصل.

فصل.5. العوامل والتعبيرات

قائمة المحتويات

مقدمـــة

أغلب الإفادات (statements) (السطور المنطقية) التي ستكتبها تحتوي على **تعبيرات** expressions). 2 + 3) مثال بسيط على أحد التعبيرات.

تنقسم التعبيرات إلى عوامل ومعاملات:

العوامل (operators) هي وظيفة تقوم بعمل شيء ما ويمكن تمثيلها برمز مثل + أو كلمة مفتاحية خاصة. تحتاج العوامل لبيانات تعمل عليها وتسمى هذه *المعاملات* (operands). ففي هذه الحالة تعتبر 2 و 3 معاملات.

العوامل

سنأخذ نظرة سريعة على العوامل واستخداماتها.

فكرة مفيدة

يمكنك حساب التعبير المعطى في المثال السابق باستخدام المفسر تفاعليا. مثلا لاختبار التعبير 2 + 3 استخدم مؤشر مفسر بيثون التفاعلي:

```
>>> 2 + 3
5
>>> 3 * 5
15
>>>
```

جدول 5.1. العوامل الرياضية واستخداماتها

العامل	الاسم	الشرح	أمثلة
+	زائد	يجمع عنصرين	3 + 5 تعطي 8. 'a' + 'b' تعطي 'ab'.
-	ناقص	إما يعطي عددا سالبا، أو يطرح عددا من آخر	-5. 2 تعطي عددا سالبا. 50 - 24 تعطي 26.
*	ضرب	تعطي حاصل ضرب عددين أو تعيد سلسلة مكررة هذا العدد من المرات.	2 * 3 تعطي 6. 'la' * 3' تعطي 'lalala'.
**	أُس	تعيد س أُس ص	3 ** 4 تعطي 81 (أي 3 * 3 * 3 * 3)
/	قسمة	اقسم س على ص	4/3 تعطي 1 (ناتج قسمة الأعداد الصحيحة عدد صحيح). 4. 0/3 أو 4/3. 0 تعطي 1. 3333333333333333
//	Floor Division	Returns the floor of the quotient	4 // 3. 0 تعطي 1. 0
%	Modulo	يعيد باقي عملية القسمة	8%3 تعطي 2. -25. 5%2. 25 تعطي 1. 5.
>>	إزاحة لليسار	تزيح بتات العدد لليسار بعدد البتات المحدد. (كل عدد يمثل في الذاكرة ببتات أو أرقام ثنائية؛ 0 و 1)	2 >> 2 تعطي 8. - 2 تمثل 10 ثنائيا. الإزاحة لليسار ب 2 بتات تعطي 1000 والتي تمثل العدد العشري 8.
<<	إزاحة لليمين	تزيح بتات العدد لليمين بعدد البتات المحدد.	11 >> 1 تعطي 5. 11تمثل ب 1011 ثنائيا، والتي تعطي 101 عند إزاحتها بتة واحدة لليمين، وهو ما يساوي 5عشريا.
&	"و" بتّية	"و" بتّية للأعداد	5 3 تعطي 1.
|	"أو" بتّية	"أو" بتّية للأعداد	5 | 3 تعطي 7
^	Bit-wise XOR	5 ^ 3 تعطي 6	
~	"معكوس" بتّية	المعكوس البتّي ل س هو -(س+1)	~5 تعطي -6.
>	أقل من	تعيد ما إذا كانت س أصغر من ص. كل عوامل المقارنة تعيد 1 للصحيح و 0 للخطأ. يساوي هذا المتغيرات الخاصة صحيح و خطأ بالترتيب. لاحظ الحروف الكبيرة في أسماء هذه المتغيرات.	5 > 3 تعطي 1 (أي صحيح) و 3 > 5 تعطي 0 (أي خطأ). يمكن سلسلة المقارنة: 3 < 5 < 7 تعطي صحيح.
<	أكبر من	تعيد ما إذا كانت س أكبر من ص	5 > 3 تعيد صحيح. إذا كان كلا المعاملين أعداد

العامل	الاسم	الشرح	أمثلة
			فيتم تحويلهما أولا إلى نوع مشترك. وإلا فإنه يعيد دائما خطأ.
=>	أقل من أو يساوي	تعيد ما إذا كانت س أقل من أو تساوي ص	x = 3; y = 6; x <= y تعيد صحيح.
=<	أكبر من أو يساوي	تعيد ما إذا كانت س أكبر من أو تساوي ص	x = 4; y = 3; x >= 3 تعيد صحيح.
==	يساوي	تقارن ما إذا كان الكائنين يتساويان	x = 2; y = 2; x == y تعيد صحيح. x = 'str'; y = 'stR'; x == y تعيد خطأ. x = 'str'; y = 'str'; x == y تعيد صحيح.
=!	لا يساوي	تقارن ما إذا كان الكائنين لا يتساويان	x = 2; y = 3; x != y تعيد صحيح.
not	"ليس" منطقية	إذا كانت سصحيح، تعيد خطأ. إذا كانت س خطأ، تعيد صحيح.	x = True; not y تعيد خطأ.
and	"و" منطقية	x and y تعيد خطأ إذا كانت x خطأ، وإلا فإنها تعيد y	x = False; y = True; x and y تعيد خطأ لأن x خطأ. في هذه الحالة لن تعيد بيثون قيمة y لأنها تعرف أن قيمة التعبير ستكون خطأ (لأن x خطأ). يسمى هذا التقييم المختصر.
or	"أو" منطقية	إذا كانت x صحيح فإنها تعيد صحيح وإلا فإنها تعيد قيمة y.	x = True; y = False; x or y تعيد صحيح. التقييم المختصر ينطبق هنا أيضا.

أسبقية العوامل

إذا كان لديك عملية حسابية مثل 2 + 3 * 4 هل سيتم الجمع أولا أم الضرب ؟ رياضيات المدرسة الثانوية تخبرنا أن عملية الضرب يجب إجراؤها أولا؛ يعني هذا أن لعملية الضرب أسبقية على عملية الجمع.

الجدول التالي يعطينا بيان بأسبقية العوامل في بيثون بدءا بالأسبقية الأدنى (أقل إلزاما) إلى الأسبقية الأعلى (أكثر إلزاما). معنى هذا أنه بداخل التعبيرات المعطاة فإن بيثون ستقوم بحساب قيمة العوامل الأدنى في الجدول قبل العوامل الأعلى في الجدول.

الجدول التالي (نفس ذلك الموجود في مرجع بيثون) موجود من باب زيادة العلم. ومع هذا، أنصحك باستخدم الأقواس ( ) لتجميع العوامل والمعاملات لتحديد الأسبقية بوضوح ولتجعل برنامجك أسهل في القراءة قدر الإمكان. على سبيل المثال، 2 + (3 * 4) بالطبع أكثر وضوحا من 2 + 3 * 4. كأي شيء آخر، يجب استخدام الأقواس بحكمة ولا ينبغي أن تكون زائدة عن الحد. (مثل 2 + (3 + 4)).

جدول 5.2. أسبقية العوامل

العامل	الوصف
lambda	تعبير لامدا
or	"أو" منطقية
and	"و" منطقية
not x	"ليس" منطقية
in, not in	اختبارات العضوية
is, is not	اختبارات الهوية
<, <=, >, >=, !=, ==	المقارنات
|	"أو" بتّية
^	Bitwise XOR
&	"و" بتّية
<<, >>	الإزاحة
+, -	الجمع والطرح
*, /, %	الضرب والقسمة والباقي
+x, -x	الموجب والسالب
~x	"ليس" بتّية
**	الأُس
x.attribute	إخالة الخصائص
x[index]	الاشتراك
x[index:index]	الجز
f(arguments...)	استدعاء الدوال
(expressions, ...)	Binding or tuple display
[expressions, ...]	عرض القوائم
{key:datum, ...}	عرض القواميس
`expressions, ...`	تحويل السلاسل

العوامل التي لم نمر عليها سابقا ستشرح في الفصول القادمة.

العوامل *المتساوية في الأسبقية* وضعت في نفس الصف في الجدول السابق. على سبيل المثال + و - لهما نفس الأسبقية.

ترتيب الحساب

مبدئيا؛ يحدد جدول أسبقية العوامل أي عامل له أسبقية على الآخر. لذلك إذا أردت تغيير ترتيب حساب العمليات، فعليك استخدام الأقواس ( )، على سبيل المثال إذا أردت أن تتم عملية الجمع قبل عملية الضرب في أحد التعبيرات حينئذ يمكن استخدام الأقواس مثل `(2 + 3) * 4`.

الارتباطية

العوامل ترتبط عادة من اليسار إلى اليمين، أي أن العوامل المشتركة في الأسبقية تحسب من اليسار إلى اليمين. على سبيل المثال `2 + 3 + 4` تقيم مثل `(2 + 3) + 4`. بعض العوامل مثل عوامل الإسناد ترتبط من اليمين إلى اليسار، أي أن `a = b = c` تعامل باعتبارها `a = (b = c)`.

التعبيرات

استخدام التعبيرات

مثال 5.1. استخدام التعبيرات

```
#!/usr/bin/python
# Filename: expression.py

length = 5
breadth = 2

area = length * breadth
print 'Area is', area
print 'Perimeter is', 2 * (length + breadth)
```

الخرْج

```
$ python expression.py
Area is 10
Perimeter is 14
```

كيف يعمل

طول وعرض المستطيل تخزن في المتغيرات (length و breadth). ونحن نستخدمها لحساب مساحة ومحيط المستطيل بمساعدة التعبيرات. نخزن نتيجة التعبير `length * breadth` في المتغير في `area` ثم نطبعه باستخدام الإفادة `print`. في الحالة الثانية، فإننا نستخدم مباشرة قيمة التعبير `2 * (length + breadth)` في الإفادة print.

أيضا، لاحظ كيف تطبع بيثون النتيجة بأناقة. ورغم أننا لم نحدد مسافة بين `'Area` و`is'` والمتغير `area` فقد وضعتها بيثون لنا بحيث نحصل على ناتج نظيف ولطيف ويصبح البرنامج أكثر مقروئية بهذه الطريقة (حيث أننا لا نحتاج للاهتمام بالمسافات في الناتج). هذا مثال على كيف أن بيثون تجعل حياة المبرمج أسهل.

الخلاصة

رأينا كيف نستخدم العوامل (operators) و المعاملات (operands) والتعبيرات (expressions) وهي اللبنات الأساسية لأي برنامج. وفيما يلي سنرى كيف نقوم باستخدامها في برامجنا باستخدام الإفادات.

فصل 6. التحكم في التدفق

قائمة المحتويات

مقدمـــة

في البرامج التي رأيناها حتى الآن كان هناك دائما تتابع من الإفادات؛ وبايثون تنفذها محترمة نفس الترتيب. ماذا لو أردت تغيير طريقة انسياب العمل؟ على سبيل المثال، تريد من البرنامج اتخاذ بعض القرارات والقيام بأشياء مختلفة تبعا للمواقف المختلفة مثل طباعة 'Good Morning' أو 'Good Evening' معتمدا على الوقت الحالي من اليوم؟

وكما خمنت، فإن ذلك يتم عبر استخدام إفادات التحكم في التدفق. يوجد ثلاث إفادات للتحكم في التدفق قي بيثون؛ if، و for، و while.

إفادة if

تستخدم إفادة if للتحقق من حالة معينة و *إذا* كانت الحالة صحيحة، نشغل لبنة من الإفادات (تسمى *if-block*)، *وإلا* فإننا نعالج لبنة أخرى من الإفادات (تسمى *else-block*). بند *else* اختياري.

استخدام إفادة if

مثال 6.1.استخدام إفادة if

```
#!/usr/bin/python
# Filename: if.py

number = 23
guess = int(raw_input('Enter an integer : '))

if guess == number:
        print 'Congratulations, you guessed it.' # New block
starts here
        print "(but you do not win any prizes!)" # New block
ends here
elif guess < number:
        print 'No, it is a little higher than that' # Another
block
        # You can do whatever you want in a block ...
else:
        print 'No, it is a little lower than that'
        # you must have guess > number to reach here

print 'Done'
# This last statement is always executed, after the if statement is executed
```

الخرج

```
$ python if.py
Enter an integer : 50
No, it is a little lower than that
Done
$ python if.py
Enter an integer : 22
No, it is a little higher than that
Done
$ python if.py
Enter an integer : 23
Congratulations, you guessed it.
(but you do not win any prizes!)
```

```
Done
```

كيف يعمل

في هذا البرنامج، فإننا نأخذ تخمينات من المستخدم ونقارن بينه وبين العدد الذي لدينا. نسند المتغير number إلى أي عدد صحيح نريده، مثلا 23. ثم، أخذنا تخمين المستخدم باستخدام الدالة raw_input(). الدوال ما هي إلا أجزاء من البرامج يمكن إعادة استخدامها. سنقرأ المزيد عنها في الفصل التالي.

أعطينا سلسلة للدالة المدمجة raw_input والتي تقوم بطباعة النص على الشاشة وتنتظر المدخلات من المستخدم. وبمجرد أن ندخل أي شيء ونضغط على **enter** تعيد الدالة المدخلات، وهي في حالة raw_input عبارة عن سلسلة نصية، بعدها نحول هذا النص إلى عدد صحيح باستخدام int ثم نخزنه في المتغير guess. في الحقيقة int عبارة عن صنف (class) ولكن كل ما تحتاج معرفته الآن هو أنك تستطيع استخدامه لتحويل السلاسل إلى أعداد صحيحة (على فرض أن السلسلة تحتوي على عدد صحيح سليم داخل النص).

بعد ذلك قارنا بين تخمين المستخدم وبين العدد الذي اخترناه. فإذا تساويا نطبع رسالة بنجاح العملية. لاحظ أننا نستخدم مستويات الإزاحة لنخبر بيثون أي الإفادات تنتمي إلى أية لبنة. ذلك يبين سبب أهمية الإزاحة في بيثون. أتمنى أن تثبت على قاعدة 'علامة تبويب (tab) واحدة لكل مستوى إزاحة'. فهل ستفعل؟

لاحظ كيف أن إفادة if تحتو ي على نقطتين رأسيتين (:) في النهاية؛ نخبر بيثون بأن لبنة من الإفادات ستتبعها.

عد ذلك تحقق من كون التخمين أقل من ذلك العدد، وإذا كان كذلك فإننا نطلب من المستخدم تخمين عدد أكبر قليلا. ما استخدمناه هنا هو بند elif والذي يدمج بين إفادتي if else else-if متتابعتين في إفادة if-elif-else واحدة. يجعل هذا البرنامج أسهل ويقلل مقدار الإزاحات المطلوبة.

يجب أنت تحتوي إفادات elif و else على (:) في نهاية السطر المنطقي متبوعة بلبنة الإفادات المقابلة لها (مع الإزاحة المناسبة بالطبع)

يمكن احتواء إفادة if على إفادة if أخرى وهكذا؛ يسمى هذا إفادات if المتتداخلة.

* تذكر ان الأجزاء elif و else اختيارية. أقل إفادة if صحيحة هي

```
if True:
  print 'Yes, it is true'
```

بعد انتهاء بيثون من تنفيذ إفادة if كاملة بما فيها بنود elif و else المندرجة تحتها، تنتقل إلى الإفادة التالية في اللبنة التي تحتوي إفادة if. وهي في هذه الحالة لبنة البرنامج الرئيسة

حيث يبدأ تنفيذ البرنامج والإفادة التالية هي print 'Done'. بعدها ترى بيثون نهاية البرنامج ومن ثم تنهيه ببساطة.

ورغم أن هذا البرنامج بسيط جدا، فقد أشرت إلى الكثير من الاشياء التي عليك أن تلاحظها حتى في هذا البرنامج البسيط. وهذه كلها أشياء واضحة (وبسيطة بشكل مفاجئ لأولئك القادمين بخلفية عن سي/سي++) ويتطلب منك أن تصبح مدركا لكل هذا في البداية، ولكن بعد ذلك، ستصبح مرتاحا معها وستشعر بأنها 'طبيعية' بالنسبة لك.

ملاحظة لمبرمجي سي/سي++

لا توجد إفادة تبديل (switch) في بيثون. يمكنك استخدام إفادة if..elif..else لفعل نفس الشيء (وفي بعض الحالات، استخدم قاموس لفعل ذلك بسرعة)

إفادة while

إفادة while تسمح لك تكرار تنفيذ لبنة من الإفادات ما دام الشرط صحيحا. إفادة while مثال لما يسمى الإفادات *الحلقية* (looping). يمكن أن تشمل إفادة while على بند else اختياري.

استخدام إفادة while

مثال 6.2. استخدام إفادة while

```
#!/usr/bin/python
# Filename: while.py

number = 23
running = True

while running:
        guess = int(raw_input('Enter an integer : '))

        if guess == number:
                print 'Congratulations, you guessed it.'
                running = False # this causes the while loop to
stop
```

```
        elif guess < number:
                print 'No, it is a little higher than that.'
        else:
                print 'No, it is a little lower than that.'
else:
        print 'The while loop is over.'
        # Do anything else you want to do here

print 'Done'
```

الخرْج

```
$ python while.py
Enter an integer : 50
No, it is a little lower than that.
Enter an integer : 22
No, it is a little higher than that.
Enter an integer : 23
Congratulations, you guessed it.
The while loop is over.
Done
```

كيف يعمل

في هذا البرنامج، ما زلنا نلعب لعبة التخمين، لكن الميزة هنا هي أن المستخدم يسمح له بمواصلة التخمين حتى يصل للتخمين الصحيح؛ لا حاجة إلى تكرار تنفيذ البرنامج لكل تخمين - كما فعلنا سابقا-. هذا يشرح بوضوح فائدة إفادة while.

نقلنا إفادة raw_input و if إلى داخل حلقة while. وجعلنا المتغير running صحيح (True) قبل حلقة while. أولا: نتحقق من كون المتغير running صحيح وبعدها ننتقل إلى تنفيذ *while-block* المناسبة. بعد تنفيذ هذه اللبنة، تفحص الحالة ثانيا -وهي المتغير running هنا-. إذاكانت صحيحة، ننفذ لبنة while ثانيا، وإلا فإننا ننتقل لتنفيذ لبنة else الاختيارية، ثم ننتقل للإفادة التالية.

يتم تنفيذ لبنة else عندمايكون الشرط في الحلقة while خاطئا وربما تكون هذه المرة الأولى التي يتم التحقق فيها من الشرط. إذا كان هناك بند else للحلقة while، فسيتم

تنفيذه حتما إلا إن كان لديك حلقة while تدور إلى ما لا نهاية دون توقف.

تسمى True و False أنواعا منطقية (Boolean types) ويمكنك أن تعتبرها معادلا لقيمة 1 و 0 على التوالي. و من المهم استعمالها حيثما يكون الشرط أو التحقق مهما وليس القيمه الفعليه 1.

في الحقيقة لبنة else تعد حشوا، حيث يمكنك وضع الإفادات التابعة لها في نفس اللبنة (مثل إفادة while) بعد while للحصول على نفس التأثير.

ملاحظة لمبرمجي سي/سي++

تذكر أنه يمكن وضع بند else لحلقة while.

الحلقة for

تعتبر for..in إفادة حلقية أخرى تَمُرّ على تتابع من الكائنات، مثل مُرّ عبر كل عنصر في تتابع. و سنتعرف على المزيد عن التتابعات في فصول لاحقة. كل ما عليك معرفته الآن أن التتابع هو مجرد مجموعة من العناصر المرتّبة.

استخدام إفادة for

مثال 6.3. استخدام إفادة for

```
#!/usr/bin/python
# Filename: for.py

for i in range(1, 5):
        print i
else:
        print 'The for loop is over'
```

الخرْج

```
$ python for.py
1
2
3
4
```

```
The for loop is over
```

كيف يعمل

في هذا البرنامج نطبع *تتابعا* من الأعداد. ولّدنا تتابع الأعداد هذا باستخدام الدالة المدمجة range.

ما قمنا به هنا أننا أعطيناه عددين ثم range يعيد لنا تتابعا من الأعداد بداية من العدد الأول ثم تصاعديا حتى يصل إلى ما قبل العدد الأخير. على سبيل المثال range(1,5) يعطي التتابع [1, 2, 3, 4] افتراضيا، يأخذ range خطوة بقيمة 1. إذا أعطيته عددا ثالثا تكون الخطوة بمقدار ذلك العدد، على سبيل المثال: range(1,5,2) تعطي [1,3]. تذكر أن المدى يمتد حتى *ما قبل* العدد الثاني-أي إنه **لا** يشمل العدد الثاني.

بعد ذلك تدور الحلقة for خلال ذلك المدى -for i in range(1,5) تعادل for i in [1, 2, 3, 4]- والتي تشبه إسناد كل عدد (أو كائن) إلى المتغير i واحدا في كل مرة، بعدها يتم تنفيذ لبنة الإفادات لكل قيمة i. في حالتنا هذه نقوم فقط بطباعة القيمة في لبنة الإفادات.

تذكر أن الجزء else اختياري. وعندما يضاف، فإنه يُنفذ مرة واحدة بعد انتهاء الحلقة for إلا إذا قابل إفادة break.

تذكر أن الحلقة for..in تعمل مع أي تتابع. هنا لدينا قائمة من الأعداد تم توليدها باستخدام الدالة المدمجة range، ولكن على العموم يمكننا استخدام أي تتابع لأي كائنات. وسنشرح هذه الفكرة بالتفصيل في الفصول القادمة.

ملاحظة لمبرمجي سي/سي++/جافا/سي#

الحلقة for في بيثون تختلف اختلافا جذريا عن سي/سي++. مبرمجو سي# سيلاحظون أن الحلقة for في بيثون مشابهة لحلقة foreach في سي#. مبرمجو Java سيلاحظون أيضا تشابهها مع (for (int i : IntArray في جافا 1. 5.

في سي/سي++، إذا أردت أن تكتب for (int i = 0; i < 5; i++)، ففي بيثون تكتب فقط for i in range(0,5). وكما ترى، فإن كتابة الحلقة في بيثون أكثر بساطة وأقل تعبيرا وعرضة للخطأ.

الإفادة break

إفادة break تستخدم لكسر الحلقة التكرارية، بعبارة أخرى: وقف تنفيذ الحلقة حتى ولو لم يصبح شرط الحلقة خاطئا أو لم تمر على تتابع العناصر كله.

من المهم ملاحظة أن كسر حلقة for أو while سيؤدي إلى **عدم** تنفيذ لبنة إفادات else.

استخدام الإفادة break

مثال 6.4. استخدام الإفادة break

```
#!/usr/bin/python
# Filename: break.py

while True:
        s = raw_input('Enter something : ')
        if s == 'quit':
                break
        print 'Length of the string is', len(s)
print 'Done'
```

الخرج

```
$ python break.py
Enter something : Programming is fun
Length of the string is 18
Enter something : When the work is done
Length of the string is 21
Enter something : if you wanna make your work also fun:
Length of the string is 37
Enter something :       use Python!
Length of the string is 12
Enter something : quit
Done
```

كيف يعمل

في هذا البرنامج كررنا أخذ المدخلات من المستخدم ثم طبعنا طول كل مدخلة في كل مرة. وقد وفرنا شرط خاص لوقف البرنامج من خلال فحص ما إذا كانت مدخلة المستخدم هي 'quit' وأوقفنا عمل البرنامج عن طريق كسر الحلقة والوصول إلى نهاية البرنامج.

يمكن معرفة طول السلسلة المدخلة باستخدام الدالة المدمجة len.

تذكر أن إفادة break يمكن استخدامها مع الحلقة for أيضا.

قصيدة 'G2' لبايثون

المدخلات المستخدمة هنا قصيدة قد كتبتها وسميتها G2's Poetic Python:

```
Programming is fun
When the work is done
if you wanna make your work also fun:
        use Python!
```

الإفادة continue

الإفادة continue تستخدم لنطلب من بيثون تخطي بقية ما ورد في لبنة الحلقة الحالية ومواصلة تكرار الحلقة.

استخدام الإفادة continue

مثال 6.5. استخدام الإفادة continue

```
#!/usr/bin/python
# Filename: continue.py

while True:
        s = raw_input('Enter something : ')
        if s == 'quit':
                break
        if len(s) < 3:
                continue
        print 'Input is of sufficient length'
        # Do other kinds of processing here...
```

الخرج

```
$ python continue.py
```

```
Enter something : a
Enter something : 12
Enter something : abc
Input is of sufficient length
Enter something : quit
```

كيف يعمل

في هذا البرنامج نقبل مدخلات المستخدم، ولكن نعالجها فقط عندما يكون طولها 3 خانات على الأقل. لذا نستخدم الدالة المدمجة len للحصول على طول العبارة، فإذا كان الطول أقل من 3 خانات نتخطي بقية الإفادات الموجود في اللبنة باستخدام الإفادة continue وإلا ننفذ بقية الإفادات في الحلقة، ويمكننا القيام بأي معالجة نريدها هنا.

لاحظ أن الإفادة continue تعمل مع الحلقة for أيضا.

الخلاصة

رأينا كيفية استخدام ثلاث أدوات للتحكم في تدفق البيانات :if و while و for مع الإفادات break و continue المرتبطة بها. وتلك بعض من أكثر الأجزاء المستخدمة عادة في بيثون؛ ولذا تعودك عليها أمر ضروري.

وفيما يلي سنرى كيف ننشئ ونستخدم الدوال.

فصل 7. الدوال

قائمة المحتويات

مقدمة

الدوال هي أجزاء من البرامج يمكن إعادة استخدامها. تسمح لك بإعطاء اسم ما للبنة من الإفادات، ويمكنك تشغيل هذه اللبنة في أي مكان من برنامجك، وأي عدد من المرات. يسمى هذا *استدعاء* الدالة. و قد استخدمنا بالفعل بعضا من الدوال المدمجة مثل الدالة len والدالة range.

تُعَرَّف الدالة بالكلمة المفتاحية def متبوعة باسم *المعرف* للدالة ثم زوجين من الأقواس الهلالية () التي قد تحوي بعض أسماء المتغيرات وينتهي السطر بنقطتين (:). يعقب ذلك لبنة من الإفادات التي تشكل هذه الدالة. وهذا المثال يوضح مدى بساطتها:

تعريف دالة

مثال 7.1. تعريف دالة

```
#!/usr/bin/python
```

```
# Filename: function1.py

def sayHello():
    print 'Hello World!' # block belonging to the function
# End of function

sayHello() # call the function
```

الخرْج

```
$ python function1.py
Hello World!
```

كيف يعمل

ذُعرِّف دالة اسمها sayHello باستخدام التركيب الموضح أعلاه. هذه الدالة لا تأخذ أي معاملات، وبالتالي لم يُعلن عن أي متغيرات بين القوسين. معاملات الدالة هي مجرد مدخلات للدالة حتى نتمكن من تمرير قيم مختلفة لها ونحصل على النتائج المقابلة.

معاملات الدالة

يمكن أن تأخذ الدالة معاملات، والتي ليست سوى قيم تعطيها لهذه الدالة لتتمكن من *عمل* شيئ ما. هذه المعاملات تشبه المتغيرات غير أن قيم هذه المتغيرات يتم تحديدها عندما نستدعي الدالة، ولا يسند لها قيم داخل الدالة نفسها.

تحدد المعاملات داخل زوج من الأقواس () في تعريف الدالة، ومفصولة بنقطتين (:). عندما نستدعي الدالة نعطيها القيم بنفس الطريقة. لاحظ المصطلحات المستخدمة؛ الأسماء المعطاة في تعريف الدالة تدعى *parameters* (معاملات)، بينما القيم التي تعطى عند استدعاء الدالة تدعى *arguments* (معطيات).

استخدام معاملات الدالة

مثال 7.2. استخدام معاملات الدالة

```
#!/usr/bin/python
# Filename: func_param.py

def printMax(a, b):
```

```
    if a > b:
        print a, 'is maximum'
    else:
        print b, 'is maximum'

printMax(3, 4) # directly give literal values

x = 5
y = 7

printMax(x, y) # give variables as arguments
```

الخرْج

```
$ python func_param.py
4 is maximum
7 is maximum
```

كيف يعمل

هنا، عرفنا دالة باسم printMax حيث نأخذ اثنين من المعاملات تسمى a و b. واستنتجنا العدد الأكبر باستخدام بسيط للإفادة if..else ثم طبعنا العدد الأكبر.

في أول استخدام للدالة printMax، نعطي الأعداد؛ أي المعطيات. في الاستخدام الثاني، نستدعي الدالة باستخدام المتغيرات. (printMax(x, y تؤدي إلى إسناد قيمة المعطى x إلى المعامل a و قيمة المعطى y تسند إلى المعامل b. الدالة printMax تعمل بنفس الطريقة في الحالتين.

المتغيرات المحلية

عندما تقوم بالإعلان عن المتغيرات داخل تعريف الدالة، فإنه لا تكون مرتبطة بأي حال من الأحوال مع المتغيرات الأخرى التي تحمل نفس الاسم خارج تعريف الدالة. أسماء المتغيرات تعتبر محلية (local) داخل الدالة ، وهذا ما يسمى نطاق مدى (scope) المتغير. جميع المتغيرات محدودة بمدى اللبنة التي أُعان عنها فيها، بداية من نقطة تعريف الاسم.

استخدام المتغيرات المحلية

مثال 7.3. استخدام المتغيرات المحلية

```
#!/usr/bin/python
# Filename: func_local.py
```

```
def func(x):
    print 'x is', x
    x = 2
    print 'Changed local x to', x

x = 50
func(x)
print 'x is still', x
```

الخرْج

```
python func_local.py $
x is 50
Changed local x to 2
x is still 50
```

كيف يعمل

في هذه الدالة، المرة الأولى التي تستخدم فيها قيمة الاسم X تستخدم بيثون قيمة المعامل المعلن عنه في الدالة.

بعد ذلك أسندناالقيمة 2 إلى x. الاسم X يعتبر محليا داخل الدالة التي لدينا؛ لذا عندما نغير قيمة X في الدالة، تظل X المعرفة في اللبنة الرئيسة كما هي.

في إفادة print الأخيرة نتأكد من أن قيمة X في اللبنة الرئيسة لم تُمس بالفعل.

استخدام الإفادة global

إذا أردت إسناد قيمة إلى اسم معرّف خارج الدالة، فعليك إخبار بيثون أن الاسم ليس محليا ولكنه عمومي (global). ويتم ذلك باستخدام إفادة global. لا يمكن إسناد قيمة إلى متغير معرف خارج الدالة دون استخدام إفادة global.

يمكنك استخدام قيم المتغيرات المحددة خارج الدالة (بافتراض عدم وجود متغير يحمل نفس الاسم داخل الدالة). غير أنه لا يُنصح بهذا ويفضل تجنبه لأنه يجعل من غير الواضح لقارئ البرنامج أين عُرّف هذا المتغير. استخدام إفادة global يشير بوضوح إلى أن المتغير معرّف في لبنة خارجية.

مثال 7.4. استخدام الإفادة global

```
#!/usr/bin/python
```

```
# Filename: func_global.py

def func():
    global x

    print 'x is', x
    x = 2
    print 'Changed global x to', x

x = 50
func()
print 'Value of x is', x
```

الخرْج

```
$ python func_global.py
x is 50
Changed global x to 2
Value of x is 2
```

كيف يعمل

تستخدم الإفادة global لإعلان أن x متغير عمومي، ولهذا؛ عندما نسند قيمة إلى x داخل الدالة، فإن هذا التغيير يظهر عندما نستخدم قيمة x في اللبنة الرئيسة.

يمكنك تحديد أكثر من متغير global واحد باستخدام نفس الإفادة global. على سبيل المثال : global x, y, z.

القيم المبدئية للمعطيات

قد ترغب في جعل معاملات بعض الدوال *اختيارية* واستخدام قيم مبدئية في حال لم يرغب المستخدم في إعطاء قيم لهذه المعاملات. ويتم ذلك باستخدام قيم مبدئية للمعطيات. يمكنك تحديد القيم المبدئية للمعطيات بإتباع اسم المعامل في تعريف الدالة بعلامة (=) متبوعة بالقيمة المبدئية.

علما بأن القيمة المبدئية للمعطى ينبغي أن تكون ثابتا. وسيتم شرح ذلك بشيء أكثر من التفصيل في فصول لاحقة. اما الآن، فقط تذكر هذا.

استخدام القيم المبدئية للمعطيات

مثال 7.5. استخدام القيم المبدئية للمعطيات

```
#!/usr/bin/python
# Filename: func_default.py

def say(message, times = 1):
    print message * times

say('Hello')
say('World', 5)
```

الخرْج

```
python func_default.py $
Hello
WorldWorldWorldWorldWorld
```

كيف يعمل

تستخدم الدالة المسماة say لطباعة سلسلة ما عددا من المرات حسبمانريد. وإذا لم نعطها أية قيمة، فالوضع الافتراضي هو طباعة الجملة لمرة واحدة فقط. نحقق ذلك عن طريق جعل قيمة المعطى المبدئية للمعامل times تساوي 1.

في أول استخدام للدالة say، نعطي النص فقط فتقوم هي بطباعة الجملة مرة واحدة. في المرة الثانية نعطيها كلا من النص والمعطى 5 والذي يعني أننا نريد *قول* الجملة خمس مرات.

هام

المعاملات التي في نهاية قائمة المعاملات فقط يمكن أن تُعطى قيم معطيات مبدئية؛ بعبارة أخرى لا يمكن إعطاء قيمة معطى مبدئية لمعامل قبل معامل بدون قيمة معطى مبدئية.

وذلك بسبب أن القيم تسند إلى المعاملات حسب وضعيتها. على سبيل المثال: def func(a, b=5) صالحة بينما def func(a=5, b) *غير صالحة*.

معطيات الكلمات المفتاحية

إذا كان لديك بعض الدوال ذات العديد من المعاملات وتريد أن تحدد بعضا منها فقط، حينئذ يمكنك أن تعطي قيما لهذا المعاملات عن طريق تسميتها - وهذا ما يسمى*معطيات الكلمات المفتاحية* (keyword arguments) - نستخدم الاسم (الكلمة المفتاحية) بدلا من الموضع

(الذي كنا نستخدمه طوال الوقت) لتحديد معطيات الدالة.

ولهذا *ميزتان*: الأولى؛ استخدام الدالة يكون أسهل حيث أننا لسنا في حاجة إلى الاهتمام بترتيب المعطيات. والثانية؛ أننا نستطيع إعطاء قيم للوسائط التي نريدها فقط، بافتراض أن للمعاملات الأخرى قيم معطيات مبدئية.

استخدام معطيات الكلمات المفتاحية

مثال 7.6. استخدام معطيات الكلمات المفتاحية

```
#!/usr/bin/python
# Filename: func_key.py

def func(a, b=5, c=10):
    print 'a is', a, 'and b is', b, 'and c is', c

func(3, 7)
func(25, c=24)
func(c=50, a=100)
```

الخرْج

```
$ python func_key.py
a is 3 and b is 7 and c is 10
a is 25 and b is 5 and c is 24
a is 100 and b is 5 and c is 50
```

كيف يعمل

الدالة المسماه func تحتوي على معامل واحد بدون قيم مبدئية لمعطياته، يليه معاملان مع قيم مبدئية لمعطياتهما.

في الاستخدام الأول (func(3, 7)، المعامل a يأخذ القيمة 3، والمعامل b يأخذ القيمة 5 والمعامل c يأخذ القيمة المبدئية 10.

في الاستخدام الثاني (func(25, c=24 المتغير a يأخذ القيمة 25 بسبب موضع المعطى. بعدها المعامل c يحصل على القيمة 24 بينما المتغير b يحصل على قيمته المبدئية 5.

في الاستخدام الثالث (func(c=50, a=100؛ استخدمنا بشكل كامل معطيات بكلمات مفتاحية لتحديد القيم. لاحظ أن القيمة المحددة للمعامل c موضوعة قبل a على رغم أننا حددنا a قبل c أثناء تعريف الدالة.

الإفادة return

تستخدم الإفادة return في *الرجوع* من دالة، بعبارة أخرى الخروج من الدالة . يمكننا اختياريا *إرجاع قيمة* من الدالة أيضا.

استخدام الإفادة return

مثال 7.7. استخدام الإفادة return

```
#!/usr/bin/python
# Filename: func_return.py

def maximum(x, y):
    if x > y:
        return x
    else:
        return y

print maximum(2, 3)
```

الخرْج

```
$ python func_return.py
```

كيف يعمل

الدالة maximum ترجع لنا الحد الأكبر من المعاملات، وهي في هذه الحالة الأعداد المعطاة للدالة. وهي تستعمل إفادة if..else بسيطة للعثور على قيمة العدد الأكبر وبعدها ترجع لنا تلك القيمة.

ملاحظة :عندما تكون إفادة return بدون أية قيمة فإنها تساوي None. تعتبر None نوعا خاصا في بيثون يمثل لا شيء. فهي على سبيل المثال تستخدم لتعني أن المتغير لا يحمل أي قيمة إذا كانت قيمته None.

كل دالة تحتوي ضمنيا على إفادة return None في نهايتها ما لم تكتب إفادة return الخاصة بك. ويمكنك رؤية ذلك بتنفيذ print someFunction literal>()

```
def someFunction():
    pass
```

إفادة pass

تستخدم في بيثون للإشارة إلى لبنة فارغة من الإفادات.

جمل التوثيق

بيثون لديها ميزة أنيقة تدعى جمل *التوثيق (سلاسل)* (documentation strings) والتي يشار إليها عادة من خلال اسمها المختصر *docstrings*. جمل التوثيق أداة هامة يجب عليك أن تستفيد منها حيث إنها تساعد على توثيق البرنامج بشكل أفضل، وتجعله أكثر سهولة للفهم. يمكننا حتى -ويا للعجب- من الحصول على جمل التوثيق من دالة مثلا بينما يعمل البرنامج بالفعل!

استخدام جمل التوثيق

مثال 7.8. استخدام جمل التوثيق

```
#!/usr/bin/python
# Filename: func_doc.py

def printMax(x, y):
    '''Prints the maximum of two numbers.

    The two values must be integers.'''
    x = int(x) # convert to integers, if possible
    y = int(y)

    if x > y:
        print x, 'is maximum'
    else:
        print y, 'is maximum'

printMax(3, 5)
print printMax.__doc__
```

الخرْج

```
$ python func_doc.py
```

```
5 is maximum
Prints the maximum of two numbers.

    The two values must be integers.
```

كيف يعمل

الجملة النصية في السطر المنطقي الأول من الدالة هي *جملة توثيق* هذه الدالة. لاحظ أن جمل التوثيق تنطبق أيضا على الوحدات و الأصناف والتي سوف نقوم بشرحها في فصولها الخاصة.

العادة المتبعة في جما التوثيق هي استخدام سلسلة نصية متعددة الأسطر حيث أول سطر يبدأ بحرف كبير وينتهي بنقطة. بعد ذلك السطر الثاني فارغ متبوعا بجملة من الشرح المفصل يبدأ في السطر الثالث. *ينصح بشدة* أن تتبع هذا النظام في كل جمل التوثيق لكل دوالك غير البديهية.

يمكننا الوصول إلى جملة توثيق الدالة printMax باستخدام خاصية __doc__ (لاحظ الشرطة المنخفضة المزدوجة) الخاصة بالدالة. فقط تذكر أن بيثون تعامل *كل شيء* على أنه كائن، وهذا يشمل الدوال أيضا. وسوف نتعلم المزيد عن الكائنات في الفصل المتعلق بالطبقات.

اذا كنت قد استخدمت help() في بيثون ، فلابد أنك رأيت بالفعل استخدام جمل التوثيق. كل ما تفعله هو جلب خاصية __doc__ المنتمية لهذه الدالة وتعرضها لك بأسلوب أنيق. يمكنك تجربة ذلك على الدالة المبينة أعلاه - فقط أضِف help(printMax) في برنامجك. وتذكر أن تضغط مفتاح q للخروج من المساعدة.

توجد الأدوات مأتمتة يمكنها استحضار وثائق برنامجك بذات الطريقة. لذا، فإنني *أنصح بشدة* أن تستخدم جمل توثيق لأي دالة غير بديهية تكتبها. الأمر pydoc الذي يأتي مع توزيعة بيثون يعمل كمثل help() مستخدما جمل التوثيق.

الخلاصة

لقد رأينا الكثير من الجوانب المتعلقة بالدوال، ولكن لاحظ أننا لم نغطي كافة جوانبها. ورغم ذلك فقد قمنا بالفعل بتغطية معظم الأمور التي ستستخدمها كل يوم من ناحية دوال بيثون.

وفيما يلي؛ سوف نرى كيف نقوم باستخدام وإنشاء وحدات بيثون.

فصل 8. الوحدات

قائمة المحتويات

مقدمة

قد رأيت كيف يمكنك إعادة استخدام الكود في برنامجك عن طريق تعريف الدوال مرة واحدة. ماذا لو أردت إعادة استخدام عدد من الدوال في البرامج الأخرى التي تكتبها؟ نعم كما قد خمنت، الجواب هو الوحدات (modules). الوحدة ببساطة هي ملف يحتوي جميع الدوال والمتغيرات التي قمت بتعريفها. ولإعادة استخدام هذه الوحدة في برامج أخرى، يجب أن يكون اسم ملف الوحدة بامتداد .py.

يمكن للبرامج الأخرى *استيراد* الوحدة للاستفادة من وظيفتها. وهي نفس الطريقة التي أمكننا بها استخدام مكتبة بيثون القياسية. أولا، سوف نرى كيفية استخدام وحدات المكتبة القياسيه.

استخدام الوحدة sys

إفادة sys

مثال 8.1. استخدام الوحدة sys

```
#!/usr/bin/python
# Filename: using_sys.py
import sys
print 'The command line arguments are:'
for i in sys.argv:
```

```
    print i
print '\n\nThe PYTHONPATH is', sys.path, '\n'
```

الخرْج

```
$ python using_sys.py we are arguments
The command line arguments are:
using_sys.py
we
are
arguments


The PYTHONPATH is ['/home/swaroop/byte/code', '/usr/lib/python23.zip',
'/usr/lib/python2.3', '/usr/lib/python2.3/plat-linux2',
'/usr/lib/python2.3/lib-tk', '/usr/lib/python2.3/lib-dynload',
'/usr/lib/python2.3/site-packages', '/usr/lib/python2.3/site-packages/gtk-2.0']
```

كيف يعمل

في البداية قمنا *باستيراد* الوحدة sys باستخدام الإفادة import. مبدئيا، يعني هذا إخبار بيثون أننا نريد استخدام تلك الوحدة. الوحدة sys تحتوي على وظائف ترتبط بمفسر بيثون والبيئة الخاصة به.

عندما تنفذ بيثون إفادة import sys، فإنها تبحث عن الوحدة sys.py في أحد الأدلة المسرودة في المتغير sys.path. فإذا وجد الملف يتم تشغيل الإفادات الموجودة في اللبنة الرئيسة الخاصة بالوحدة وبعدها تصبح الوحدة *متاحة* للاستعمال. لاحظ أن هذا الاستبداء يتم فقط عند *أول* مرة تستدعى فيها الوحدة. كذلك اعلم أن 'sys' اختصار 'system' (نظام).

المتغير argv في الوحدة sys يشار إليه باستخدام الترقيم النقطي -sys.argv- أحد مميزات هذا الأسلوب أن الاسم لا يشتبه مع أية متغير argv آخر مستخدم في برنامجك. و كذلك فإنه يدل بوضوح على كون ذلك الاسم جزءا من الوحدة sys.

المتغير sys.argv عبارة عن *قائمة* من السلاسل (ستشرح القوائم بالتفصيل في أقسام لاحقة). وبشكل أكثر تحديدا فإن sys.argv يحتوي على قائمة*معطيات سطر الأوامر* بعبارة أخرى، المعطيات الممررة إلى برنامجك في سطر الأوامر.

إذا كنت تستخدم بيئة تطوير لكتابة وتشغيل برامجك فابحث عن طريقة لتحديد معطيات سطر الأوامر لبرنامجك في قوائم بيئة التطوير.

وهنا؛ عندما ننفذ python using_sys.py we are arguments، نقوم بتشغيل الوحدة

using_sys.py باستخدام الأمر **python** والأشياء الأخرى التالية له معطيات يتم تمريرها إلى البرنامج. تقوم بيثون بتخزينها في المتغير sys.argv.

تذكر أن اسم البرنامج الذي يعمل هو دائما أول معطى في قائمة sys.argv. لذا في هذه الحالة سيكون لدينا 'using_sys.py' ك sys.argv[0] و 'we' ك sys.argv[1] و 'are' ك sys.argv[2] و 'arguments' ك sys.argv[3]. لاحظ أن بيثون تبدأ العدّ من 0 وليس من 1.

يحتوي sys.path على قائمة بأسماء الأدلة التي يتم استيراد الوحدات منها. مع ملاحظة أن أول سلسلة في sys.pathفارغة – هذه السلسلة الفارغة تشير إلى أن الدليل الحالي هو كذلك جزء من sys.path، والذي هو نفسه متغير البيئة PYTHONPATH. يعني هذا أن بإمكانك استيراد الوحدات الموجودة في الدليل الحالي مباشرة. وإلا فسيكون عليك أن تضع الوحدة في أحد هذه الأدلة المسرودة في sys.path.

ملفات البيتات المصرّفة .pyc

استيراد وحدة أمر مكلّف نسبيا، لذا فإن بيثون تقوم ببعض الحيل لجعلها تعمل بشكل أسرع. أحد هذه الطرق هي إنشاء ما يسمى بملفات *البيتات المصرفة* (byte-compiled) ذات الامتداد.pyc الذي يرتبط بالصورة البينية التي تحول بيثون البرامج إليها (هل تذكر المقدمة في طريقة عمل بيثون؟). هذا الملف .pyc مفيد عندما تستورد الوحدة في المرة التالية من برنامج مختلف - وسيكون أكثر سرعة حيث أن جزء المعالجة المطلوب أثناء استيراد الوحدة قد تم عمله بالفعل. أيضا، ملفات البيتات المصرّفة هذه مستقلة عن منصة العمل. وبذلك نكون قد عرفنا ما هي ملفات .pyc.

الإفادة from..import

إذا أردت استيراد المتغير argv داخل برنامجك (لتجنب كتابة sys. كل مرة ت ريده)، فيمكنك استخدام إفادة from sys import argv. إذا أردت استيراد كل الأسماء المستخدمة في الوحدة sys يمكنك استخدام إفادة from sys import *. وهذا يعمل مع أي وحدة. وبوجه عام تجنب استخدام عبارة from..import واستخدم بدلا منها عبارة import، حيث أن البرنامج سيكون بهذه الطريقة أكثر سهولة في قراءته، وسوف تتجنب أي اشتباه في الأسماء بهذه الطريقة.

خاصية __name__ للوحدة

كل نموذج له اسم وإفادات في الوحدة يمكن اكتشاف اسم الوحدة الخاص بها. وذلك أمر متيسر بصفة خاصة في حالة بعينها- كما ذكر سابقا ، عندما ويتم استيراد الوحدة لأول مرة ، يتم تشغيل اللبنة الرئيسة في تلك الوحدة . ماذا لو كنا

نريد تشغيل اللبنة فقط إذا كان البرنامج نفسه مستخدما وليس عند استيراده من وحدة أخرى؟ ويمكن تحقيق ذلك باستخدام السمة __name__ من الوحدة.

استخدام الوحدة __name__

مثال 8.2 على استخدام خاصية الوحدة __name__

```
 usr/bin/python/!#

# Filename: using_name.py

if __name__ == '__main__':

    print 'This program is being run by itself'

else:

    print 'I am being imported from another module'
```

الخـــرج

```
$ python using_name.py

This program is being run by itself

$ python

>>> import using_name

I am being imported from another module
```

كيف يعمل

كل وحدة في بيثون لها _name_ معرف، فإن كان ذلك هو __main__ فذلك يعني أن الوحدة ستعمل بذاتها من قِبل المستخدم ويمكننا أن نقوم بما يناسب ذلك من أحداث.

عمل وحداتك الخاصة

إنشاء وحدات خاصة بك أمر سهل، وقد فعلت هذا طوال الوقت. كل برنامج بيثون يعتبر وحدة في نفس الوقت. فقط عليك التأكد من احتوائه على امتداد .py. والمثال التالي سيوضح لك ذلك.

إنشاء وحداتك الخاصة

مثال 8.3. كيف تنشئ وحداتك الخاصة

```
#!/usr/bin/python
# Filename: mymodule.py

def sayhi():
    print 'Hi, this is mymodule speaking.'

version = '0.1'

# End of mymodule.py
```

هذا البرنامج المبين أعلاه هو وحدة بسيطة. وكما ترى لا يوجد شيء مميز بالمقارنة بما اعتدناه في برامج بيثون. وفيما يلي سنرى كيف نستخدم هذه الوحدة في برامج بيثون الأخرى.

تذكر أن هذه الوحدة يجب أن توضع في نفس الدليل الذي يعمل منه البرنامج، أو تكون في أحد الأدلة المسرودة في قائمة sys.path.

```
#!/usr/bin/python
# Filename: mymodule_demo.py

import mymodule

mymodule.sayhi()
print 'Version', mymodule.version
```

الخرْج

```
$ python mymodule_demo.py
Hi, this is mymodule speaking.
Version 0.1
```

كيف يعمل

لاحظ أننا نستخدم نفس الترقيم النقطي للوصول إلى عناصر الوحدة. تجيد بيثون إعادة استخدم نفس الترقيم لإضفاء الشعور 'البايثوني' المميز، لذا لا يجب علينا أن نظل نتعلم طرقًا جديدة لعمل الأشياء.

إفادة from..import

ههنا نسخة تستخدم الصيغة from..import

```
#!/usr/bin/python
# Filename: mymodule_demo2.py
from mymodule import sayhi, version
# Alternative:
# from mymodule import *
sayhi()
print 'Version', version
```

خرْج mymodule_demo2.py هو نفسه خرْج mymodule_demo.py.

الدالة dir()

يمكنك استخدام الدالة المدمجة dir لسرد المعرفات تحددها الوحدة. هذه المعرفات هي الدوال، والمتغيرات، والأصناف المعرفة في الوحدة.

عند إعطاء اسم وحدة للدالة dir()، فإنها تعيد لنا قائمة الأسماء المعرفة في تلك الوحدة. وعند عدم إعطائها أية معطيات فإنها تعيد لنا قائمة بالأسماء المعرفة في الوحدة الحالية.

استخدام الدالة dir

مثال 8.4. استخدام الدالة dir

```
$ python
>>> import sys
>>> dir(sys) # get list of attributes for sys module
['__displayhook__', '__doc__', '__excepthook__', '__name__', '__stderr__',
'__stdin__', '__stdout__', '_getframe', 'api_version', 'argv',
'builtin_module_names', 'byteorder', 'call_tracing', 'callstats',
'copyright', 'displayhook', 'exc_clear', 'exc_info', 'exc_type',
'excepthook', 'exec_prefix', 'executable', 'exit', 'getcheckinterval',
```

```
'getdefaultencoding', 'getdlopenflags', 'getfilesystemencoding',
'getrecursionlimit', 'getrefcount', 'hexversion', 'maxint', 'maxunicode',
'meta_path','modules', 'path', 'path_hooks', 'path_importer_cache',
'platform', 'prefix', 'ps1', 'ps2', 'setcheckinterval', 'setdlopenflags',
'setprofile', 'setrecursionlimit', 'settrace', 'stderr', 'stdin', 'stdout',
'version', 'version_info', 'warnoptions']
>>> dir() # get list of attributes for current module
['__builtins__', '__doc__', '__name__', 'sys']
>>>
>>> a = 5 # create a new variable 'a'
>>> dir()
['__builtins__', '__doc__', '__name__', 'a', 'sys']
>>>
>>> del a # delete/remove a name
>>>
>>> dir()
['__builtins__', '__doc__', '__name__', 'sys']
>>>
```

كيف يعمل

أولا، نرى استخدام الدالة dir مع الوحدة المستوردة sys. نستطيع أن نرى قائمة ضخمة من العناصر المشتملة عليها. بعدها قمنا باستخدام الدالة dir بدون تمرير أي معاملات إليها لذا فهي تعيد إلينا قائمة العناصر المنتمية للوحدة الحالية. لاحظ أن قائمة الوحدات المستوردة جزء من هذه القائمة أيضا.

ومن أجل ملاحظة dir وهي تعمل، قمنا بتعريف متغير جديد a وأسندنا إليه قيمة وبعدها نقوم بفحص dir وسنلاحظ أن هناك قيمة مضافة إلى القائمة بنفس الاسم. نقوم بحذف خصائص/متغيرات الوحدة الحالية باستخدام إفادة del وسوف ينعكس هذا التغير مرة أخرى في خرْج الدالة dir.

ملاحظة على del: تستخدم هذه الإفادة لحذف اسم أو متغير وبعد تشغيل الإفادة -وهي في هذه الحالة del a- لا يعود بإمكاننا الوصول إلى المتغير a؛ وكأنه لم يكن موجودا من قبل.

الخلاصة

الوحدات مفيدة لأنها تمدك بخدمات ووظائف يمكننا إعادة استخدامها في البرامج الأخرى.
والمكتبة القياسية التي تأتي مع بيثون تعتبر مثال على الوحدات. وقد رأينا كيف نستخدم هذه الوحدات وإنشاء الوحدات الخاصة بنا أيضا.

وفيما يلي سوف نتعلم بعض المفاهيم المهمة والتي تدعى هياكل البيانات.

فصل 9. هياكل البيانات

قائمة المحتويات

مقدمـــة

هياكل البيانات هي ببساطة كما يظهر من اسمها؛ *هياكل* يمكنها حمل بعض *البيانات* معا. وبعباره أخرى، فهي تستخدم لتخزين مجموعة من البيانات ذات الصلة.

وهناك ثلاثة أنواع من هياكل البيانات مدمجة في بيثون -- القوائم (list)، والصفوف (tuple)، والقواميس (dictionary). وسنرى كيفية استخدام كل منها، وكيف أنها تجعل الحياة أسهل.

القائمة

القائمة أحد هياكل البيانات التي تحمل مجموعة من العناصر المرتّبة، بعبارة أخرى يمكنك أن تخزن *تتابعا* من العناصر في قائمة. سيسهل عليك تصور هذا أن تفكر في قائمة التسوق حيث لديك عناصر تعدها للشراء، فيما عدا أنك ربما تضع كل عنصر في سطر منفصل في قائمة التسوق، في حين أن بيثون تضع فاصلة فيما بينها.

قائمة العناصر ينبغي أن تكون بين قوسين مربعين [ ] حتى تفهم بيثون أنك تريد تحديد قائمة. بمجرد إنشاء قائمة، يمكنك إضافة أو إزالة أو البحث عن العناصر في القائمة. وحيث أننا نستطيع إضافة و حذف العناصر، نقول أن القائمة هي نوع بيانات متغير أي أن هذا النوع يمكن تغييره.

مقدمة سريعة إلى الكائنات والفئات

رغم أنني قد كنت أخرت وحتى الآن مناقشة الكائنات والفئات بوجه عام، إلا أننا بحاجة لشرح صغير الآن حتى تتمكن من فهم القوائم بصورة أفضل. سنشرح هذا الموضوع بالتفصيل في الفصل الخاص به.

القائمة مثال على استخدام الكائنات والفئات. عندما تستخدم متغير i وتعطيه قيمة - مثلا العدد الصحيح 5 - فإنك تنظر لها على أنه إنشاء **للكائن** (سيرورة) i من **الفئة** (النوع) int. في الحقيقة تستطيع مطالعة help(int) لتفهم هذا أكثر.

ويمكن للفئة أن تشتمل على **طرق**؛ دوال معرّفة للاستخدام مع هذه الفئة فقط. يمكنك استعمال هذه الأجزاء فقط عندما يكون لديك كائن منهذه الفئة. على سبيل المثال؛ توفر بيثون الطريقة append للفئة list التي تمكنك من إضافة عنصر إلى نهاية القائمة. مثلا mylist.append('an item') ستضيف هذه السلسلة إلى القائمة mylist.append('an item'). لاحظ الترقيم النقطي للوصول إلى أساليب الكائن.

الفئة يمكن أن يكون لها أيضا **حقول** (fields) وهي ليست سوى متغيرات معرّفة لاستخدامها فيما يخص تلك الفئة فقط. يمكنك استخدام هذه المتغيرات/الأسماء فقط عندما يكون لديك كائن من تلك الفئة. الحقول أيضا متاحة من خلال الترقيم النقطي، مثلا، mylist.field .

استخدام القوائم

مثال 9.1. استخدام القوائم

```
#!/usr/bin/python
# Filename: using_list.py

# This is my shopping list
shoplist = ['apple', 'mango', 'carrot', 'banana']

print 'I have', len(shoplist), 'items to purchase.'

print 'These items are:', # Notice the comma at end of the line
for item in shoplist:
```

```
    print item,

print '\nI also have to buy rice.'
shoplist.append('rice')
print 'My shopping list is now', shoplist

print 'I will sort my list now'
shoplist.sort()
print 'Sorted shopping list is', shoplist

print 'The first item I will buy is', shoplist[0]
olditem = shoplist[0]
del shoplist[0]
print 'I bought the', olditem
print 'My shopping list is now', shoplist
```

الخرْج

```
$ python using_list.py
I have 4 items to purchase.
These items are: apple mango carrot banana
I also have to buy rice.
My shopping list is now ['apple', 'mango', 'carrot', 'banana', 'rice']
I will sort my list now
Sorted shopping list is ['apple', 'banana', 'carrot', 'mango', 'rice']
The first item I will buy is apple
I bought the apple
My shopping list is now ['banana', 'carrot', 'mango', 'rice']
```

كيف يعمل

المتغير shoplist عبارة عن قائمة تسوق لشخص ذاهب إلى السوق. في shoplist نقوم بتخزين سلاسل النصية بأسماء العناصر التي سيشتريها، لكن تذكر أن بإمكانك إضافة أي نوع من *الكائنات* إلى القائمة بما في ذلك الأعداد وحتى القوائم الأخرى.

كما استخدمنا أيضا الحلقة for..in للتنقل خلال عناصر القائمة. الآن، لابد أنك أدركت أن القائمة هي تتابع أيضا. وسوف نناقش خصوصية التتابعات في قسم لاحق.

لاحظ أننا نستخدم *فاصلة* "," في نهاية إفادة print لمنع الطباعة التلقائية لفاصل الأسطر بعد كل إفادة print. قد تعتبر هذه طريقة سيئة لفعل ذلك، ولكنها بسيطة وتنجز المهمة.

بعدها، أضفنا عنصرا إلى القائمة باستخدام طريقة الإرفاق append في الكائنات من فئة list، كما سبق أن ناقشنا من قبل. ثم، نتحقق من أن العنصر قد تم إضافته إلى القائمة عن طريق طبع محتويات القائمة بتمريرها ببساطة إلى إفادة print التي تطبعها لنا بطريقه أنيقة.

ثم نقوم بترتيب القائمة باستخدام الطريقة sort. لاحظ أن هذه الطريقة تؤثر على القائمة نفسها ولا تعيد قائمة معدلة -- وهذا يختلف عن طريقة عمل السلاسل النصية. وهذا ما نعنيه بقولنا إن القوائم *متغيرة* والسلاسل النصية *ثابتة*.

ثم، عندما ننتهي من شراء بند من السوق، نريد إزالته من القائمة. نحقق ذلك عن طريق استخدام الإفادة del. هنا، نحدد عنصر القائمة الذي نريد إزالته وتقوم إفادة del بحذفه لنا. نحدد أننا نريد إزالة البند الأول من القائمة، وبالتالي نستخدم del shoplist[0] (تذكر أن بيثون تبدأ العد من 0).

إذا أردت معرفة كل الطرق التي يُعرّفها كائن القائمة فطالع help(list) لترى التفاصيل الكاملة.

الصف

الصفوف (Tuples) مثل القوائم إلا أنها **ثابتة** مثل السلاسل النصية؛ أي لا يمكنك تغيير الصفوف. تُعرّف الصفوف عن طريق تحديد عناصر مفصولة بفاصلة داخل زوج من الأقواس ( ). تستخدم الصفوف عادة في الحالات التي تستطيع فيها الإفادة أو الدالة التي يُعرّفها المستخدم أن تفترض بأمان أن مجموعة القيم - صف القيم - المستخدمة لن تتغير.

استخدام الصفوف

مثال 9.2. استخدام الصفوف

```
#!/usr/bin/python
# Filename: using_tuple.py

zoo = ('wolf', 'elephant', 'penguin')
print 'Number of animals in the zoo is', len(zoo)

new_zoo = ('monkey', 'dolphin', zoo)
print 'Number of animals in the new zoo is', len(new_zoo)
print 'All animals in new zoo are', new_zoo
print 'Animals brought from old zoo are', new_zoo[2]
print 'Last animal brought from old zoo is', new_zoo[2][2]
```

الخرْج

```
$ python using_tuple.py
Number of animals in the zoo is 3
Number of animals in the new zoo is 3
All animals in new zoo are ('monkey', 'dolphin', ('wolf', 'elephant', 'penguin'))
Animals brought from old zoo are ('wolf', 'elephant', 'penguin')
Last animal brought from old zoo is penguin
```

كيف يعمل

المتغير zoo يشير إلى صف من العناصر. نرى أن الدالة len يمكن استخدامها للحصول على طول الصف. وهذا يدل أيضا على أن الصفوف هي تتابعات أيضا.

الآن ننقل هذه الحيوانات إلى حديقة حيوانات جديدة (new_zoo) حيث إن الحديقة القديمة (zoo) يتم إغلاقها. لذا يحتوي الصف الجديد new_zoo على بعض الحيوانات الموجودة بالفعل جنبا إلى جنب مع الحيوانات التي جلبت من الحديقة القديمة. عودة إلى الواقع، نلاحظ أن الصف داخل صف آخر لا يفقد هويته

يمكننا الوصول إلى العناصر في الصف عن طريق تحديد موضع العنصر داخل زوج من الأقواس المربعة [ ] مثل ما فعلنا مع القوائم. وهذا ما يسمى *عامل الفهرسة* (indexing operator). نستطيع الوصول إلى البند الثالث في new_zoo بتحديد new_zoo [2] ، ويمكن الوصول إلى البند الثالث في الصف new_zoo بتحديد new_zoo [2] [2]. هذا واضح جدا ، بمجرد أن تفهم الطريقة.

صف بدون عناصر أو عنصر واحد. يُنشأ صف فارغ عن طريق زوج من الأقواس الخالية () مثل myempty =() . لكن تحديد صف ذي عنصر واحد ليس بهذه البساطة. عليك أن تحدده مستخدما فاصلة "," بعد العنصر الأول (والوحيد)؛حتى تستطيع بيثون التفرقة بين صف وبين زوج من الأقواس المحيطة بكائن داخل تعبير. أي عليك أن تحدد singleton = (2 ,) إذا كنت تعني أنك تريد صفا يحتوي العنصر 2.

ملاحظة لمبرمجي بيرل

القائمة داخل قائمة لا تفقد هويتها .فمثلا القوائم غير منبسطة كما هو في بيرل . نفس الأمر ينطبق على صف داخل صف ، أو الصف داخل قائمة، أو قائمة داخل صف إلخ . فبقدر ما يتعلق الأمر ببايثون ، فإنه فقط عبارة عن كائنات مخزنة باستخدام كائن آخر، هذا كل ما في الموضوع .

الصفوف وإفادة print

واحدة من أكثر الاستعمالات الشائعة هي الصفوف مع الإفادة print. وإليك هذا المثال

مثال 9.3 استخدام الصفوف

```
#!/usr/bin/python
# Filename: print_tuple.py

age = 22
name = 'Swaroop'

print '%s is %d years old' % (name, age)
print 'Why is %s playing with that python?' % name
```

الخرج

```
$ python print_tuple.py
Swaroop is 22 years old
Why is Swaroop playing with that python?
```

كيف يعمل البرنامج

- إفادة print يمكن ان تأخذ سلسلة نصية باستخدام مواصفات معينة يتبعها الرمز % يليها tuple من البنود المطابق للمواصفات. المواصفات تستخدم في صياغه النتائج بطريقة معينة. المواصفات يمكن أن تكون على غرار s% للسلاسل النصية strings و d% للأعداد الصحيحة. الصف يجب أن يحتوي على بنود مقابلة لهذه المواصفات في نفس النظام .
- لاحظ أن أول استعمال حيث نستخدم s% أولا وهذا مطابقا لاسم المتغير الذي هو البند الأول في الصف ، والوصف الثاني هو d% المقابل لل age الذي هو البند الثاني في الصف.
- ما يعمله بيثون هنا هو أنه يحول كل بند في الصف إلى سلسلة نصية وبدائل لقيمة هذه السلسلة داخل مكان المواصفات. لذا s% هو استعاضة عن قيمة المتغير name وهلم جرا.
- هذا الاستخدام للإفادة print يجعل من السهل للغاية كتابة الناتج ويتجنب الكثير من التلاعب بالنص لتحقيق ذات الأمر. كما انه يتجنب استعمال الفواصل في كل مكان كما فعلنا حتى الآن.
- معظم الوقت ، يمكنك استخدام الوصف s%. واترك لبايثون العناية بالباقي من أجلك. وهذا يعمل حتى مع الأرقام. ومع ذلك ، قد ترغب في إعطاء المواصفات الصحيحة حيث إن هذا

يضيف مستوى واحداً من التأكد من صحة برنامجك .

- في الإفادة الثانية print، نستخدم أحد المواصفات التي يتبعها الرمز % يليه بند واحد -- لا يوجد زوج من الأقواس. هذا يعمل فقط في حالة عندما يكون هناك وصف واحد في السلسلة النصية.

القاموس

القاموس هو بمثابة كتاب عنونة حيث يمكنك أن تجد عنوانا أو تفاصيل للاتصال مع شخص عن طريق معرفه اسمه /اسمها . مثلا ؛ نحن نتشارك المفاتيح (الاسم) مع القيم (التفاصيل). علما بأن المفتاح يجب أن يكون فريدا unique حيث أنه لا يمكنك الحصول على معلومات صحيحة إذا كان لديك شخصان بنفس الاسم بالضبط .

علما بأنه يمكنك استخدام كائنات ثابتة فقط (مثل السلاسل النصية) لمفاتيح القاموس ولكن يمكنك استخدام كائنات ثابتة أو قابلة للتغيير لقيم القاموس يمكننا أن نترجم ذلك بقولنا أنه ينبغي أن لا تستخدم سوى أشياء بسيطة للمفاتيح.

زوج من المفاتيح والقيم المذكورة في القاموس باستخدام العبارة
d = {key1 : value1, key2 : value2} لاحظ أن أزواج المفتاح/القيمة منفصلان عن طريق النقطتين":" والأزواج أنفسهم منفصلان عن طريق فاصلة , و كل هذا داخل في زوج من الأقواس المجعدة { } .
تذكر أن أزواج key/value في القاموس ليست لها أي طريقة ترتيب. إذا أردت ترتيبا معينا ، سيتعين عليك ترتيبها بنفسك قبل استعمالها.

القواميس التي ستقوم باستخدامها تعتبر حالات/كائنات instances/objects من الطبقة "dict" .

استخدام القواميس

مثال 9.4 استخدام القواميس

```
#!/usr/bin/python
# Filename: using_dict.py

# 'ab' is short for 'a'ddress'b'ook
```

```
ab = {              'Swaroop'   : 'swaroopch@byteofpython.info',
            'Larry'     : 'larry@wall.org',
            'Matsumoto' : 'matz@ruby-lang.org',
            'Spammer'   : 'spammer@hotmail.com'
        }

print "Swaroop's address is %s" % ab['Swaroop']

# Adding a key/value pair
ab['Guido'] = 'guido@python.org'

# Deleting a key/value pair
del ab['Spammer']

print '\nThere are %d contacts in the address-book\n' % len(ab)

for name, address in ab.items():
        print 'Contact %s at %s' % (name, address)

if 'Guido' in ab: # OR ab.has_key('Guido')
        print "\nGuido's address is %s" % ab['Guido']
```

الخرج

```
$ python using_dict.py
Swaroop's address is swaroopch@byteofpython.info

There are 4 contacts in the address-book

Contact Swaroop at swaroopch@byteofpython.info
Contact Matsumoto at matz@ruby-lang.org
```

Contact Larry at larry@wall.org

Contact Guido at guido@python.org

Guido's address is guido@python.org

كيف يعمل البرنامج

باستخدام الترقيم الذي سبق مناقشته. ثم شغلنا أزواج مفتاح/قيمة ab قمنا بصنع القاموس key/value من خلال تحديد المفتاح باستخدام عامل الفهرسة indexing operator كما نوقش في الكلام عن القوائم و الصفوف. نلاحظ أن التركيب بسيط جدا للقواميس كذلك.

ويمكننا أن نضيف أزواج جديدة من مفتاح/قيمة ببساطة عن طريق استخدام عامل الفهرسة للوصول إلى مفتاح وإسناد قيمة إليه ، كما فعلنا ل Guido في الحالة المذكورة أعلاه.

نحن ببساطة نحدد .del يمكننا حذف أزواج المفتاح/القيمة باستخدام صديقنا القديم الإفادة لإزالة المفتاح وتمرير ذلك إلى الإفادة indexing operator القاموس و عامل الفهرسة del. ليست هناك حاجة لمعرفة القيمة المقابلة للمفتاح في هذه العملية.

من items في القاموس باستخدام طريقة key/value بعد ذلك نصل إلى كل زوج من القاموس التي تعيد قائمة من الصف حيث كل صف يحتوي زوجا من البنود – والمفتاح متبوعا بقيمة. نسحب هذا الزوج ونسنده إلى اسم المتغيرات والعنوان المقابل لكل زوج for. ثم نطبع هذه القيم في اللبنة ، for..in باستخدام الحلقة
أو حتى طريقة in موجود باستخدام العامل key/value يمكننا معرفة ما اذا كان زوج تستطيع أن ترى الوثائق للاطلاع على القائمة الكاملة للطرق من dict من الفئة has_key الفئة dict باستخدام (help(dict

الكلمات المفتاحية للمعطيات والقواميس

على صعيد آخر نلاحظ ، أنك إن كنت قد استخدمت الكلمات المفتاحية للمعطياتا في الدوال الخاصة بك ، ولقد سبق أن استخدمت قواميس! فقط فكر في ذلك – زوج مفتاح/قيمة محدد من قِبلِك في قائمة معاملات تعريف الدالة ، وعند تشغيل المتغيرات بداخل الدالة ، وهو مجرد مفتاح الوصول إلى القاموس (وهو ما يسمى symbol table في مصطلح تصميم المصرف) .

المتسلسلات

الصفوف والقوائم والسلاسل النصية هي أمثلة على المتسلسلات Sequences ، ولكن ما هي المتسلسلة ، وماذا فيها من الخصوصية ؟ اثنان من السمات الرئيسية للمتسلسلة هي عملية الفهرسة التي تتيح لنا جلب بند بعينه في المتسلسلة مباشرة ، وعملية التقطيع الذي يتيح لنا أن نستعيد شريحة من المتسلسلة أي جزءا من المتسلسلة.

استخدام المتسلسلات

مثال 9.5 استخدام المتسلسلات

```
#!/usr/bin/python
# Filename: seq.py

shoplist = ['apple', 'mango', 'carrot', 'banana']

# Indexing or 'Subscription' operation
print 'Item 0 is', shoplist[0]
print 'Item 1 is', shoplist[1]
print 'Item 2 is', shoplist[2]
print 'Item 3 is', shoplist[3]
print 'Item -1 is', shoplist[-1]
print 'Item -2 is', shoplist[-2]

# Slicing on a list
print 'Item 1 to 3 is', shoplist[1:3]
print 'Item 2 to end is', shoplist[2:]
print 'Item 1 to -1 is', shoplist[1:-1]
print 'Item start to end is', shoplist[:]

# Slicing on a string
name = 'swaroop'
```

```
print 'characters 1 to 3 is', name[1:3]
print 'characters 2 to end is', name[2:]
print 'characters 1 to -1 is', name[1:-1]
print 'characters start to end is', name[:]
```

الخرج

```
$ python seq.py
Item 0 is apple
Item 1 is mango
Item 2 is carrot
Item 3 is banana
Item -1 is banana
Item -2 is carrot
Item 1 to 3 is ['mango', 'carrot']
Item 2 to end is ['carrot', 'banana']
Item 1 to -1 is ['mango', 'carrot']
Item start to end is ['apple', 'mango', 'carrot', 'banana']
characters 1 to 3 is wa
characters 2 to end is aroop
characters 1 to -1 is waroo
characters start to end is swaroop
```

كيف يعمل البرنامج

أولا ، نرى كيفية استخدام الفهارس للحصول على عناصر فردية من المتسلسلة. وهذا أيضا يشار اليه على انه عملية الاكتتاب. كلما قمت بتحديد عدد للمتسلسلة بين معقوفتين [ ] كما هو مبين أعلاه ، سوف يجلب لك بيثون البند المقابل لموضعه في المتسلسلة . نتذكر ان بيثون يبدأ عد الأرقام من 0. ومن هنا shoplist [0] يجلب البند الأول و shoplist [3] يجلب البند الرابع في متسلسلة shoplist

يمكن للفهرس أيضا أن يكون عددا سلبيا ، في هذه الحالة، يحسب من نهاية المتسلسلة. لذا فإن shoplist [-1] يشير إلى البند الأخير في المتسلسلة و shoplist [-2] يجلب ثاني آخر بند في المتسلسلة.

عملية التقطيع slicing operation تستخدم عن طريق تحديد اسم المتسلسلة يليها -اختياريا- زوج من الأرقام مفصولة بنقطتين داخل قوسين مربعين [:] . نلاحظ أن هذا الأمر يشبه إلى حد بعيد جدا عملية الفهرسة التي قد قمت باستعمالها. تذكر أن الأرقام اختيارية ولكن النقطتان الرأسيتان ":" ليست كذلك.

الرقم الأول (قبل النقطتين) في عملية التقطيع يشير إلى الموضع الذي تبدأ منه الشريحة ، والعدد الثاني (بعد النقطتين) يشير فيها للموضع الذي تتوقف عنده الشريحة. إذا كان اول عدد غير محدد ، فإن بيثون ستبدأ من بداية المتسلسلة. وإذا كان الرقم الثاني متروكا فإن بيثون ستتوقف في نهاية المتسلسلة. علما أن الشريحة تعاود البدء عند موضع البداية، وستنتهي قبل موضع الانتهاء . مثلا؛ موضع البداية يضاف و أما موضع الانتهاء فهو مستبعد من شريحة المتسلسلة.

وهكذا shoplist [1:3] تعيد قطعة من المتسلسلة بدءا من الموضع 1 بالإضافة إلى موضع 2 ، ولكن يتوقف عند الموضع 3 ، وبالتالي فإن هناك قطعة من هذين البندين يعود. وبالمثل ، shoplist [:] تعيد نسخة من المتسلسلة بأكملها.

يمكنك أيضا التقطيع مع المواضع السالبة. وتستخدم الأرقام السالبة للمواضع من نهاية المتسلسلة. على سبيل المثال ،shoplist [: -1] سيعيد قطعة من المتسلسلة التي تستثني البند الأخير في المتسلسلة ، ولكنه لا يتضمن أي شيء آخر.

جرب توليفات مختلفة من مواصفات هذه الشريحة باستخدام مفسر بيثون التفاعلي .أي المحث الفوري بحيث يمكنك أن ترى النتائج فورا. والشيء العظيم في المتسلسلات هو أنك يمكنك تشغيل الصفوف ، و القوائم و النصوص ، الجميع بنفس الطريقة!

References

عندما تصنع كائنا ويسند إلى أحد المتغيرات ، لا تشير المتغير إلا إلى كائن ولا يمثل كائنا في حد ذاته! وهذا هو المعنى المراد ، أي أن اسم المتغير يشير إلى ذلك الجزء من ذاكرة الكمبيوتر حيث تخزن فيه الكائنات . وهذا ما يسمى ربط binding الاسم بالكائن .

عموما ، لست بحاجة إلى أن تشعر بالقلق إزاء هذا الأمر ، ولكن ثمة تأثير رقيق بسبب references التي تحتاج إلى أن تكون على علم بها . ويتضح ذلك من المثال التالي :

Objects and References

Example 9.6. Objects and References

```
#!/usr/bin/python
# Filename: reference.py

print 'Simple Assignment'
shoplist = ['apple', 'mango', 'carrot', 'banana']
mylist = shoplist # mylist is just another name pointing to the same object!

del shoplist[0] # I purchased the first item, so I remove it from the list

print 'shoplist is', shoplist
print 'mylist is', mylist
# notice that both shoplist and mylist both print the same list without
# the 'apple' confirming that they point to the same object

print 'Copy by making a full slice'
mylist = shoplist[:] # make a copy by doing a full slice
del mylist[0] # remove first item

print 'shoplist is', shoplist
print 'mylist is', mylist
# notice that now the two lists are different
```

الخرج

```
$ python reference.py
Simple Assignment
```

```
shoplist is ['mango', 'carrot', 'banana']
mylist is ['mango', 'carrot', 'banana']
Copy by making a full slice
shoplist is ['mango', 'carrot', 'banana']
mylist is ['carrot', 'banana']
```

كيف يعمل البرنامج

معظم الشرح متاح في التعليقات نفسها. الأمر الذي تحتاج إلى تذكره انك إذا أردت أن تجعل نسخة من القائمة أو من تلك الأنواع من المتسلسلات أو الكائنات المعقدة (ليست كائنات بسيطة مثل الأعداد الصحيحة) ، فإن عليك أن تستخدم عملية التقطيع slicing operation لعمل نسخة. إذا قمت فقط بمجرد إسناد اسم المتغير إلى اسم آخر ، كلاهما يشير إلى الكائن ذاته ، فهذا يمكن أن يؤدي إلى جميع أنواع المتاعب إذا لم تكن حذرا.

ملاحظة لمبرمجي بيرل :

تذكر أن إسناد فئة إلى القوائم لا ينشئ نسخة منها ، عليك أن تقوم بعملية تقطيع slicing operation لعمل نسخة من المتسلسلة .

المزيد عن السلاسل النصية

لقد ناقشنا بالفعل السلاسل النصية بالتفصيل في وقت سابقا . ما المزيد الذي يمكن معرفته عنها؟

حسنا ، هل تعرف ان السلاسل النصية تعتبر هي أيضا objects ولديها الاساليب لفعل كل شيء من أول فحص جزء من النص حتى تعرية المساحات!

السلاسل النصية التي تستخدمها في البرنامج هي جميع الكائن من الفئة (str) بعض من الطرق المفيدة لهذه الفئة تتجلى في المثال التالي. وللحصول على قائمة كاملة من هذه الطرق ، انظر help(str).

مثال 9.7 طرق السلاسل النصية

```
#!/usr/bin/python
# Filename: str_methods.py

name = 'Swaroop' # This is a string object
```

```
if name.startswith('Swa'):
        print 'Yes, the string starts with "Swa"'

if 'a' in name:
        print 'Yes, it contains the string "a"'

if name.find('war') != -1:
        print 'Yes, it contains the string "war"'

delimiter = '_*_'
mylist = ['Brazil', 'Russia', 'India', 'China']
print delimiter.join(mylist)
```

الخرج

```
$ python str_methods.py
Yes, the string starts with "Swa"
Yes, it contains the string "a"
Yes, it contains the string "war"
Brazil_*_Russia_*_India_*_China
```

كيف يعمل البرنامج

هنا ، نرى الكثير من أساليب السلاسل النصية strings داخل العمل طريق. Startswith تستخدم لمعرفة ما اذا كانت السلسلة النصية تبدأ مع الجملة المعطاة. المشغل in يستخدم لفحص ما اذا كان النص المعطى هو جزء من السلسلة النصية..

طريقة find تستخدم لايجاد موضع النص المعطى في string أو إعادة 1- اذا لم يتم النجاح في العثور على النص الثانوي. الفئة str لها أيضا طريقه بارعة في ربط بنود من المتسلسلة مع السلسلة النصية string بصفتها محدد بين كل بند من المتسلسلة sequence وتعيد

أكبر سلسلة نصية متولدة منها .

الخلاصة

لقد قمنا باستكشاف هياكل البيانات المدمجة في بيثون بالتفصيل. هياكل البيانات هذه ستكون أساسية عند كتابة برامج بحجم معقول.

والآن لدينا الكثير من أساسيات بيثون في مكان واحد، و سوف نرى فيما يلي كيفية تصميم وكتابة برنامج في العالم الحقيقي لبايثون

فصل 10.حل مشكلة - كتابه سكربت بيثون

قائمة المحتويات

لقد استكشفنا أجزاء مختلفة من لغة بيثون والآن سوف نلقي نظرة على الطريقة التي تناسب جميع هذه الأجزاء معا ، عن طريق تصميم وكتابة البرنامج الذي يفعل شيئا مفيداً .

المشكلة

المشكلة هي أنني أريد برنامجا يقوم بعمل نسخة احتياطية من جميع ملفاتي ورغم أن هذا بمثابة مشكلة بسيطة ، ليست هناك معلومات كافية بالنسبة لنا لنبدأ عملية الحل. نحتاج مزيدا من التحليل . على سبيل المثال ، كيف يمكننا أن نحدد الملفات التي سيتم نسخها احتياطيا ؟ أين ستوضع النسخة الاحتياطية المخزنة ؟ كيف يتم تخزينها في النسخة الاحتياطية؟
بعد تحليل المشكلة بشكل صحيح ، **نصمم برنامجنا**. نقوم بتجهيز قائمة من الأمور حول كيفية عمل برنامجنا. وفي هذه الحالة ، قمت بإنشاء القائمة التالية بشأن كيفية قيامها بالعمل. إذا قمت بعمل التصميم ، لعلك لا تواجه نفس النوع من المشاكل -- كل شخص له طريقته الخاصة لتسيير الأمور ، وهذا أمر طيب.

1. الملفات والأدلة التي نعمل لها نسخة احتياطية محددة في قائمة.
2. النسخة الاحتياطية يجب أن تكون مخزنة في الدليل الرئيس للنسخ الاحتياطي.
3. ستحزم الملفات في ملف مضغوط.
4. اسم الأرشيف المضغوط يتضمن التاريخ والوقت الحالي.
5. نستخدم الأمر القياسي zip المتاح بشكل افتراضي في أي توزيعة قياسية من لينكس / يونكس. ويمكن لمستخدمي ويندوز استخدام برنامج Info-Zip program – علما بأنه يمكنك استخدام أي أمر لبرنامج أرشفة تريده طالما ان له واجهة سطر الأوامر ، حتى يتسنى

لنا تمرير القيم إليه من السكربت الخاص بنا.

الحـل:

وبما أن تصميم برنامجنا الآن مستقر ، يمكننا أن نكتب الكود الذي يُعتبر أداتنا لتنفيذ الحل .

الإصدار الأول

مثال 10.1 سكربت النسخ الاحتياطي – الإصدار الأول

```
#!/usr/bin/python
# Filename: backup_ver1.py

import os
import time

# 1. The files and directories to be backed up are specified in a list.
source = ['/home/swaroop/byte', '/home/swaroop/bin']
# If you are using Windows, use source = [r'C:\Documents', r'D:\Work'] or
something like that

# 2. The backup must be stored in a main backup directory
target_dir = '/mnt/e/backup/' # Remember to change this to what you will be using

# 3. The files are backed up into a zip file.
# 4. The name of the zip archive is the current date and time
target = target_dir + time.strftime('%Y%m%d%H%M%S') + '.zip'

# 5. We use the zip command (in Unix/Linux) to put the files in a zip archive
zip_command = "zip -qr '%s' %s" % (target, ' '.join(source))

# Run the backup
if os.system(zip_command) == 0:
```

```
        print 'Successful backup to', target
else:
        print 'Backup FAILED'
```

الخرج

```
$ python backup_ver1.py
Successful backup to /mnt/e/backup/20041208073244.zip
```

نحن الآن في مرحلة الاختبار ؛ حيث إننا نختبر كون برنامجنا يعمل بشكل سليم. فإذا لم يتصرف كما هو متوقع ، فسيكون علينا الانتقال إلى مرحلة تصحيح/debug برنامجنا ؛ أي إزالة الأخطاء bugs من البرنامج.

كيف يعمل

ستلاحظ كيف قمنا بتحويل ما لدينا من تصميم إلى شفرة خطوة فخطوة. ونحن نستخدم الوحدة os و time ولذا قمنا باستيرادها. ثم ، نحدد الملفات والأدلة التي سيتم نسخها احتياطيا في قائمة "source" . الدليل الوجهة يعني مكان تخزين جميع الملفات الاحتياطية ، وهذا هو المحدد في المتغير "target_dir" اسم الأرشيف المضغوط الذي سنقوم بإنشائه هو التاريخ الحالي والوقت الذي يجلب لنا باستخدام الدالة time.strftime() . وسوف يكون أيضا. بامتداد .zip وسيخزن في الدليل target_dir

الدالة time.strftime() تأخذ مواصفات مثل التي استخدمناها في البرنامج المذكور أعلاه. الصفة Y% سيحل محلها السنة بدون القرن ، والصفة m% سيحل محلها الشهر بوصفها رقم عشري بين 01 و 12 وهلم جرا. والقائمة الكاملة لهذه المواصفات يمكن العثور عليها في [الدليل المرجعي لبايثون] [Python Reference Manual] الذي يأتي مع بيثون في التوزيعة الخاصة بك. لاحظ أن هذا مشابه (ولكن ليس على النحو نفسه) للمواصفات المستخدمة في إفادة print (باستخدام ٪ متبوعة بصف- tuple)

قمنا بعمل اسم الدليل المضغوط target باستخدام عامل الإضافة الذي يشبك الجمل أي يربط بين اثنين معا ويعيدها إلينا واحدة جديدة. ثم ، ننشئ سلسلة نصية : zip_command ، والتي تتضمن الأمر الذي سنقوم بتنفيذه. يمكنك معرفة ما إذا كان هذا الأمر يعمل عن طريق تشغيله في الصدفة (طرفية لينكس أو محث دوس)

الأمر zip الذي نستخدمه يحتوي بعض الخيارات والمعاملات الممررة – الخيار q يستخدم للإشارة إلى أن الأمر zip ينبغي أن يعمل بهدوء تماما – الخيار r يحدد أن الأمر zip يجب أن يعمل بشكل تتابعيا (recursively) في الأدلة أي ينبغي أن تشمل الأدلة الفرعية والملفات

داخل الأدلة الفرعية كذلك. وقد تم الجمع بين خيارين لا ثالث لهما بشكل مختصروهما qr – هذه الخيارات متبوعة باسم الأرشيف المضغوط المراد إنشاؤه متبوعا بقائمة الملفات والأدلة التي سنقوم بنسخها احتياطيا. نحن نحول قائمة source داخل الجملة باستخدام طريقة join لضم الجمل والتي شاهدنا بالفعل كيفية استخدامها.

وأخيرا نشغل الأمر باستخدام الدالة os.system التي تشغل الأمر كما لو كان يعمل من داخل النظام في الصدفة – وهو يعيد لنا 0 إذا تمت العملية بنجاح ، وإلا فانه يعيد إلينا رقم الخطأ.

واعتمادا على نتيجة الأمر ، نقوم بطباعة رسالة مناسبة بأن النسخة الاحتياطية فشلت أو نجحت ، وهذا هو كل ما في الموضوع ، لقد قمنا بإنشاء سكربت لعمل نسخة احتياطية من الملفات المهمة!

ملاحظة لمستخدمي ويندوز :

يمكنك أن تحدد القائمة source والدليل target لأي اسم ملف أو دليل، ولكن يجب أن تكون حذرا قليلا في ويندوز. والمشكلة هي أن ويندوز يستخدم (\\) الشرطة المائلة الخلفية كفاصل الأدلة، ولكن بيثون تستخدم الشرطة المائلة الخلفية لتمثيل تتابعات الخلوص (escape sequences).

لذلك، عليك أن تمثل الشرطة الخلفية ذاتهاباستخدام تتابع خلوص أو عليك أن تستخدم raw strings. على سبيل المثال، استخدم'C:\\Documents' أو r'C:\Documents' ولكن **لا** تستخدم 'C:\Documents' فأنت هكذا تستخدم تتابع خلوص مجهول \D.

الآن، وحيث أننا قمنا بعمل سكربت للنسخ الاحتياطي، يمكننا استخدامه حينما نريد أن نأخذ نسخة احتياطية من الملفات. وينصح مستخدمو لينكس/يونكس باستخدام طريقة الملف التنفيذي على النحو الذي سبق مناقشته، حتى يتمكنوا من تشغيل برنامج النسخ الاحتياطي في أي وقت وفي أي مكان. وهذا ما يسمى مرحلة **التشغيل** أو مرحلة **نشر** البرمجيات.

يعمل البرنامج أعلاه بشكل صحيح، ولكن (عادة) البرامج الأولى لا تعمل تماما كما كنت تتوقع. على سبيل المثال، قد تكون هناك مشاكل إذا كنت لم تصمم البرنامج بشكل صحيح، أو إذا كنت قد أخطأت في كتابه الكود، إلخ. سيتعين عليك العودة إلى مرحلة التصميم أو ستضطر لتتبع علل برنامجك.

الإصدار الثاني :

الإصدار الأول من برنامجنا يعمل. ولكن، يمكنناإدخال بعض التحسينات عليه حتى يمكنه أن يعمل على نحو أفضل على أساس يومي. وهذا ما يسمى مرحلة **صيانة** البرمجيات.

أحد هذه التحسينات التي شعرت بفائدتها هي ميكانيكية أفضل لتسمية الملفات - باستخدام *time* كاسم للملف بداخل دليل يحمل التاريخ الحالي *date* كدليل ضمن دليل النسخة الاحتياطية الرئيس. أحد الميزات هي أن نسخك الاحتياطية سيتم تخزينها بطريقة هرمية، ولذا سيكون من الأسهل إدارتها. وهناك ميزة أخرى وهي أن طول أسماء الملفات أقصر بكثير بهذه الطريقة. وأيضا هناك ميزة أخرى هي أن الأدلة المنفصلة ستساعدك أن تعرف بسهولة إذا ما كنت قد قمت بعمل نسخة احتياطية عن كل يوم حيث لن ينشأ الدليل إلا إذا كنت قد أخذت نسخة احتياطية لذلك اليوم.

مثال 10.2 سكربت النسخ الاحتياطي- الإصدار الثاني

```
#!/usr/bin/python
# Filename: backup_ver2.py

import os
import time

# 1. The files and directories to be backed up are specified in a list.
source = ['/home/swaroop/byte', '/home/swaroop/bin']
# If you are using Windows, use source = [r'C:\Documents', r'D:\Work'] or
something like that

# 2. The backup must be stored in a main backup directory
target_dir = '/mnt/e/backup/' # Remember to change this to what you will be using

# 3. The files are backed up into a zip file.
# 4. The current day is the name of the subdirectory in the main directory
today = target_dir + time.strftime('%Y%m%d')
# The current time is the name of the zip archive
now = time.strftime('%H%M%S')

# Create the subdirectory if it isn't already there
if not os.path.exists(today):
```

```
        os.mkdir(today) # make directory
        print 'Successfully created directory', today

# The name of the zip file
target = today + os.sep + now + '.zip'

# 5. We use the zip command (in Unix/Linux) to put the files in a zip archive
zip_command = "zip -qr '%s' %s" % (target, ' '.join(source))

# Run the backup
if os.system(zip_command) == 0:
        print 'Successful backup to', target
else:
        print 'Backup FAILED'
```

الخرج

```
$ python backup_ver2.py
Successfully created directory /mnt/e/backup/20041208
Successful backup to /mnt/e/backup/20041208/080020.zip

$ python backup_ver2.py
Successful backup to /mnt/e/backup/20041208/080428.zip
```

كيف يعمل

معظم أجزاء هذا البرنامج لم يعترها التغيير. والتغيرات هي أننا نتحقق إذا كان هناك دليل باسم اليوم الحالي داخل الدليل الرئيس للنسخة الاحتياطية باستخدام الدالة os.exists. فإذا لم يكن موجودا ، فنحن ننشئه مستخدمين الدالة os.mkdir.

لاحظ استخدام المتغير os.sep - فهو يعطيفاصلا للأدلة وفقا لنظام التشغيل الخاص بك أي انه سيكون '/' في لينكس ، ويونيكس ، وسيكون '\ \' في ويندوز و':' في نظام تشغيل ماكنتوش. استخدام Os.sep بدلا من هذه الحروف بشكل مباشر ستجعل برنامجنا محمولا ويعمل عبر هذه النظم.

الإصدار الثالث :

النسخة الثانية تعمل جيدا عندما أقوم بعمل الكثير من النسخ الاحتياطية، ولكن عندما تكون هناك كم كبير من النسخ الاحتياطية، وجدت صعوبة في التفريق بين الغرض من كل نسخة احتياطية، فعلى سبيل المثال، إذا قمت بعض التغييرات الرئيسة على برنامج أو عرض، فأريد ربط هذه التغييرات باسم الملف المضغوط. وهذا يمكن تحقيقه بسهولة عن طريق إرفاق تعليق من المستخدم باسم الأرشيف المضغوط..

مثال 10.3 سكربت النسخ الاحتياطي (لا يعمل)

```
#!/usr/bin/python
# Filename: backup_ver2.py

import os
import time

# 1. The files and directories to be backed up are specified in a list.
source = ['/home/swaroop/byte', '/home/swaroop/bin']
# If you are using Windows, use source = [r'C:\Documents', r'D:\Work'] or
something like that

# 2. The backup must be stored in a main backup directory
target_dir = '/mnt/e/backup/' # Remember to change this to what you will be using

# 3. The files are backed up into a zip file.
# 4. The current day is the name of the subdirectory in the main directory
today = target_dir + time.strftime('%Y%m%d')
# The current time is the name of the zip archive
now = time.strftime('%H%M%S')

# Take a comment from the user to create the name of the zip file
comment = raw_input('Enter a comment --> ')
if len(comment) == 0: # check if a comment was entered
```

```
        target = today + os.sep + now + '.zip'
else:
        target = today + os.sep + now + '_' +
                comment.replace(' ', '_') + '.zip'

# Create the subdirectory if it isn't already there
if not os.path.exists(today):
        os.mkdir(today) # make directory
        print 'Successfully created directory', today

# 5. We use the zip command (in Unix/Linux) to put the files in a zip archive
zip_command = "zip -qr '%s' %s" % (target, ' '.join(source))

# Run the backup
if os.system(zip_command) == 0:
        print 'Successful backup to', target
else:
        print 'Backup FAILED'
```

الخرج

```
$ python backup_ver3.py
File "backup_ver3.py", line 25
target = today + os.sep + now + '_' +
                                    ^
SyntaxError: invalid syntax
```

كيف لم يعمل البرنامج ؟

هذا البرنامج لا يعمل!. بيثون تقول إن ثمة خطأ صياغي مما يعني أن البرنامج لا يوفي التركيب الذي تتوقعه بيثون . عندما نلاحظ الخطأ الذي قدمته بيثون ، كذلك تخبرنا عن المكان الذي اكتشفت الخطأ . لذا ستبدأ *تتبع الخطأ* من هذا السطر.

وبالملاحظة الدقيقة، نرى أن سطرا منطقيا قد انقسم الى سطرين ماديين، ولكننا لم نوضح أن

هذين السطرين الماديين معا. ببساطة، فقد وجدت بيثون أن عامل الإضافة (+) بدون حدود حسابية operand في هذا السطر، وبالتالي لا تعرف كيف تواصل العمل. تذكر أننا يمكن أن نحدد أن السطر المنطقي لا يزال متواصلا في السطر المادي القادم باستخدام الشرطة المائلة الخلفية (backslash) في نهاية السطر المادي. لذا نقوم بهذا التصحيح في برنامجنا. وهذا ما يسمى إصلاح الخطأ (**bug fixing**).

الإصدار الرابع

مثال 10.4 سكربت النسخ الاحتياطي

```
#!/usr/bin/python
# Filename: backup_ver2.py

import os, time

# 1. The files and directories to be backed up are specified in a list.
source = ['/home/swaroop/byte', '/home/swaroop/bin']
# If you are using Windows, use source = [r'C:\Documents', r'D:\Work'] or something like that

# 2. The backup must be stored in a main backup directory
target_dir = '/mnt/e/backup/' # Remember to change this to what you will be using

# 3. The files are backed up into a zip file.
# 4. The current day is the name of the subdirectory in the main directory
today = target_dir + time.strftime('%Y%m%d')
# The current time is the name of the zip archive
now = time.strftime('%H%M%S')
```

```
# Take a comment from the user to create the name of the zip file
comment = raw_input('Enter a comment --> ')
if len(comment) == 0: # check if a comment was entered
    target = today + os.sep + now + '.zip'
else:
    target = today + os.sep + now + '_' + \
        comment.replace(' ', '_') + '.zip'
    # Notice the backslash!

# Create the subdirectory if it isn't already there
if not os.path.exists(today):
    os.mkdir(today) # make directory
    print 'Successfully created directory', today

# 5. We use the zip command (in Unix/Linux) to put the files in a zip archive
zip_command = "zip -qr '%s' %s" % (target, ' '.join(source))

# Run the backup
if os.system(zip_command) == 0:
    print 'Successful backup to', target
else:
    print 'Backup FAILED'
```

الخرج

```
$ python backup_ver4.py
Enter a comment --> added new examples
Successful backup to /mnt/e/backup/20041208/082156_added_new_examples.zip

$ python backup_ver4.py
```

```
Enter a comment -->
Successful backup to /mnt/e/backup/20041208/082316.zip
```

كيف يعمل البرنامج :

هذا البرنامج يعمل الآن! دعونا نستعرض التحسينات الفعلية التي قمنا بعملها في النسخه 3. نأخذ تعليقات المستخدم باستخدام دالة `raw_input` ثم نتحقق ما إذا كان المستخدم قد أدخل به شيئا بالفعل باستوضاح طول المدخلات باستخدام الدالة `len`. إذا قام المستخدم بمجرد بالضغط على مفتاح **enter** لسبب ما (لعلها كانت مجرد نسخة روتينة، أو لم تتم أية تغييرات)، فإننا نواصل كما فعلنا من قبل.

ولكن إذا كان هناك تعليق، فهذا التعليق يلحق باسم الأرشيف المضغوط قبل الامتداد .zip. ولاحظ أننا نستبدل المساحات في التعليق بشرط سفلية (_) وذلك لأن التعامل مع أسماء الملفات هكذا أسهل بكثير.

مزيد من التحسينات

الإصدارة الرابعة هي سكربت يعمل بصورة مُرضية لمعظم المستخدمين، ولكن هناك دائما مجال للتحسين. مثلا، يمكنك إضافة مستوى من *الاستفاضة* للبرنامج، حيث يمكنك تحديد الخيار `-v` لجعل برنامجك يصبح أكثر تكلما.

من التحسينات الممكنة الأخرى أن تسمح بتمرير الملفات والأدلة الاضافية إلى البرنامج في سطر الاوامر. وسنتوصل إلى ذلك من قائمة sys.argv ونستطيع أن نضيفها إلى قائمة source التي لدينا باستخدام طريقة extend التي يوفرها الصنف list.

من التحسينات التي أحبذها استخدام الأمر **tar** بدلا من الأمر **zip**. أحد مزايا ذلك الأمر أنه عند استخدامك **tar** جنبا إلى جنب مع **gzip** يصبح النسخ الاحتياطي أكثر سرعة والنسخة الاحتياطية تكون أقل حجما. وإذا أردت استخدام هذا الأرشيف في ويندوز فإن WinZip يتعامل مع ملفات .tar.gz أيضاً بسهولة. والأمر **tar** متاح بشكل افتراضي في معظم أنظمة لينكس/يونكس. ويمكن لمستخدمي ويندوز أيضا تنزيله من الإنترنت وتثبيته.

الآن سيكون الأمر بالشكل التالي :

```
tar = 'tar -cvzf %s %s -X /home/swaroop/excludes.txt' % (target, '
'.join(srcdir))
```

والخيارات موضحة أدناه.

c- يشير إلى creation إنشاء أرشيف.

v- يشير إلى verbose أي أن الأمر يجب ان تكون اكثر تكلما وثرثرة talkative.

z- يشير إلى مرشح gzip الذي ينبغي استخدامه.

f- يشير إلى force أي الإجبار في إنشاء الأرشيف أي إنه ينبغي أن يحل محل ملف آخر إذا كان يحمل نفس الاسم بالفعل.
x- يشير إلى الملف الذي يتضمن قائمة أسماء الملفات التي يجب استبعادها excluded من النسخة الاحتياطية. على سبيل المثال ، يمكنك تحديد ~* في هذا الملف لعدم إدراج أي أسماء للملفات المنتهية ب ~ في النسخة الاحتياطية

*مهم *

أكثر الطرق المفضلة لإنشاء مثل هذا النوع من الأرشيف سيكون باستخدام الوحدة zipfile أو tarfile على التوالي. إنها تشكل جزءا من مكتبة بيثون القياسية والمتاح لك استخدامها بالفعل. باستخدام هذه المكتبات أيضا تتجنب استخدام os.system والتي لا ينصح باستخدامها على وجه العموم، لانها من السهل أن تكلفك أخطاء باهظة باستخدامها.
ومع ذلك ، فقد كنت أستخدم طريقة os.system لعمل نسخ احتياطية لأغراض تعليمية بحتة ، لذا يعتبر ذلك مثال بسيط بشكل كاف ليكون مفهوما من قبل الجميع ، ولكنها حقيقية أيضا بما يكفي.

عملية تطوير البرمجيات:

الآن وقد قمنا باجتياز المراحل المختلفة في عملية كتابة البرمجيات. فإن هذه المراحل يمكن تلخيصها على النحو التالي :

1. ماذا (التحليل)
2. كيف (التصميم)
3. افعلها (التنفيذ)
4. الاختبار (اختبار وتصحيح الأخطاء)
5. الاستخدام (أو عملية النشر)
6. الصيانة (التحسين)

مهم

الطريقة الموصى بها لكتابة البرامج هي الإجراء الذي اتبعناه في برنامج عمل النسخ الاحتياطية – قم بالتحليل ثم التصميم. ابدأ بتنفيذ صيغة بسيطة للبرنامج . الاختبار والتصحيح . استخدام البرنامج للتأكد من أنه يعمل كما هو متوقع. والآن ، أضف أية ميزات تريدها واستمر في تكرار هذه الدورة : "افعل-جرب-استخدم" "Do It-Test-Use" لأي عدد من المرات على النحو المطلوب. وتذكر ؛ البرمجيات تنمو كالزرع ، ولا تبنى ! " Software is grown, not built "

الخلاصة

ولقد رأينا كيفية عمل البرامج الخاصة في بيثون والمراحل المختلفة التي تشارك في كتابة مثل هذه البرامج. وربما تجد أنه من المفيد إنشاء برامجك بنفسك مثلما فعلنا في هذا الفصل حتى تصبح مرتاحا مع بيثون فضلا عن القدرة على حل المشاكل. وفيما يلي؛سوف نناقش البرمجة الكائنية "object-oriented"

فصل.11. البرمجة الكائنية الموجهة

جدول المحتويات

مقدمــة:

في جميع برامجنا وحتى الآن، لقد قمنا بتصميم برنامجنا حول دوال أو لبنات من الإفادات التي تتلاعب بالبيانات. ويسمى هذا طريقة البرمجة *الإجرائية الموجهة*. وهناك طريقة أخرى لتنظيم برنامجك وهو الجمع بين الوظيفة والبيانات وتغليفها معا فيما يسمى بالكائن object. وهذا ما يسمى نموذج البرمجة *الكائنية الموجهة*. في معظم الوقت يمكنك استخدام البرمجه الاجراءيه ولكن في بعض الأحيان عندما تريد كتابة برامج كبيرة، أو ان يكون ذلك هو الحل الأنسب لها، يمكنك استخدام تقنيات البرمجة كائنية التوجه.

الفئات والكائنات يعتبران هما الشكلان الرئيسيان للبرمجه الكائنية الموجهة.حيث إن الفئة class تنشئ نوعاً جديدا حيث تعتبر **الكائنات** *أمثلة* من الفئة. أحد الأقيسة على ذلك هو أنه يمكن أن يكون لديك متغيرات من نوعية العدد الصحيح int والتي تترجم إلى قولنا إن المتغيرات التي تخزن الأعداد الصحيحه هي المتغيرات التي تعتبر حالات (أوكائنات objects) من الفئة int.

ملاحظة لمبرمجي C/C++/Java/C#*:

نلاحظ أنه حتى الأعداد الصحيحة تعامل على أنها كائنات (من الفئة int). وهذا بخلاف سي+ + وجافا (قبل الإصدار 1.5) حيث الأعداد الصحيحة هي أنواع بدائية الأصل. انظر help(int) لمزيد من التفاصيل حول الفئة- class .

مبرمجو سي# و جافا 1.5 سيجدون ذلك الأمر مألوفا إليهم حيث إنه يشبه مفهوم boxing و unboxing

يمكن للكائنات تخزين البيانات باستخدام المتغيرات العاديه التي تنتمي إلى هذه الكائن. والمتغيرات التي تنتمي إلى الفئة أو الكائن تسمى حقول **fields** لهذه الفئة. يمكن للكائنات أن يكون لها ايضا وظائف باستخدام الدوال التي تنتمي إلى الفئة.هذه الدوال تسمى طرقاً **methods** لهذ الفئة. هذه المصطلحات مهمة لانها تساعدنا في التفريق بين الدوال والمتغيرات التي هي مستقلة في حد ذاتها، وتلك التي تنتمي إلى فئة معينة أو كائن ما. وكلها جميعا الحقول والطرق يمكن ان يشار اليها على أنها **مواصفات** لتلك الفئة.

الحقول تتكون من نوعين – يمكن لكل منهما ان تنتمي إلى حالة/كائن من الفئة، أو يمكنها ان تنتمي إلى الفئة نفسها.فهي تسمي المتغيرات الآنية **instance variables**ومتغيرات الفئة **class variables**على التوالي.

الفئة يتم إنشاؤها باستخدام الكلمة المفتاحية class.الحقول وطرق الفئة مدرجة في منظومة اللبنة.هناك فارق واحد محدد لطرق الفئة يخالف الدوال العادية، وذلك انها يجب أن تكون لها اسم أول إضافي يضاف إلى بداية باراميتر القائمة، ولكنك **لاتعطي** قيمة لهذا الباراميتر عندما تستدعي ال method، وبيثون سوف تزودنا به. هذا المتغير المحدد يشير إلى الكائن ذاته، وحسب الاتفاق، فإنها تحظى باسم الذات self.

The self

هناك فارق واحد محدد لطرق الفئة يخالف الدوال العادية،وذلك انها يجب أن تكون لها اسم أول إضافي يضاف إلى بداية باراميتر القائمة، ولكنك **لاتعطي** قيمة لهذا الباراميتر عندما تستدعي ال method، وبيثون سوف تزودنا به. هذا المتغير المحدد يشير إلى الكائن ذاته، وحسب الاتفاق ، فإنها تحظى باسم الذات self.

ورغم انه يمكنك إعطاء أي اسم لهذه الباراميتر، *يوصى بشدة* ان تستخدم اسم self- اي اسم آخر هو بالتأكيد يؤدي إلى العبوس. وهناك العديد من المزايا لاستخدام اسم معياري- وأي قارئ لبرنامجك سوف يعترف به فورا وحتى برامج المخصصة ل (بيئة التطوير المتكامله)يمكن ان تساعدك اذا كنت تستخدم self.

ملاحظة لمبرمجي #C++/Java/C

الإفادة self في بيثون تعادل المؤشر self في لغة C++ و الإشارة this في جافا و سي#

عليك أن تكون مندهشا؛ كيف أن بيثون تعطي قيمة ل self، ولماذا لست بحاجة إلى إعطاء قيمة لها ؟.أحد الأمثلة سيجعل هذا الأمر واضحا. لنقل مثلا أن لديك فئة تدعى MyClassومثال هذه الفئة يسمى MyObject. فعندما تستدعي ال method لهذا الكائن كـما يلي : MyObject.method(arg1, arg2)، فإنه يتم تحويلها تلقائيا عن طريق بيثون إلى MyClass.method(MyObject, arg1, arg2)،وهذا كل ما يخص self.

وهذا يعني أيضا انه إذا كان لديك method لا تأخذ أي argument، فإنك لا تزال بحاجة إلى تحديد طريقة للحصول على قيمة self.

الفئات

أبسط صورة للفئة يمكن رؤيتها منن خلال المثال التالي:

إنشاء الفئة

مثال 11.1 إنشاء الفئة

```
#!/usr/bin/python

# Filename: simplestclass.py

class Person:

        pass # An empty block

p = Person()

print p
```

الخرج

```
$ python simplestclass.py

<__main__.Person instance at 0xf6fcb18c>
```

كيف يعمل

قمنا بإنشاء فئة جديدة باستخدام الإفادة class متبوعة باسم الفئة. وتلك تتبع لبنة مزاحة من الإفادات التي تصوغ هيكل الفئة . وفي هذه الحالة لدينا لبنة فارغة من الإفادات يشار إليها باستخدام الإفادة pass.

ثم قمنا بإنشاء كائن/مثال لهذه الفئة باستخدام اسم الفئة متبوعا بزوج من الأقواس الهلالية. وسوف نتعلم المزيد عن instantiation في القسم التالي. ومن أجل التحقق ، نقوم بفحص نوع المتغير ببساطة بمجرد طباعته. وهو يخبرنا أن لدينا مثالا من الفئة person في الوحدة __main__ .

لاحظ أن عنوان ذاكرة الحاسب عند تخزينك للكائن يتم كتابته أيضاً. وقيمة هذا العنوان سيكون لها قيمة مختلفة على حاسبك حيث إن بيثون تستطيع تخزين الكائن حالما تجد مساحة فارغة.

طرق الكائن

لقد ناقشنا بالفعل أن الفئات/الكائنات يمكنها أن تحتوي طرقا تشبه الدوال إلا إذا كان لدينا متغير self إضافي . والآن سوف نرى مثالا على ذلك .

استخدام طرق الكائن

مثال 11.2 طرق الكائن

```
#!/usr/bin/python
# Filename: method.py

class Person:
    def sayHi(self):
        print 'Hello, how are you?'
p = Person()
p.sayHi()

# This short example can also be written as Person().sayHi()
```

الخرج

```
$ python method.py
Hello, how are you?
```

كيف يعمل :

لقد رأينا طريقة عمل self. لاحظ أن طريقة sayHi لا تأخذ أي بارميتر ولكنها تظل محتفظة ب selfفي تعريف الدالة.

طريقة __init__ :

هناك العديد من أسماء الطرق لها أهمية خاصة في فئات لغة بيثون. وسوف نرى المغزى من طريقة___init الآن.

تعمل طريقة__init__بمجرد عمل الكائن المنتمي للفئة. هذه الطريقة مفيدة لعمل أي تهيئة *initialization*تريد عملها مع الكائن. ولاحظ الشرطتين السفليتين في بداية الاسم ونهايته.

استخدام طريقة ___init :

مثال 11.3 استخدام __method__ init :

```
#!/usr/bin/python
# Filename: class_init.py

class Person:
    def __init__(self, name):
        self.name = name
    def sayHi(self):
        print 'Hello, my name is', self.name

p = Person('Swaroop')
p.sayHi()

# This short example can also be written as Person('Swaroop').sayHi(
```

الخرج

```
$ python class_init.py
Hello, my name is Swaroop
```

كيف يعمل البرنامج :

هنا قمنا بتعريف الطريقة__init__ بإعطائها المعامل__init__(مع selfالمعتاد).وهنا؛ قمنا فحسب بإنشاء حقل جديد كذلك يسمى name.ولاحظ أنهما متغيران مختلفان رغم أنهما يحملان نفس الاسم. الاسم المنقوط يتيح لنا التفرقة بينهما.

والأكثر أهمية، أن تلاحظ أننا لا نستدعي صراحة طريقة__init__ولكن نقوم بتمرير قيم بداخل القوسين بعد اسم الفئة عندما ننشئ عنصر-instance جديد من هذه الفئة. وهذا هو المغزى الخاص من هذه الطريقة.

الآن، نحن قادرون على استخدام حقل self.name في طرقنا التي تتجلى في طريقة sayHi.

ملاحظة لمبرمجي سي++/جافا/سي#

طريقة __init__ مماثلة لـ constructor في سي++/جافا/سي#

متغيرات الكائن والفئة

لقد سبق أن ناقشنا بالفعل الجزء المتعلق بوظيفة الفئات والكائنات ، والآن سنرى جزء البيانات الخاص بها. في الواقع ، إنها ليست سوى متغيرات عادية مرتبطة **بحيزات أسماء** الفئات والكائنات . هذه الأسماء صالحة ضمن سياق هذه الفئات والكائنات فقط.

وهناك نوعان من الحقول -- متغيرات الفئة class variables و متغيرات الكائن object variables والتي تصنف تبعا لما إذا كانت الفئة أو الكائن - على التوالي - تمتلك للمتغيرات .

متغيرات الفئة تشترك في معنى أنها تعمل من خلال جميع الكائنات (العناصر) لهذه الفئة. لا يوجد سوى نسخة من متغير الفئة وعندما يقوم الكائن بعمل ما على متغير الفئة ، ينعكس هذا التغيير في جميع الحالات (الكائنات) الأخرى أيضا.
تختص *متغيرات الكائن* بكل كائن على حدة في العنصر الخاص بالفئة . فمثلا هم لا يشتركون ولا يرتبطون بأي طريقة بنفس الاسم في عنصر مختلف في نفس الفئة. ورؤية مثال واحد سيجعل الأمر سهل الفهماستخدام متغيرات الفئة والكائن

استخدام متغيرات الكائن والفئة

مثال 11.4. استخدام متغيرات الكائن والطبقة

```
#!/usr/bin/python
# Filename: objvar.py
class Person:
    '''Represents a person.'''
    population = 0

    def __init__(self, name):
        '''Initializes the person's data.'''
        self.name = name
        print '(Initializing %s)' % self.name

        # When this person is created, he/she
        # adds to the population
        Person.population += 1

    def __del__(self):
        '''I am dying.'''
        print '%s says bye.' % self.name

        Person.population -= 1

        if Person.population == 0:
            print 'I am the last one.'
        else:
            print 'There are still %d people left.' % Person.population
```

```
        def sayHi(self):
                '''Greeting by the person.

                Really, that's all it does.'''
                print 'Hi, my name is %s.' % self.name

        def howMany(self):
                '''Prints the current population.'''
                if Person.population == 1:
                        print 'I am the only person here.'
                else:
                        print 'We have %d persons here.' % Person.population

swaroop = Person('Swaroop')
swaroop.sayHi()
swaroop.howMany()

kalam = Person('Abdul Kalam')
kalam.sayHi()
kalam.howMany()

swaroop.sayHi()
swaroop.howMany()
```

الخرج

```
$ python objvar.py
(Initializing Swaroop)
Hi, my name is Swaroop.
I am the only person here.
```

```
(Initializing Abdul Kalam)
Hi, my name is Abdul Kalam.
We have 2 persons here.
Hi, my name is Swaroop.
We have 2 persons here.
Abdul Kalam says bye.
There are still 1 people left.
Swaroop says bye.
I am the last one.
```

كيف يعمل :

هذا مثال طويل ولكنه يساعد في تبيين طبيعة متغيرات الفئات والكائنات ؛ وهنا population تنتمي إلى الفئة Person ، ولذا تعتبر متغيرا للفئة . والمتغير name ينتمي إلى الكائن (وهو مسند باستخدام self) وبالتالي هو متغير للكائن .

وهكذا نشير إلى متغير الفئة " population" كـ Person.population وليس كـ self.population . لاحظ أن متغير كائن يحمل نفس اسم متغير فئة سوف يخفي متغير الفئة ! ونحن نشير إلى اسم متغير الكائن name باستخدام self.name في الطرق الخاصة بالكائن .تذكر أن هناك اختلاف بسيط بين متغيرات الفئة ومتغيرات الكائن .

لاحظْ بأنّ طريقة __init__ تُستَعملُ لعمل initialize للحالة Person مَع name . وفي هذه الطريقة، نزيد عدد population بمقدار 1 حيث نحصل Person واحد مضافا . كذلك نلاحظ أن قيم self.name تحدد لكل كائن يشير إلى طبيعة متغيرات الكائن .

تذكر أنه يجب أن تشير إلى المتغيرات والطرق الخاصة بنفس الكائن باستخدام المتغير self **وفقط**. وذلك يدعى *attribute reference*.

في هذا البرنامج نرى أيضا استخدام **docstrings** للفئات وكذلك الطرق. يمكننا الوصول إلى class docstring في وقت التشغيل باستخدام Person.__doc__ والطريقة docstring كـ Person.sayHi.__doc__

وهناك طريقة أخرى تشبه __init__ مثل __del__،التي تستدعى عندما يوشك كائن ما على الموت .ولا يمكن استخدامه بعد ذلك، وستتم إعادته إلى النظام لإعادة استعمال هذا الجزء من الذاكرة. وفي هذه الطريقة نقوم ببساطة بإنقاص حساب ال Person.population بمقدار 1.

طريقة __del__ تعمل حين لا يكون الكائن مستخدما، وليس هناك ضمان *متى* ستعمل هذه الطريقة. وإذا أردت عمل ذلك بوضوح عليك فقط أن تستخدم إفادة del التي استعملناها في الأمثلة السابقة.

ملاحظة لمبرمجي C++/Java/C :

جميع عناصر الفئة(إضافة إلى عناصر البيانات) تعتبر *عمومية* وجميع الطرق تعتبر *تخيلية* في بيثون.

وهناك استثناء واحد؛ إذا كنت تستخدم عناصر البيانات مع الأسماء باستخدام *بادئة الشرطتين السفليتين* مثل __privatevar؛ تستخدم بيثون صقل الاسم بفاعلية ليجعل لها قيمة منفردة.

هكذا فإن الاتفاقية المتبعة هي أن كل متغير يستعمل فقط داخل الفئة أو الكائن يجب أن يصبح بشرطة سفلية، وجميع الأسماء الأخرى عامة-public ويمكن أن تستخدم من قبل أي فئات/كائنات .تذكر أن هذه اتفاقية فحسب وليست إجبارية من بيثون (ما عدا بادئة الشرطة السفلية المزدوجة).

لاحظ أيضاً أن الطريقة __del__ تماثل مفهوم *destructor*.

التوارث

أحد المنافع الرئيسة للبرمجة الكائنية الموجهة هو **إعادة استعمال**الشفرة، وأحد وسائل ذلك يتم عمله من خلال آلية *التوارث* .التوارث يمكن تخيله بشكل افضل على أنه تطبيق علاقة بين *نوع رئيس ونوع فرعي* *subtype*بين الفئات .

لنفترض انك تريد كتابة برنامج يقوم بمتابعة المعلمين والطلاب في كلية. ولديهم بعض الخصائص المشتركة مثل الاسم والسن والعنوان. ولديهم كذلك خصائص معينة مثل الراتب والدورات العلمية، وإجازات للمعلمين، ودرجات ومصاريف للطلبة.

يمكنك أن تنشئ نوعين مستقلين من الفئات لكل نوع وتعالجهما، ولكن بإضافة خاصية مشتركة جديدة، معناها إضافتها إلى كل فئة على حدة. وسريعا يصبح هذا الأمر ثقيلا جدا .

والطريقة الأفضل يمكن ان تكون بإنشاء فئة مشتركة تسمى SchoolMember، وبعدها تجعل فئة teacher وفئة student ترث من هذه الفئة الأولى. وبمعنى آخر سيصبحان أنواع فرعية لهذا النوع (الفئة). وبعد ذلك يمكننا أن نحدد خصائص هذه الأنواع الفرعية.

هناك عدة مميزات في هذه الطريقة.إذا أضفت/غيرت اي وظيفة في SchoolMember، سوف ينعكس هذا تلقائياً على الأنواع الفرعية كذلك. على سبيل المثال؛ يمكنك ان تضيف حقل بطاقة هوية جديدا field card ID لكل من المعلمين والطلاب ببساطة عن طريق إضافة الفئة، classname>SchoolMember. وعلى أية حال التغيرات الحادثة في الأنواع

الفرعية لا تؤثر في الأنواع الفرعية الأخرى .والميزة الأخرى انه يمكنك أن تشير إلى كائنات المعلمين أو الطلبة باعتبارها كائن SchoolMember والذي يمكن أن يكون مفيدا في بعض الحالات، مثل حساب عدد أعضاء المدرسة. وذلك يسمى تعدد الأوجه**تعدد الأوجه**؛ حيث ان النوع الفرعي يمكن أن يستبدل في أي حالة عندما يكون النوع الأصل متوقعا. فمثلا الكائن يمكن تكراره بصفته حالة من الفئة الأصلية.

ونلاحظ كذلك أننا نعيد *استخدام* شفرة الفئة الأصل، ولسنا بحاجة إلى تكرارها في فئات مختلفة، كما كان واجبا في حالة ما استخدمنا فئات مستقلة .

الفئة SchoolMember في هذه الحالة تعرف بأنها *الفئة الأساسية* أو *superclass*، وفئة Teacher وفئة Student تسمى *derived classes* أو *subclasses* .

وسنرى الآن هذا المثال التالي في صورة برنامج.

استخدام التوارث

مثال 11.5 استخدام التوارث

```
#!/usr/bin/python
# Filename: inherit.py

class SchoolMember:
    '''Represents any school member.'''
    def __init__(self, name, age):
        self.name = name
        self.age = age
        print '(Initialized SchoolMember: %s)' % self.name

    def tell(self):
        '''Tell my details.'''
        print 'Name:"%s" Age:"%s"' % (self.name, self.age),

class Teacher(SchoolMember):
    '''Represents a teacher.'''
```

```
        def __init__(self, name, age, salary):
                SchoolMember.__init__(self, name, age)
                self.salary = salary
                print '(Initialized Teacher: %s)' % self.name

        def tell(self):
                SchoolMember.tell(self)
                print 'Salary: "%d"' % self.salary

class Student(SchoolMember):
        '''Represents a student.'''
                SchoolMember.__init__(self, name, age)
                self.marks = marks
                print '(Initialized Student: %s)' % self.name

        def tell(self):
                SchoolMember.tell(self)
                print 'Marks: "%d"' % self.marks

t = Teacher('Mrs. Shrividya', 40, 30000)
s = Student('Swaroop', 22, 75)

print # prints a blank line
members = [t, s]
for member in members:
member.tell() # works for both Teachers and Student
```

الخرج

```
$ python inherit.py
(Initialized SchoolMember: Mrs. Shrividya)
```

(Initialized Teacher: Mrs. Shrividya)

(Initialized SchoolMember: Swaroop)

(Initialized Student: Swaroop)

Name:"Mrs. Shrividya" Age:"40" Salary: "30000"

Name:"Swaroop" Age:"22" Marks: "75"

كيف يعمل :

لاستعمال التوارث لاستخدام التوارث، نقوم بتحديد اسم الفئة الأساسية في tuple متبوعا باسم الفئة في تعريف الفئة. بعد ذلك نلاحظ أن الطريقة __init__الخاصة بالفئة الأساسية تستدعى بشكل واضح باستخدام المتغير self،من أجل ذلك يمكننا إعداد جزء الفئة الأساسية في الكائن. من الأمور المهمة التي عليك أن تتذكرها، إن بيثون لا تقوم تلقائي باستدعاء الدالة المشيّدة للفئة الأساسية، وعليك أن تقوم باستدعائها بشكل واضح بنفسك.

كذلك نلاحظ أنه يمكننا استدعاء طرق الفئة الأساسية عن طريق تقديم اسم الفئة لاستدعاء ال method، وبعد ذلك نمرر إلى المتغير self مع أي قيمة.

نلاحظ أنه يمكننا معالجة الفئات Teacher أو Student كمجرد عناصر للفئة SchoolMember. عندما نستخدم الطريقة tell للفئة SchoolMember.

ونلاحظ كذلك أن الطريقة tell الخاصة بالنوع الفرعي يتم استدعاؤها وليست الطريقة tellالخاصة بالفئة SchoolMember. أحد الطرق لفهم ذلك هو أن بيثون *دائماً* تبدأ في البحث عن الطرق methods في النوع، وهي في هذه الحالة تفعل ذلك. وإذا لم تستطع إيجاد ال method فإنها تبدأ في البحث عن ال methods المنتمية إلى الفئات الأساسية واحدة تلو الأخرى من أجل تحديدها في صف tuple في تعريف الطبقة .

ملاحظة خاصة بالمصطلح –إذا كان هناك أكثر من فئة مندرجة في صف-tuple التوريث، عندئذ تسمى *بالتوريث المتعدد* .

الخلاصـة

لقد قمنا الآن باستكشاف الجوانب المختلفة للفئات والكائنات فضلا عن مختلف المصطلحات المرتبطة بها. وقد شاهدنا أيضا فوائد ومتطلبات البرمجة الكائنية الموجهة. بيثون تعتبر لغة برمجة عالية المستوى في مجال البرمجة الكائنية الموجهة وفهم هذه المفاهيم بعناية سيساعدك كثيرا في المدى البعيد.

وفيما يلي ، سنتعلم كيفية التعامل مع المدخلات والمخرجات وكيفية الوصول إلى الملفات في بيثون.

الفصل 12. الدخل/الخـــرج

قائمة المحتويات

سيكون لديك الكثير من الوقت عندما ترغب في إعطاء برنامجك قدرة على التفاعل مع المستخدم (ولعله يكون أنت نفسك). فأنت تريد إن تأخذ مدخلاً من المستخدم ، ثم تطبع بعض النتائج السابقة. ولا يمكننا أن نحقق ذلك باستخدام الإفادات raw_input و print على التوالي. بالنسبة للخـــرج ، يمكننا أيضا استخدام مختلف طرق الفئة str (string) على سبيل المثال ، يمكنك استخدام طريقة rjust لتحصل على سلسلة نصية ذات محاذاة إلى اليميـــن لنطـــاق محـــدد . انظـــر help(str) لمزيـــد مـــن التفـــا صـــيل .

يوجد نوع شائع آخر من الدخل/الخـــرج تتعامل مع الملفات. إن القدرة على إنشاء، وقراءة وكتابة الملفات أمر أساسي للعديد من البرامج ، و سنبحث في هذا الجانب في هذا الفصل .

الملفات

يمكنك فتح واستخدام الملفات بغرض القراءة أو الكتابة عن طريق إنشاء كائن للفئة file واستخدام طرق read, readline أو write بشكل مناسب للقراءة من الملف أو الكتابة إلى الملف. إن قابلية القراءة أو الكتابة إلى ملف يتوقف على النمط الذي قمت بتحديده لفتح الملف. ثم أخيرا ، وعندما تنتهي من الملف ، يمكنك استدعاء طريقة close لتبلغ بيثون بأننا انتهينا من استخدام الملف .

مثال 12.1 استخدام الملف

```
#!/usr/bin/python
# Filename: using_file.py

poem = '''\
Programming is fun
When the work is done
if you wanna make your work also fun:
        use Python!
'''
```

```
f = file('poem.txt', 'w') # open for 'w'riting
f.write(poem) # write text to file
f.close() # close the file

f = file('poem.txt') # if no mode is specified, 'r'ead mode is assumed by default
while True:
        line = f.readline()
        if len(line) == 0: # Zero length indicates EOF
                break
        print line, # Notice comma to avoid automatic newline added by Python
f.close() # close the file
```

الخـــرج

```
$ python using_file.py
Programming is fun
When the work is done
if you wanna make your work also fun:
        use Python!
```

كيف يعمل البرنامج

أولا، قمنا بإنشاء مثال للفئة `file` عن طريق تحديد اسم الملف والنمط الذي نريد فتح الملف به. النمط يمكن أن يكون للقراءة بواسطة (`'r'`) و أو نمط للكتابة (`'w'`)أو نمط مشترك (`'a'`)، وهناك في الواقع العديد من الأنماط المتاحة، وسوف تعطيك `help(file)` المزيد من التفاصيل عنها.

أولا نفتح ملف في نمط الكتابة واستخدام طريقة write للفئة file للكتابة إلى الملف،ثم أخيرا غلق close هذا الملف.

بعد ذلك، نفتح نفس الملف مرة أخرى للقراءة. وإذا لم نحدد نمطا، يكون نمط القراءة هو الافتراضي. نقرأ في كل سطر من الملف باستخدام الطريقة readline في حلقة. هذه الطريقة ترجع إلينا سطرا كاملا بالإضافة إلى حرف لسطر جديد في نهاية السطر.ولذا عندما يرجع إلينا سلسلة نصية *فارغة*، فهو يشير إلى أن نهاية الملف قد تم الوصول إليها وتتوقف الحلقة.

ولاحظ أننا نستخدم فاصلة مع الإفادة print لمنع حدوث سطر جديد تلقائيا، والذي تضيفه إفادة print لأن السطر الذي يقرأ من الملف بالفعل ينتهي مع إشارة سطر جديد. ثم، أخيرا غلق close هذا الملف.

الآن، اطلع على محتويات الملف poem.txt للتأكد من أن البرنامج يعمل بشكل صحيح.

Pickle

توفر لنا بيثون وحدة معيارية تدعى pickle،والتي يمكن استخدامهافي تخزين **أي** كائن في بيثون في ملف، ثم تحصل عليها لاحقا دون مساس. وهذا ما يسمى تخزين الكائن *الدائم*.

وهناك وحدة أخرى تسمى cPickle والتي وظيفتها بالضبط نفس ما تقوم به الوحدة pickle؛ ما عدا انهامكتوبة بلغة C وهي أسرع بمقدار(1000 مرة أو أكثر). يمكنك استخدام أي من هذه الوحدات cPickle، على الرغم من أننا سوف تستخدم الوحدة cPickle هنا. تذكر، نحن نشير إلى أن كل من هذه الوحدات ببساطة مجرد وحدة pickle

Pickling and Unpickling

مثال 12.2 Pickling and Unpickling

```
#!/usr/bin/python
# Filename: pickling.py

import cPickle as p
#import pickle as p

shoplistfile = 'shoplist.data' # the name of the file where we will store the object

shoplist = ['apple', 'mango', 'carrot']

# Write to the file
f = file(shoplistfile, 'w')
p.dump(shoplist, f) # dump the object to a file
f.close()

del shoplist # remove the shoplist

# Read back from the storage
f = file(shoplistfile)
storedlist = p.load(f)
print storedlist
```

الخـــرج

```
$ python pickling.py
['apple', 'mango', 'carrot']
```

كيف يعمل البرنامج

أولا، نلاحظ أن نستخدم الصيغة import..as. و هذا أمر يسير حيث يمكننا استخدام اسم اقصر للوحدة. وحتى في هذه الحالة، فإنه يتيح لنا الانتقال إلى وحدة مختلفة (cPickle أو pickle)من خلال تغيير بسيط لسطر واحد! في بقية البرنامج، ونحن ببساطة نشير إلى هذه الوحدة، كـ p.

لتخزين كائن في ملف، أولا نقوم بفتح الكائن file في نمط الكتابة وتخزين الكائن في الملف المفتوح عن طريق استدعاء الدالة dump من الوحدة pickle. هذه العملية تسمى *pickling*.

بعد ذلك؛ نسترجع الكائن باستخدام الدالة loadللوحدة pickleالتي ترجع الكائن. هذه العملية تسمى *unpickling*

الخلاصة

لقد ناقشنا مختلف أنواع المدخلات والمخرجات وأيضا معالجة الملفات واستخدام الوحدة pickle.

وفيما يلي سنبحث في مفهوم الاستثناءات exceptions .

فصل 13. الاستثناءات

قائمة المحتويات

تقع الاستثناءات عندما تحدث حالات استثنائية معينة في برنامجك. على سبيل المثال ،ماذا يحدث لو كنت ذاهبا لقراءة ملف ما والملف غير موجود؟ أو ما إذا كنت حذفت بالمصادفة برنامجا كان يعمل؟ مثل هذه الحالات تعالج باستخدام الاستثناءات.

ماذا لو كان لبرنامجك بعض التصريحات غير الصالحة ؟ هذه الأمور تتولاها بيثون والذي ترفع يديها {معلنة لك} وتخبرك أن هناك خطأ ما .

الأخطـــاء

نظرة بسيطة إلى إفادة print. ماذا لو أخطأنا إملائيا في كتابة print وكتبناها كـ Print ؟ لاحظ الحرف الكبير والحرف الصغير. وفي هذه الحالة ، ترفع لنا بيثون خطأ صياغة syntax error.

```
>>> Print 'Hello World'
  File "<stdin>", line 1
    Print 'Hello World'
                ^
SyntaxError: invalid syntax

>>> print 'Hello World'
Hello World
```

نلاحظ أن هذا syntaxerror يرفع ، وأيضا المكان الذي تم اكتشاف خطأ الكتابة عنده. وهذا هو ما يفعله معالج الأخطاء error handler لهذا الخطأ.

Try..Except

سنحاول قراءة مدخلات من المستخدمين. اضغط Ctrl-d وانظر ماذا يحدث.

```
>>> s = raw_input('Enter something --> ')
Enter something --> Traceback (most recent call last):
  File "<stdin>", line 1, in ?
```

EOFError

بيثون ترفع إلينا خطأ يدعى EOFError والذي يعني أساسا أنه تم العثور على نهاية الملف حيث لم يكن ذلك متوقعا (الذي يتمثل من خلال الضغط على مفتاحي Ctrl-d)

وفيما يلي ، سنرى كيفية التعامل مع مثل هذه الأخطاء.

معالجة الاستثناءات

يمكننا معالجة الاستثناءات باستخدام إفادة try..except. وقد وضعنا بالأساس البيانات المعتادة ضمن لبنة try ،وكذلك وضعنا كل معالجات الأخطاء التي لدينا في لبنة except .

مثال 13.1 معالجة الاستثناءات

```
#!/usr/bin/python
# Filename: try_except.py

import sys

try:
        s = raw_input('Enter something --> ')
except EOFError:
        print '\nWhy did you do an EOF on me?'
        sys.exit() # exit the program
except:
        print '\nSome error/exception occurred.'
        # here, we are not exiting the program
print 'Done'
```

الخرج

```
$ python try_except.py
Enter something -->
Why did you do an EOF on me?

$ python try_except.py
Enter something --> Python is exceptional!
```

```
Done
```

كيف يعمل البرنامج

نضع كل الإفادات التي قد تثير خطأ في اللبنة try ومن ثم محاولة معالجة جميع الأخطاء والاستثناءات في البند / اللبنة except . البند except يمكنه معالجة خطأ أو استثناء واحد محدد ، أو قائمة من الجمل الاعتراضية للأخطاء/الاستثناءات. إذا لم يكن هناك أسماء من الأخطاء أو الاستثناءات معطاة ، ستعالج جميع الأخطاء والاستثناءات. ويجب أن يكون هناك بند except واحد على الأقل مرتبط مع كل بند من try .

إذا لم يعالج أي خطأ أو استثناء فإن المعالج الافتراضي لبيثون يستدعي و يوقف تنفيذ البرنامج ويطبع رسالة. وقد رأينا بالفعل في هذا العمل.

يمكنك أيضا الحصول على بند else مرتبطا مع اللبنة try..catch. ويتم تنفيذ البند else عند عدم وجود أي استثناء

يمكن نا كذلك الحصول على الاستثناء لذا يمكننا الحصول على معلومات إضافية عن الاستثناء الذي حدث. ويتجلى هذا في المثال التالي.

رفع الاستثناءات

يمكنك رفع الاستثناءات باستخدام الإفادة raise. يجب عليك أيضا أن تحدد اسم الخطأ /الاستثناء ، والكائن الاستثنائي يكون موضوعا جنبا إلى جنب مع الاستثناء. الخطأ أو الاستثناء الذي يمكنك رفعه ينبغي أن يكون فئة والتي تعتبر بشكل مباشر أو غير مباشر فئة مشتقة عن الفئة Error أو الفئة Exception على التوالي.

كيفية رفع الاستثناءات

مثال 13.2 كيفية رفع الاستثناءات

```
#!/usr/bin/python
# Filename: raising.py

class ShortInputException(Exception):
        '''A user-defined exception class.'''
        def __init__(self, length, atleast):
                Exception.__init__(self)
                self.length = length
                self.atleast = atleast

try:
```

```
        s = raw_input('Enter something --> ')
        if len(s) < 3:
                raise ShortInputException(len(s), 3)
        # Other work can continue as usual here
except EOFError:
        print '\nWhy did you do an EOF on me?'
except ShortInputException, x:
        print 'ShortInputException: The input was of length %d, \
                was expecting at least %d' % (x.length, x.atleast)
else:
        print 'No exception was raised.'
```

الخـرج

```
$ python raising.py
Enter something -->
Why did you do an EOF on me?

$ python raising.py
Enter something --> ab
ShortInputException: The input was of length 2, was expecting at least 3

$ python raising.py
Enter something --> abc
No exception was raised.
```

كيف يعمل

هنا، قمنا بإنشاء نوع من الاستثناء خاص بنا على الرغم من أننا قد لا يمكننا استخدام أي استثناء/خطأ محدد سلفا لأغراض إيضاحية. وهذا النوع من الاستثناء الجديد هو الفئة ShortInputException وهي تحتوي على حقلين هما : length وهو طول المدخلات، و atleast وهو أقل طول يتوقعه البرنامج. الجديد هو الفئة . وهي تحتوي على حقلين :هما length وهو طول المدخلات،

في البند except ،نذكرك بالفئة error أيضا إضافة إلى المتغير الذي يقوم بإجراء المقارنة مع الكائن الخطأ/الاستثناء. والتي تعتبر مماثلة للقيم والمعاملات في استدعاء الدالة. وفي داخل الفئة exceptالخاصة نستخدم الحقل length والحقل atleast لطباعة رسالة مناسبة للمستخدم.

Try..Finally

ماذا لو كنت تقرأ الملف وأردت إغلاق هذا الملف سواء تم رفع استثناء أو لا ؟ ويمكن أن يتم ذلك باستخدام finally اللبنة. علما أنه يمكنك استخدام أحد بنود except جنبا إلى جنب مع لبنة finally لنفس اللبنة try المقابلة لها. ستضطر لتضمين واحد بداخل الآخر إذا كنت ترغب في استخدام كليهما .

استخدام Finally

مثال 13.3 استخدام Finally

```
#!/usr/bin/python
# Filename: finally.py

import time

try:
        f = file('poem.txt')
        while True: # our usual file-reading idiom
                line = f.readline()
                if len(line) == 0:
                        break
                time.sleep(2)
                print line,
finally:
        f.close()
        print 'Cleaning up...closed the file'
```

الخرج

```
$ python finally.py
Programming is fun
When the work is done
Cleaning up...closed the file
Traceback (most recent call last):
  File "finally.py", line 12, in ?
    time.sleep(2)
KeyboardInterrupt
```

كيف يعمل :

نقوم كالمعتاد بمجموعة العمل الخاصة بقراءة الملف، ولكني كنت أدخلت طريقة اعتباطية من الإسبات لمدة 2 ثانية قبل طباعة كل سطر باستخدام طريقة الدالة time.sleep. السبب الوحيد لذلك هو أن البرنامج يعمل ببطء نقوم بطاقم العمل. (بيثون سريعة جدا بطبيعتها).

وعندما يظل البرنامج يعمل، اضغط Ctrl+c لمقاطعة/إلغاء البرنامج

لاحظ أن الاستثناء KeyboardInterrupt يلقى و البرنامج في طريقه للخروج، ولكنه قبل وجود البرنامج، وفي النهاية ينفيذ البند ويتم إغلاق الملف.

الخلاصة

لقد ناقشنا استخدام بيانات try..except و try..finally . ولقد رأينا كيف ننشئ منطقتنا أنواع استثناء خاصة بنا وكيفية رفع الاستثناءات كذلك.
وفي الفصل المقبل سنقوم باستكشاف مكتبة بيثون القياسية.

فصل. 14. مكتبة بيثون القياسية

قائمة المحتويات.

مقدمة :

مكتبه بيثون القياسية متاحة مع كل تركيب لبايثون. وهي تحتوي على عدد هائل من الوحدات/ modules المفيدة جدا. و من الأهمية بمكان أن تعتاد على مكتبة بيثون القياسية ؛ حيث إن معظم المشاكل يمكن حلها بسهولة وبسرعة إذا كنت معتادا على هذه المكتبة

سنبحث بعضا من الوحدات المستخدمة في هذه المكتبة. يمكنك العثور على التفاصيل الكاملة لجميع الوحدات في مكتبة بيثون القياسية في قسم " Library Reference " في الوثائق التي تأتي مع تركيب بيثون الخاص بك.

الوحدة sys :

تحتوي هذه الوحدة " sys" على وظيفة محددة من النظام. وقد رأينا بالفعل قائمة sys.argv التي تحتوي على معطيات سطر الأوامر .

معطيات سطر الأوامر

مثال 14.1 استخدام الوحدة sys.argv

```
#!/usr/bin/python
# Filename: cat.py

import sys

def readfile(filename):
        '''Print a file to the standard output.'''
```

```
        f = file(filename)
        while True:
                line = f.readline()
                if len(line) == 0:
                        break
                print line, # notice comma
        f.close()

# Script starts from here
if len(sys.argv) < 2:
        print 'No action specified.'
        sys.exit()

if sys.argv[1].startswith('--'):
        option = sys.argv[1][2:]
        # fetch sys.argv[1] but without the first two characters
        if option == 'version':
                print 'Version 1.2'
        elif option == 'help':
                print '''\
This program prints files to the standard output.
Any number of files can be specified.
Options include:
  --version : Prints the version number
  --help    : Display this help'''
        else:
                print 'Unknown option.'
        sys.exit()
else:
        for filename in sys.argv[1:]:
                readfile(filename)
```

الخرج

```
$ python cat.py
No action specified.

$ python cat.py --help
This program prints files to the standard output.
Any number of files can be specified.
Options include:
```

```
  --version : Prints the version number
   --help    : Display this help

$ python cat.py --version
Version 1.2

$ python cat.py --nonsense
Unknown option.

$ python cat.py poem.txt
Programming is fun
When the work is done
if you wanna make your work also fun:
        use Python!
```

كيف يعمل:

هذا البرنامج يحاول تقليد الأمر cat المألوف عند مستخدمي لينكس/يونكس. عليك فقط تحديد أسماء بعض الملفات النصية وسوف يقوم الأمر بطباعتها إلى الخرج-output

عندما يعمل برنامج لبيثون بطريقة غير تفاعلية، هناك دائما عنصر واحد على الأقل في قائمة sys.argv هو اسم البرنامج الحالي يصبح عاملا ويكون متاحا كـ sys.argv حيث إن بيثون تبدأ العد من الصفر. ومعاملات سطر الأوامر الأخرى تتبع ذلك العنصر.

لجعل البرنامج سهل الاستعمال علينا أن نمده ببعض الخيارات التي من المؤكد أنها تحدد للمستخدم معرفة المزيد عن البرنامج. نحن نستخدم أول argument لمعرفة ما إذا كان أي من الخيارات محددة لبرنامجنا. إذا كان الخيار --version مستخدما، يتم طباعة رقمإصدارة البرنامج. وبالمثل، عندما نحدد الخيار help -- ،نعطي قليل من الشرح حول البرنامج. نحن نستفيد من استعمال دالة sys.exit للخروج من البرنامج. وكما تعودنا دائما، انظر help(sys.exit) لمزيد من التفاصيل.

عندما لا يكون هناك خيارات محددة وأسماء الملفات يتم تمريرها إلى البرنامج، تتم ببساطة طباعة كل سطر من كل ملف، واحدا تلو الآخر في ترتيب محدد على سطر الأوامر.

وبالمناسبة، الأمر cat اختصار لكلمة *concatenate* وهي في الأساس ما يقوم به هذا البرنامج – حيث يمكنه طباعة ملف أو سلسلة ملفات مرتبطة أو ملحقة، اثنان أو أكثر من الملفات معا على الشاشة أو الخرج/output.

المزيد عن sys :

السلسلة النصية sys.version تعطيك معلومات عن إصدارة بيثون التي قمت بتثبيتها . و الصف- tuple المسماة sys.version_info تعطيك طريقة أسهل لإتاحة أجزاء محددة من إصدارة بيثون لبرنامجك .

```
[swaroop@localhost code]$ python
>>> import sys
>>> sys.version
'2.3.4 (#1, Oct 26 2004, 16:42:40) \n[GCC 3.4.2 20041017 (Red Hat 3.4.2-6.fc3)]'
>>> sys.version_info
(2, 3, 4, 'final', 0)
```

للمبرمجين المحترفين :

العناصر الأخرى ذات الأهمية في الوحدة sys تتضمن sys.stdin و sys.stdout و sys.stderr تتطابق مع سريان الخرج والدخل والخطأ القياسي في برنامجك على التوالي .

الوحدة os

هذه الوحدة البرمجية تمثل وظيفة عامة لنظام التشغيل . هذه الوحدة لها أهمية خاصة إذا كنت تريد عمل منصات مستقلة لبرامجك - أي إنه يسمح للبرنامج ليكون مكتوبا لكي يعمل على لينوكس أو على ويندوز كذلك من دون أي مشاكل ودون أن يتطلب ذلك أي تغييرات. ومن الأمثلة على ذلك استخدام المتغير os.sep بدلا من عملية تحديد مسار أو بيئة مستقلة لنظام محدد.

أكثر الأجزاء فائدة من الموديل os مدرجة أدناه ومعظمها واضح بذاته.

- السلسلة النصية os.name تحدد المنصة التي تستخدمها ، فمثلا "nt" لويندوز و 'posix' لمستخدمي لينكس/يونيكس .
- الدالة os.getcwd() للحصول على دليل العمل الحالي ، مثل الدليل الحالي الذي يعمل عليه سكريبت لبايثون .
- الدوال os.getenv() و os.putenv() تستخدم للحصول على أو إعداد متغيرات البيئة على التوالي .
- الدالة os.listdir() تعيد أسماء كل الملفات والمجلدات في الدليل الحالي .
- الدالة os.remove() تستخدم لحذف أحد الملفات .

- الدالة os.system() تستخدم لتشغيل أمر للصدفة .
- السلسلة النصية os.linesep تفيد إنهاء السطر المستخدم على المنصة الحالية على سبيل المثال يستخدم وندوز '\r\n' ، ويستخدم لينكس '\n' ، ويستخدم ماك '\r'
- الدالة os.path.split() تعيد اسم الدليل واسم الملف في المسار .

```
>>> os.path.split('/home/swaroop/byte/code/poem.txt')
('/home/swaroop/byte/code', 'poem.txt')
```

- الدوال os.path.isfile() و os.path.isdir() تستخدم لفحص ما إذا كان المسار المعطى يشير إلى ملف أو مجلد على التوالي . وبالمثل ، الدالة os.path.exists() تستخدم لمعرفة ما إذا كان المسار المعطى موجودا بالفعل .

يمكنك البحث في وثائق بيثون القياسية لمزيد من التفاصيل . ويمكن أن تستخدم help(sys) كذلك .

الخلاصة

قد رأينا بعضا من وظائف الوحدة sys في مكتبة بيثون القياسية . ينبغي عليك أن تبحث في وثائق بيثون القياسية لتحصل على المزيد حولها والمزيد من الوحدات كذلك .

وفي الفصل التالي سوف نغطي جوانب متنوعة من بيثون ،والتي ستجعل جولتنا في بيثون أكثر اكتمالاً .

فصل.15. المزيد من بيثون

جدول المحتويات

الآن وقد قمنا بنجاح بتغطية جوانب رئيسة ومتنوعة من بيثون والتي سوف تستخدمها ، في هذا الفصل سنغطي المزيد من الجوانب التي تجعل معرفتنا بلغة بيثون أكثر اكتمالا

الطرق الخاصة

هناك بعض الأساليب الخاصة التي لها أهمية خاصة في الفئات/classes مثل طرق __init__ و __del__ والتي لها أهمية قد شهدناها بالفعل.
عموما ، الطرق الخاصة تستخدم لتقليد سلوك معين. فعلى سبيل المثال ، إذا أردت أن تستعمل x[key] لعملية الفهرسة الخاصة بك من أجل فئة لديك (مثل التي تستخدمها في القوائم والصفوف tuples) ثم مجرد تنفيذ طريقة __getitem__() ويتم عملك . إذا كنت تفكر في ذلك ، فهذا ما تقوم بيثون بعمله مع الفئة list نفسها!

بعض هذه الطرق/Methods المفيدة الخاصة مدرجة في الجدول التالي. إذا كنت تريد أن تتعرف على كل الطرق الخاصة ، هناك قائمة ضخمة متاحة في الدليل المرجعي لبايثون.

جدول 15.1 بعض الأساليب الخاصة

الاسم	الشرح
__init__(self, ...)	وهذه الطريقة تستدعى فقط عندما يعود الكائن المنشأ حديثا للاستعمال
__del__(self)	تستدعى فقط قبل أن يتم تدمير الكائن

الاسم	الشرح
__str__(self)	تستدعى عندما نستخدم الإفادة print مع كائن أو عندما نستخدم str()
__lt__(self, other)	وبالمثل يوجد أساليب خاصة لجميع العوامل "+،>،إلخ" " Less than "> تستدعى عند استخدام العامل
__getitem__(self, key)	تستدعى عند استخدام عملية الفهرسة x[key]
__len__(self)	تستدعى عند استعمال الدالة المدمجة len() لكائن المتسلسلة

لبنات الإفادات المفردة

والآن، ينبغي أن يكون لديك فهم راسخ أن كل لبنة من الإفادات تمثل مجموعة من الأجزاء لها مستوى من الإزاحة الخاص بها{راجع معنى الإزاحة في الفصل الرابع}. حسنا ، هذا صحيح بالنسبة لمعظم الأجزاء ، ولكنها ليست دقيقة 100 ٪. إذا كانت لبنة الإفادات لا تتضمن سوى إفادة واحدة ، حينئذ يمكنك أن تحدده على نفس السطر ، لنقل مثلا ، إفادة conditional أو looping .

والمثال التالي يشرح ذلك بوضوح :

```
>>> flag = True
>>> if flag: print 'Yes'
...
Yes
```

كما نرى ، فإن الإفادة الواحدة تستخدم في داخل ذات المكان - وليس كبند مستقل من اللبنة. على الرغم من ذلك ، يمكنك استخدام هذا لجعل برنامجك **أصغر** ، وإنني أوصي بشدة ألا تستخدم طريقة الرمز المختصر short-cut هذه باستثناء حالة التحقق من الأخطاء ، الخ.أحد الأسباب الرئيسية أنه سيكون من الأسهل بكثير إضافة إفادة إضافية إذا كنت تستخدم الإزاحة السليمة .

أيضا لاحظ أنه عند استخدام مفسر بيثون في النمط التفاعلي ، فإن ذلك يساعدك في إدخال البيانات عن طريق تغيير المؤشرات بشكل ملائم. وفي حالة aboe ، بعد أن تدخل الكلمة المفتاحية if ، فإنها تغير المؤشر إلى ... لتشير إلى أن الإفادة لم يتم الانتهاء منها بعد. عندما نكمل الإفادة بهذه الطريقة ، نضغط مفتاح enter لتأكيد أن البيانات قد اكتملت. بعد ذلك ، ينهي بيثون تنفيذ الإفادة كلها والعودة إلى المؤشر القديم وانتظار المدخلات التالية.

القوائم المضمنة

تستخدم لاستخلاص/اشتقاق قائمة جديدة من القائمة الحالية. على سبيل المثال ، لديك قائمة

من الأعداد ، و تريد أن تحصل على قائمة مناظرة مع جميع الأرقام مضروبة في 2 ولكن فقط عندما تكون أكبر من 2. فالقائمة المضمنة تكون نموذجية لمثل هذه الحالات.

استخدام القوائم المضمنة

مثال 15.1 استخدام القوائم المضمنة

```
#!/usr/bin/python
# Filename: list_comprehension.py

listone = [2, 3, 4]
listtwo = [2*i for i in listone if i > 2]
print listtwo
```

الخرج

```
$ python list_comprehension.py
[6, 8]
```

كيف يعمل

هنا ، نشتق قائمة جديدة من خلال تحديد التغيرات التي ينبغي القيام بها (i*2) عندما تتحقق بعض الشروط (if i > 2) . لاحظ أن القائمة الأصلية لا تزال غير معدلة. في الكثير من المرات نستخدم الحلقات/ loops للوصول إلى كل عنصر من عناصر القائمة ، ونفس الشيء يمكن أن يتحقق باستخدام list comprehensions وهي طريقة أكثر دقة ، وإحكاما ، ووضوحا .

استقبال الصفوف والقوائم في الدوال

وهناك طريقة خاصة ، لاستقبال معاملات الدالة مثل الصف tuple أو قاموس باستخدام بادئة * أو ** على التوالي. وهذا أمر مفيد عندما نأخذ متغيرا عدديا من المعاملات في الدالة .

```
>>> def powersum(power, *args):
...     '''Return the sum of each argument raised to specified power.'''
...     total = 0
...     for i in args:
...         total += pow(i, power)
...     return total
...
>>> powersum(2, 3, 4)
25
```

```
>>> powersum(2, 10)
100
```

و بالنظر إلى البادئة* أمام المتغير args،يتم تخزين كل المعاملات المضافة إلى الدالة في args باعتباره تيوبل. وإذا تم استخدام البادئة **

قد استخدمت بدلا من * ، يجب أن ينظر إلى المعاملات لتكون أزواج من مفتاح/قيمة key/value للقاموس.

نماذج لامدا

تستخدم الإفادة lambda لإنشاء كائنات دالة جديدة ثم إرجاعها أثناء وقت التشغيل

استخدام نماذج لامدا

مثال 15.2 استخدام نماذج لامبدا

```
#!/usr/bin/python
# Filename: lambda.py

def make_repeater(n):
        return lambda s: s * n

twice = make_repeater(2)

print twice('word')
print twice(5)
```

الخرج

```
$ python lambda.py
wordword
10
```

كيف يعمل :

هنا ؛ استخدمنا الدالة make_repeater لإنشاء كائنات دالة جديدة في وقت التشغيل/runtime وإرجاعها .

الإفادة lambda تستخدم لإنشاء كائن للدالة . في الأساس ، تأخذ lambda معاملا متبوعا بتعبير واحد فقط والذي يصبح الجسم لهذه الدالة ، وقيمة هذا التعبير يتم إرجاعها من خلال الدالة الجديدة . لاحظ أنه حتى الإفادة print لا يمكن أن تستخدم داخل نموذج لامدا . ولكن تستخدم كعبارات فقط.

إفادات exec و eval

تستخدم لتنفيذ إفادات بيثون التي يتم تخزينها في عبارة نصية أو ملف. على سبيل المثال ، يمكننا توليد سلسلة نصية تحتوي شفرة لبايثون عند وقت التشغيل/ runtime ومن ثمّ تنفيذ هذه الإفادات باستخدام الإفادات exec . وهذا مثال بسيط تراه بأسفل :

```
>>> exec 'print "Hello World"'
Hello World
```

إفادة eval

تستخدم لحساب تعابير بيثون الصالحة التي تخزن في السلسلة النصية كما يتضح في المثال بأسفل:

```
>>> eval('2*3')
6
```

إفادة assert

تستخدم إفادة assert للتأكد من تحقق شيء ماعلى سبيل المثال إذا كنت متأكدا بأن لديك عنصراً واحداً على الأقل تستخدمه في قائمة وتريد التحقق من ذلك وترفع خطأ إذا لم يكن متحققاً حينئذ تعتبر الإفادة assert إفادة مثالية في هذه الحالة وعندما تفشل الإفادة assert يرتفع لنا classname>AssertionError

```
>>> mylist = ['item']
>>> assert len(mylist) >= 1
>>> mylist.pop()
'item'
>>> assert len(mylist) >= 1
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
AssertionError
```

الدالة repr

تستخدم الدالة reprt للحصول على تمثيل قانوني لسلسلة نصية لكائن ما.وتقوم Backticks (وقد يسمى تحويل أو عكس علامات الاقتباس) بنفس الأمر.لاحظ أنه سيكون لديك eval(repr(object)) == object في أغلب الأحيان

```
>>> i = []
>>> i.append('item')
>>> `i`
"['item']"
```

```
>>> repr(i)
"['item']"
```

وببساطة ، تستخدم الدالة repr او backticks للحصول على تمثيل للكائن قابلة للطبع. يمكنك التحكم فيما تعيده الكائنات الخاصة بك للدالة repr من خلال تحديد طريقة __ repr__ في الفئة الخاصة بك.

الخلاصة :

لقد تناولنا المزيد من مميزات بيثون في هذا الفصل، و يمكنك التأكد من أننا لم نغطّ بعد كل ملامح بيثون. ولكن، في هذه المرحلة، نكون قد غطينا معظم ما سوف تستخدمه في تطبيقاتك . وهذا الأمر فيه الكفاية لك لتبدأ أيا من البرامج أنت ستنشئها.
في الفصل المقبل، سوف نناقش كيفية استكشاف المزيد من بيثون.

فصل 16. وماذا بعد؟

قائمة المحتويات

إذا كنت قد قرأت هذا الكتاب بعناية حتى الآن، ومارست كتابة العديد من البرامج، فلابد أنك أصبحت متآلفا ومستريحا مع بيثون الآن. وربما تكون أنشأت بعض البرامج لاستكشاف بعض الأشياء وتطبيق المهارات التي اكتسبتها في بيثون. إذا لم تقم بذلك بالفعل، فعليك أن تفعل. والسؤال الآن هو 'ماذا بعد؟'.

وأود أن اقترح عليك معالجة هذه المشكلة: اصنع لنفسك برنامج دفتر *عناوين* يعمل بسطر الأوامر - باستخدامه يمكنك إضافة أو تعديل أو حذف، أو البحث عن معارفك؛ مثل الأصدقاء والأسرة والزملاء، ومعلوماتهم مثل عنوان البريد الإلكتروني و/أو رقم الهاتف. التفاصيل يجب تخزينها لاسترجاعها في وقت لاحق.

هذا أمر سهل إلى حد ما، إذا تذكرت العناصر المختلفة التي مررنا بها حتى الآن. وإذا كنت تريد توجيهات بشأن كيفية المضي قدما، فإليك هذه التلميحة.

تلميحة (لا يجب عليك أن تقرأ هذا).

. أنشئ فئة لتمثيل معلومات الشخص. استخدم قاموسا لتخزين كائنات الشخص مع جعل اسمه المفتاح. استخدم وحدة cPickle لحفظ الكائنات على القرص الصلب. استخدم الأساليب المدمجة في القاموس لإضافة وحذف وتعديل الأشخاص.

عندما تستطيع فعل هذا، يمكنك أن تدعي أنك مبرمج بيثون. والآن، وعلى الفور أرسل لي بريد شكر لهذا الكتاب العظيم ;-). هذه الخطوة اختيارية ولكني أوصيك بها.

إليك بعض الطرق لمواصلة رحلتك مع بيثون:

البرمجيات الرسومية

مكتبات **الواجهة الرسومية** باستخدام بيثون - تحتاجها لعمل برامجك الرسومية باستخدام بيثون. يمكنك إنشاء برامج مثل IrfanView أو Kuickshow أو أي شيء مثل ذلك باستعمال مكتبات الواجهة الرسومية في بيثون مع الارتباطات الخاصة بها. الارتباطات هي ما يسمح لك بكتابة البرامج ببايثون مع استخدام المكتبات التي تمت كتابتها في حد ذاتها بلغة C

أو C++ أو غيرها من اللغات.

هناك الكثير من الخيارات للواجهة الرسومية باستخدام بيثون:

- PyQt. هذه ارتباطات بيثون بعُدة الأدوات Qt (وهي الأساس الذي بنيت عليه كيدي). Qt سهلة الاستعمال للغاية، وقوية جدا خصوصا نظرا لمصمم Qt ووثائقها المذهلة. يمكنك استخدامها بصورة حرة/مجانية على لينكس، ولكن ستضطر لدفع ثمنها إذا كنت تريد استخدامها على ويندوز. PyQt حرة/مجانية إذا أردت إنشاء برمجيات حرة (GPL) على لينكس/يونكس، وبمقابل إذا أردت إنشاء برمجيات مملوكة. من مصادر PyQt الجيدة 'GUI Programming with Python: Qt Edition' طالع الصفحة الرسمية لمزيد من التفاصيل.
- PyGTK. هذه ارتباطات بيثون بعُدة الأدوات جتك+ "GTK+" (وهي الأساس الذي بنيت عليه جنوم). جتك+ لها غرابتها، ولك بمجرد أن تعتادها سيمكنك إنشاء تطبيقات رسومية بسرعة. المرور بمصمم الواجهة "جلاد" أمر لا غنى عنه. الوثائق ما تزال بحاجة للتحسين. تعمل جتك+ جيدا على لينكس ولكن نقلها إلى ويندوز غير مكتمل. يمكنك إنشاء برمجيات حرة أو مملوكة مستخدما جتك+. طالع الصفحة الرسمية للمزيد من التفاصيل.
- wxPython. هذه ارتباطات بيثون بعُدة الأدوات wxWidgets. د wxPython منحنى تعليمي مرتبط بها، ورغم ذلك فهي محمولة جدا وتعمل على لينكس، و ويندوز، و ماك، وحتى على المنصات المدمجة (embedded platforms). يوجد العديد من بيئات تطوير wxPython التي تشمل مصممات واجهة رسومية مثل (SPE (Stani's Python Editor و مصمم الواجهات wxGlade. يمكنك إنشاء برمجيات حرة أو مملوكة مستخدما wxPython. طالع الصفحة الرسمية للمزيد من التفاصيل.
- TkInter. هذه واحدو من أقدم عُدد أدوات الواجهة الرسومية الموجودة. إذا سبق و استخدمت IDLE فقد رأيت برنامج TkInter يعمل. وثائق TkInter على موقع PythonWare.org شاملة. TkInter محمولة وتعمل على كل من لينكس/يونكس و وندوز على حد سواء. والأهم أن TkInter جزء من توزيعة بيثون القياسية.
- للمزيد من الخيارات، راجع صفحة ويكي البرمجة الرسومية في Python.org

ملخص عن الأدوات الرسومية

- للأسف لا توجد أداة قياسية واحدة للبرمجة الرسومية على بيثون. اقترح أن تختار واحدة من تلك الأدوات المذكورة أعلاه، حسب حالتك. العامل الأول في تحديد اختيارك هو استعدادك للدفع مقابل استخدام أي من الأدوات الرسومية أو لا. العامل الثاني هو المنصة التي تريد أن يعمل عليها برنامجك؛ لينكس أم ويندوز أم كليهما. العامل الثالث هو كونك من مستخدمي جنوم أو كيدي.

الفصول المستقبلية

- لقد فكرت في كتابة فصل أو فصلين لهذا الكتاب عن برمجة الواجهات الرسومية. غالبا سأختار wxPython كعُدّة الأدوات المختارة. إذا أردت أن تعرض وجهة نظرك حول هذا الموضوع فمن فضلك اشترك في قائمة byte-of-python البريدية حيث يتناقش القراء معي حول التحسينات المحتملة لهذا الكتاب.

استكشف المزيد

- **مكتبة بيثون القياسية** مكتبة شاملة. أغلب الوقت، ستجد في هذه المكتبة ما تبحث عنه. وهذا يشار إليه بوصفه فلسفة 'البطاريات مضمّنة' في بيثون. أنا أوصي بشدة بأن تتجول خلال الوثائق القياسية لبايثون قبل المتابعة في بدء كتابة برامج كبيرة بلغة بيثون.
- Python.org: الموقع الرسمي للغة البرمجة بيثون. ستجد أحدث الإصدارات من لغة بيثون ومفسر اللغة. وهناك أيضا مختلف القوائم البريدية حيث تجري المناقشات النشطة حول مختلف جوانب بيثون.
- **comp.lang.python** هي مجموعة أخبار usenet حيث يجري النقاش حول هذه اللغة. يمكنك إرسال رسائلك واستفساراتك إلى مجموعة الأخبار تلك. يمكنك الوصول إلى هذه المجموعات على الإنترنت باستخدام مجموعات جوجل أو الانضمام إلى القائمة البريدية والتي ليست سوى مرآة لمجموعة الأخبار.
- كتاب طبخ بيثون كتاب قيم للغاية حيث جمع مجموعة من الوصفات أو النصائح حول كيفية حل أنواع معينة من المشاكل باستخدام بيثون. هذا الكتاب لا بد أن يقرأه كل مستخدم بيثون.
- بيثون الساحرة هي سلسلة من المقالات الرائعة حول بيثون بقلم داڤيد مرتز.
- غص في بيثون كتاب جيد جدا لذوي الخبرة من مبرمجي بيثون. إذا قرأت تماما الكتاب الحالي الذي تقرأه الآن، فأرجح بشدة أن تقرأ 'غص في بيثون' بعد ذلك. يغطي مجموعة من المواضيع بما في ذلك معالجة XML، والاختبارات الوحداتية، والبرمجة والوظيفية.
- Jython هو تطبيق لمفسر بيثون في لغة جافا. وهذا يعني أن بإمكانك كتابة برامج بلغة بيثون واستخدام مكتبات جافا كذلك! Jython برمجية مستقرة وناضجة. إذا كنت مبرمج جافا، فأنصحك بشدة بأن تجرب Jython.
- IronPython هو تطبيق لمفسر بيثون في لغة سي# ويمكن أن يعمل على منصة نت / مونو / دوتجنو. هذا يعني أن بإمكانك كتابة برامج بلغة بيثون واستخدام مكتبات NET والمكتبات الأخرى التي توفرها هذه المنصات الثلاث كذلك. Iron python ما تزال برمجية "قبل ألفا" ولا تصلح سوى للتجريب حاليا. جِم هوجونن، الذي كتب IronPython انضم إلى شركة ميكروسوفت، وسيعمل على إصدارة كاملة من IronPython في المستقبل.

- Lython هو واجهة ليسب للغة بيثون. وهي تشبه ليسب المشتركة (Common Lisp) وتترجم مباشرة إلى bytecode بيثون، الأمر الذي يعني أنها سوف تتفاعل مع كود بيثون المعتاد.
- هناك الكثير والكثير من الموارد عن بيثون. المهمة منها Daily Python-URL! الذي يجعلك على اطلاع دائم بآخر الأحداث، و Vaults of Parnassus، و ONLamp. com، Python DevCenter، و dirtSimple.org، و Python Notes وغيرها الكثير.

الخلاصة

- لقد وصلنا إلى نهاية هذا الكتاب ولكن، كما يقولون *هذه بداية النهاية*. أنت الآن تعتبر مستخدما نهِما لبيثون، وأنت بلا شك مستعد لحل العديد من المشاكل باستخدام بيثون. يمكنك البدء في أتمتة جهازك للقيام بكل الأشياء التي كانت في الماضي أمورا لا يمكن تخيلها، أو كتابة ألعابك الخاصة والكثير الكثير. ولذا ابدأ.

ملحق A. البرمجيات الحرة مفتوحة المصدر

- البرمجيات الحرة مفتوحة المصدر (Free/Libré and Open Source Software، FLOSS) مبنية على مبدأ الجماعة، وهي بدورها مبنية على مبدأ المشاركة وخاصة المشاركة في المعرفة. البرمجيات الحرة حرة الاستخدام، والتعديل والتوزيع.
- إذا كنت قد قرأت هذا الكتاب بالفعل، فأنت على معرفة بالبرمجيات الحرة بما أنك كنت تستخدم **بيثون** طوال الوقت.
- إذا أردت معرفة المزيد عن البرمجيات الحرة، فيمكنك استكشاف القائمة التالية. لقد ذكرت بعض البرمجيات الحرة الكبيرة مع برمجيات حرة متعددة المنصات (أي تعمل على لينكس و ويندوز إلخ.) ليمكنك تجربة استخدام هذه البرمجيات دون الحاجة للانتقال إلى لينكس مباشرة *وإن كنت ستفعل عاجلا أم آجلا ;-)*
- **لينكس**.هو نظام تشغيل حر ومفتوح المصدر يحتضنه كل العالم ببطء ، كان قد بدأه لينوس تورفالدز كطالب. والآن ، فإنه هو منافس لنظام مايكروسوفت ويندوز. وتعد آخر نواة 2.6 إصدارة رئيسية تتمتع بالسرعة والاستقرار ,قابلة للتطوير. [Linux Kernel]
- **نوبكس**.هذه توزيعة لينكس تعمل من خلال القرص المدمج ولا تتطلب التثبيت على القرص الصلب -- يمكنك فقط إعادة تشغيل حاسبك ثم وضع الإسطوانة في المشغل والبدء باستخدام جميع مميزات توزيعة لينكس! ويمكنك استخدام جميع البرامج مفتوحة المصدر التي تأتي مع توزيعة لينكس القياسية مثل تشغيل برامج بيثون ، وعمل تصريف(compiling) لبرامج سي ، ومشاهدة الأفلام ، وغير ذلك ، وبعد ذلك يمكنك إعادة تشغيل حاسبك وإخراج الإسطوانة والعودة لاستخدام نظام التشغيل الحالي الخاص بك ، وكأنشيئا لم يحدث.[Knoppix]
- **فيدورا**. تلك توزيعة ومجتمع يتم توجيهها ورعايتها من قبل رد هات وهي من أكثر توزيعات لينكس. شعبية وهي تحتوي على نواة لينكس ،وأسطح مكتب كيدي ، وجنوم XFCE وعدد كبير من البرامج مفتوحة المصدر المتاحة وكل هذا في صورة سهلة الاستعمال وسهلة التركيب.وإذا كنت مبتدئا في استخدام لينكس فإني أوصيك بشدة بأن تجرب استخدام لينكس ماندريك وأحدث إصدارة منها ماندريك 10ر1 وهي توزيعة رائعة [Fedora Linux, Mandrake Linux]
- **أو بن أوفيس**. هذا برنامج مكتبي ممتاز قائم على برنامج StarOffice من شركة صن ميكروسيستمز ويحتوي على برامج للكتابة والرسم والجداول والشرائح وأمور أخرى . ويمكنه أيضا فتح وتحرير ملفات ميكروسوفت وورد وباور بوينت بسهولة وهي تعمل على كل المنصات والإصدار القادم من أوبن أوفيس يحتوي على بعض التحسينات الجذرية[

[OpenOffice

- **فايرفوكس موزيلا**. هذا هو الجيل القادم من متصفح الويب الذي يتوقع تغلبه على إنترنت اكسبلورر (من حيث الحصة السوقية فقط ;-) في غضون سنوات قليلة. ومن في غاية السرعة وقد نال الإشادة من النقاد لخصائصه المعتبرة والمثيرة للإعجاب. تمديد مفهوم يسمح بأي نوع من الوظائف التي يمكن ان تضاف اليها وقد أضيفت إليه العديد من الإضافات تسمح بإضافة أي نوع من الوظائف إليه.والمنتج المصاحب له (ثندربيرد) عميل بريد إلكتروني ممتاز يجعل قراءة البريد أمرا غاية في السرعة . [Mozilla Firefox, Mozilla Thunderbird]

- **مونو**. هذا تطبيق مفتوح المصدر لتشغيل برامج منصة مايكروسوفت. نت . وهي تتيح إنشاء وتشغيل تطبيقات دوت نت للعمل في بيئة لينكس ، ويندوز ، FreeBSD ، و ماكنتوش وغيرها من المنصات الأخرى أيضا. لتنفذ معايير ECMA لواجهة سطر الأوامر **CLI** ولغة سي# التي اعتمدتها مايكروسوفت ، وإنتل ، وإتش بي كمعايير موحدة وأصبحت الآن مفتوحة المعايير. وهذه خطوة في اتجاه توحيد معايير **ISO** في الوقت نفسه.

وحاليا يوجد برمجيات mcs C# (كتبت بلغة سي# نفسها)،ومزودات ASP.NET و ADO.NET كاملة الخصائص، لقواعد البيانات والعديد والعديد من المزايا التي يتم تطويرها وإضافتها كل يوم.[Mono, ECMA, Microsoft .NET]

- **خادم الويب أباتشي**. هو أكثر خوادم الويب مفتوحة المصدر شعبية وفي الواقع هو أشهر خادم ويب في العالم! حيث إنه يعمل عليه قريب من 60% من المواقع الإلكترونية . نعم هذه حقيقة حيث يتعامل أباتشي مع مواقع إلكترونية أكثر بكثير من المنافسين بما فيهم خادم ميكروسوفت. [Apache]

- **MySQL**. هو واحد من أشهر خوادم قواعد البيانات مفتوح المصدر ، وهو غاية في السرعةوقد تم إضافة العديد من المزايا إلى أحدث إصداراته. [MySQL]

- **إم بلاير** هو مشغل فيديو يمكنه تشغيل أي شيء بدءا ب DivX و MP3 و Ogg و VCD وحتى DVDs (من ذا يقول إن المصادر المفتوحة لا تتمتع بروح الفكاهة؟) [MPlayer]

- Movix. هي توزيعة لينكس تقوم على توزيعة كنوبكس وتعمل من خلال القرص المدمج ولكن تم تصميمها لتشغيل إسطوانات الأفلام ذاتية الإقلاع . ويمكنك عمل إسطوانات موفيكس، وما هي إلا إسطوانات أفلام ذاتية التشغيل فعندما تقوم بإعادة تشغيل الحاسب وتضع القرص المدمج في مشغل الأقراص وبعدها يبدأ تشغل الفيلم ذاتيا، وأنت لا تحتاج حتى لقرص صلب لمشاهدة الفيلم باستخدام موفيكس. [Movix]

هذه القائمة تهدف فقط إلى إعطائك فكرة موجزة، وهناك العديد من البرمج المفتوحة المصدر الممتازة غير ذلك، مثل لغة بيرل، ولغة بي إتش بي، و نظام إدارة المحتوي للمواقع

Drupal، وخادم قواعد بيانات PostgreSQL،ولعبة سباق السيارات TORCS، وبيئة التطوير المتكاملة Kdevelop،و Anjuta، ومشغل الأفلام Xine، ومحرر النصوص VIM و Quanta+، ومشغل الصوتيات XMMS، وبرنامج تحرير الصور جيمب، ويمكننا المضي قدما إلى ما لا نهاية

- زر هذه المواقع لمزيد من المعلومات عن البرمجيات الحرة:
- سورس فورج
- FreshMeat
- كدي
- جنوم

لتعرف آخر أخبار عالم البرمجيات الحرة، طالع هذه المواقع:

- OSNews
- LinuxToday
- NewsForge
- مدونة سواروب س. هـ.
- والآن، اذهب واستكشف العالم الواسع للبرمجيات الحرة مفتوحة المصدر.

ملحق B. عنْ

قائمة المحتويات

بيانات الطبع

- تقريبا كل البرمجيات التي استخدمتها لبناء هذا الكتاب حرة ومفتوحة المصدر. في أول مسودة من الكتاب استخدمت ردهات لينكس 9. 0 كقاعدة لبيئة العمل، والآن في هذه المسودة السادسة أستخدم لينكس فيدورا كور 3 كأساس.
- في البداية استخدم KWord لكتابة الكتاب (كما وضحت في درس التاريخ في التمهيد). لاحقا تحولت إلى DocBook XML مستخدما Kate لكن وجدتها مشقة كبيرة، لذا انتقلت إلى أوبن أوفيس والذي كان رائعا بفضل مستوى التحكم الذي يوفره للتنسيق وتوليد ملفات PDF، لكنه أنتج HTML سيء جدا من المستند. في النهاية اكتشفت XEmacs وأعدت كتابة الكتاب من الصفر مستخدما DocBook XML (للمرة الثانية) بعد أن قررت أن هذا التنسيق هو حل بعيد المدى. في هذه المسودة السادسة الجديدة قررت استخدام Quanta+ للقيام بكل التحرير.
- تستخدم صفحات الطرز القياسية التي تأتي مع فيدورا كور 3. كما تستخدم الخطوط القياسية أيضا. مع هذا، فقد كتبت مستند CSS لتلوين وتنسيق صفحات HTML. كما كتبت محلل نحوي بدائي -بلغة بيثون طبعا- لإبراز التراكيب آليا في كل نماذج البرامج.

عن المؤلف

- يحب سواروپ س. ه. عمله كمطور برمجيات في ياهو! بمكتب بنجالور في الهند. اهتماماته في المجال التقني تشمل البرمجيات الحرة مثل لينكس، دوتجنو و Qt و MySQL، واللغات الرائعة مثل بيثون و سي#، وكتابة أشياء مثل هذا الكتاب وأي برمجية يستطيع إنشائها في وقت فراغه، والكتابة في مدونته. اهتماماه الأخرى تشمل القهوة، وقرائة روايات روبرت لودلم، والرحلات والسياسة.
- إذا كنت تهتم بمعرفة المزيد عن هذا الرجل، فانظر في مدونته www.swaroopch.info.

ملحق C. تأريخ المراجعة

قائمة المحتويات

الختم الزمني

وُلِّد هذا المستند في 13يناير 2005 الساعة 04:43

تاريخ المراجعة

مراجعة 1. 20 13/01/2005

إعادة كتابة شاملة باستخدام Quanta+ على فيدورا 3 بالإضافة إلى الكثير من التصحيحات والتحديثات. الكثير من الأمثلة الجديدة. إعادة كتابة بيئة DocBook الخاصة من البداية.

مراجعة 1. 15 28/03/2004

مراجعات طفيفة

مراجعة 1. 12 16/03/2004

إضافات وتصحيحات

مراجعة 1. 10 09/03/2004

المزيد من تصحيح الأخطاء الإملائية، بفضل الكثير من القراء المتحمسين والمخلصين.

مراجعة 1. 00 08/03/2004

كنتيجة للمردود والاقتراحات الكثيرة من القراء، قمت بمراجعات كبيرة على المحتوى مع تصحيح للأخطاء الإملائية.

مراجعة 0. 99 22/02/2004

إضافة فصل جديد عن الوحدات. إضافة تفاصيل عن عدد المعاملات المتغير للدوال.

مراجعة 0. 98 16/02/2004

كتبت مخطوط بيثون و صفحات طرز CSS لتحسين ناتج XHTML، شاملا محلل نحوي بدائي -لكن يعمل- للقيام بإبراز للتراكيب في نماذج البرامج على طريقة VIM.

مراجعة 0. 97 13/02/2004

مسودة جديدة معاد كتابتها بالكامل، بتنسيق DocBook XML (ثانيا). تحسن الكتاب كثيرا، أصبح أكثر تناسقا ومقروئية.

مراجعة 0. 93 25/01/2004

إضافة الحديث عن IDLE مع المزيد من المسائل الخاصة بويندوز

مراجعة 0. 92 05/01/2004

تعديلات على بعض الأمثلة.

مراجعة 0. 91 30/12/2003

تصحيح بعض الأخطاء الإملائية. مواضيع عديدة مرتجلة.

مراجعة 0. 90 18/12/2003

إضافة فصلين جديدين. تنسيق أوبن أوفيس مع مراجعات.

مراجعة 0. 60 21/11/2003

إعادة كتابة بالكامل مع توسع.

مراجعة 0. 20 20/11/2003

تصحيح بعض الأخطاء الإملائية.

مراجعة 0. 15 20/11/2003

التحويل إلى DocBook XML.

مراجعة 0. 10 14/11/2003

المسودة الأولى باستخدام KWord.

الحمد لله الذي بنعمته تتم الصالحات

تم الإنتهاء من مراجعة هذا الكتاب في ضحى يوم الأحد 21 من جمادى الآخرة 1430 هـ الموافق 14 من يونيو 2009 م

ترجمه : أشرف علي خلف

الإسكندرية – مصر

ashrafkhalaf@gmail.com

sharaf969@yahoo.com