2008

Byte of Python

خطوة على طريق بايثون

Swaroop C H

ترجمة : أشرف علي خلف

Kspersky0

www.linuxac.org

هدية لمجتمع لينكس العربي

كتاب Byte Of Python

تأليف : Swaroop C H

موقع الكتاب www.byteofpython.info

الإصدار رقم Version 1.20

حقوق النشر Copyright © 2003-2005 Swaroop C H

المقدمـة

قائمة المحتويات

مقدمة- لمن هذا الكتاب - درس تاريخ - حالة الكتاب - الموقع الرسمي - شروط الترخيص – اقتراح - أمور يجب التفكير فيها

**مقدمة

بايثون هي واحدة من تلك اللغات القليلة التي يمكننا الادعاء أنها تجمع بين كل من: البساطة والقوة.إنها لغة.جيدة للمبتدئين وللمحترفين على حد سواء ،والأمر الأهم المتعة مع البرنامج .
يهدف الكتاب الى مساعدتك في تعلم هذه اللغة الرائعه ويريك كيفية إنجاز الأمور بسرعة وبشكل غير متعب – وفى
الواقع ' هو مكافح مثالي ضد سم ومشاكلك البرمجية.

**لمن هذا الكتاب؟

هذا الكتاب بمثابة دليل تعليمي للغة البرمجة بايثون. وهي تستهدف أساسا المبتدئين. وهي مفيدة للمبرمجين ذوي الخبرة كذلك، والهدف من ذلك عموما هو أنه كل ما عليك معرفته عن أجهزة الكمبيوتر هو كيفية حفظ الملفات النصية ثم يمكنك أن تتعلم بايثون من هذا الكتاب وإذا كان لديك خبرة مسبقة عن البرمجة ، يمكنك أيضا ان تتعلم بايثون من هذا الكتاب

إذا كنت صاحب خبرة مسبقة بالبرمجة، فستكون مهتما بأوجه الاختلاف بين بايثون ولغة البرمجة المفضلة لديك لقد ألقيت الضوء على الكثير من هذه الاختلافات .
على الرغم من ذلك لي تنبيه بسيط ، بايثون عما قريب سوف تصبح لغة البرمجة المفضلة لديك ☺☺ !!

**درس تاريخ!!

في أول الأمر بدأت مع بايثون عندما احتجت إلى برنامج تثبيت لبرنامجي ،لذا استطعت أن أجعل التثبيت سهلا وكان علي الاختيار بين بايثون و بيرل ،مع أغلفة مكتبة Qt.
و قمت بإعادة البحث في شبكة الإنترنت حتى عثرت بالصدفة على مقالة ل إيريك إس رايموند ذلك الهاكر المبجل ،.والمشهور يتكلم فيها عن كيف أصبحت بايثون هي لغة البرمجة المحببة لديه
وكذلك اكتشفت أن أغلفة PyQt جيدة جدا بالمقارنة مع Perl-Qt لذلك قررت أن بايثون هي اللغة الخاصة بي بعدها بدأت البحث عن كتاب جيد في لغة بايثون. ولكني لم أجد أيا منها!! , وقد وجدت بعض الكتب لـ O'Reilly ولكنها كانت إما باهظة الثمن للغاية، أو تشبه إلى حد كبير مقدمات أقرب من كونها مراجع.

وبالتالي اتجهت إلى الوثائق التي جاءت مع بايثون، ومع ذلك كانت مختصرة جدا وصغيرة، وقد أعطتني فكرة جيدة عن بايثون، ولكنها لم تكن مكتملة، وغير وقد أمكنني التعامل معها حيث كان لدي خبرة مسبقة بالبرمجة، ولكنها غير ملائمة للمبتدئين

بعد ستة أشهر من أول لقاء لي مع بايثون قمت بتثبيت آخر توزيعة من ردهات Red Hat 9 Linux ،
وكنت ألعب حول أهم ما فيها ، وكنت أزداد إثارة وفجأة خطرت لي فكرة كتابة مادة عن بايثون ،
وقد بدأت الكتابة بقليل من الصفحات،ولكنها سريعا أصبحت ثلاثين صفحة طويلة ،بعدها صبحت جادا في عمل فائدة أكبر على شكل كتاب
وبعد العديد من إعادة الكتابات، أصبح في مرحلة كونه مرجعا مفيدا في تعلم لغة بايثون ، وأنا أأمل أن يكون هذا الكتاب مساهمة مني وتحية لمجتمع المصادر المفتوحة

وهذا الكتاب بدأ كملاحظات شخصية عن بايثون ولكني ما زلت أنظر فيه في نفس الوقت - رغم أني بذلت فيه الكثير من الجهد - ليكون أكثر قبولا عند الآخرين .
ومن خلال الروح الحقيقية لمجتمع المصدر المفتوح، تلقيت الكثير من الاقتراحات البناءة، والانتقادات وردود فعل متحمسة من القراء مما ساعدني كثيرا على تحسين هذا الكتاب.

**حالة الكتاب

هذا الكتاب مازال قيد العمل حيث إن الكثير من الفصول تتغير باستمرار وتتحسن، ومع ذلك فإن الكتاب قد نضج كثيرا، وستكون مستعدا لتعلم بايثون بسهولة من هذا الكتاب.
من فضلك أخبرني إن وجدت أي جزء في هذا الكتاب غير صحيح أو غير مفهوم
أكثر الفصول لها خطط مستقبلية ، مثل:,wxpython Twisted وربما حتى Boa Constructor

** الموقع الرسمي

الموقع الرسمي لهذا الكتاب هو http://www.byteofpython.info
ومن خلال الموقع يمكنك قراء الكتاب كاملا بشكل مباشر أو تنزيل آخر إصدار للكتاب، وكذلك إرسال الملاحظات لي

**شروط الترخيص

هذا الكتاب مرخص بموجب رخصة الإبداع العامة غير التجارية شبه المشاركة.
The Creative Commons Attribution- NonCommercial -ShareAlike License

أساسا ؛ لك الحرية في نسخ وتوزيع، وعرض الكتاب، طالما أنك تنسب الفضل لي. القيود هي أنه لا يمكنك استخدام الكتاب لأغراض تجارية بدون إذن مني.
لك الحرية في التعديل والبناء على هذا العمل، شريطة أن تقوم بوضع علامات واضحة لكل التغييرات وإصدار العمل المعدل تحت نفس الرخصة كما في هذا الكتاب
لقراءة النص الكامل من الرخصة الأصلية Creative Commons من فضل قم بزيارة موقع الرخصة أو لسهولة فهم النسخة حتى إنه يوجد بالموقع شريط مضحك لشرح الرخصة.

**اقتراح

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

مسائل يجب للتفكير فيها

هناك طريقتان لبناء تصميم البرنامج أحدها هو جعلها بسيطة جدا حيث إنها بوضوح و بلا عيوب وأما الأخرى ففيها من التعقيد بحيث لا يتضح بها أوجه القصور .. (C. A. R. Hoare))
النجاح في الحياة لا يهم فيه الذكاء والموهبة بقدر ما يهم فيها التركيز والمثابرة (C. W. Wendte)

الفصل الأول

المقدمة

قائمة المحتويات:

مقدمـــة

* بايثون هي واحدة من تلك اللغات القليلة التي يمكن الادعاء أنها بسيطة وقوية على حد سواء. ستكون مسرورا ومتفاجئا في كم هي سهلة وتركز في حل المشكلة بالمقارنة مع تراكيب وأساسيات أية لغة برمجة تعمل عليها

المقدمة الرسمية لبايثون هي :

بايثون هي واحدة من لغات البرمجة سهلة التعلم، قوية. وتحتوي بكفاءة عالية المستوى وبسيطة على هياكل البيانات ولكنها فعالة لعمل البرمجة الكائنية.
أناقة قواعد بايثون وديناميكية الكتابة فيها، جنبا إلى جنب مع طبيعة تفسيرها، تجعل من بايثون لغة مثالية لبرمجة السكربتات وسرعة تطوير التطبيق في العديد من المجالات على معظم المنصات.
سأناقش معظم هذه السمات بمزيد من التفصيل في القسم التالي.

ملاحظة:
" Guido van Rossum غويدو فان روسام " مؤلف لغة بايثون أطلق عليها ذلك الاسم بعد رؤيته عرضا لهيئة الإذاعة البريطانية باسم "سيرك مونتي للثعابين الطائرة" "Monty Python's Flying Circus" وقال إنها مثل الثعابين التي تقتل الحيوانات لتتغذى عليها عن طريق تصفية جسدها بالالتفاف حولها ، وسحقها .

مميزات لغة بايثون:

**بسيطة:
بايثون لغة بسيطة لأبعد الحدود. إن قراءة برنامج جيد لبايثون يكاد يشبه قراء ة اللغة الإنكليزية على الرغم من أنها إنجليزية صارمة!
طبيعة هذا الاسم المستعار لبايثون هو واحد من أعظم أسرار قوتها. إنه يتيح لك التركيز على حل المشكلة أكثر من اللغة نفسها.

**سهلة التعلم:
كما سترون، بايثون سهلة للغاية لتبدأ بها في تعلم البرمجة. بايثون تحتوي تراكيب سهلة وعادية، كما سبق ذكره.

**حرة ومفتوحة المصدر:
بايثون هي مثال لمصطلح FLOSS (Free/Libré and Open Source Soft-ware) البرامج الحرة والمفتوحة المصدر.
بعبارات بسيطة، يمكنك بحرية توزيع نسخ من هذه البرمجيات، وقراءة شفرة المصدر، و تقوم ببعض التغييرات عليها واستخدام أجزاء منها في برمجيات حرة جديدة، وأنت تعرف أنه يمكنك أن تفعل هذه الأشياء.
يقوم مفهوم مصطلح FLOSS على مبدأ المجتمع الذي يتشارك في المعرفة.

هذا واحد من أسباب كون بايثون جيدة جدا - لأنه قد تم إنشاؤها وتحسينها بشكل مستمر من خلال المجتمع الذي يريد فقط أن يرى بايثون أفضل.

لغة برمجة رفيعة المستوى:
عندما تكتب البرامج في بايثون، لا تحتاج أبدا إلى الضيق بالتفاصيل دقيقة المستوى مثل إدارة الذاكرة التي يستخدمها برنامجك، الخ.

** محمولة:
نظرا لطبيعة البرامج المفتوحة المصدر، تم جعل بايثون لغة محمولة (أي تم جعلها تعمل على) العديد من المنصات. كل ما تصنعه من برامج بلغة بايثون يمكنها أن تعمل على أي من هذه المنصات دون أن يتطلب ذلك أي تغييرات على الإطلاق. إذا كنت دقيقا بما فيه الكفاية لتجنب أي اعتماديات خاصة للنظام.

هذه المنصات يمكنك استخدام بايثون على :
Linux, Windows, FreeBSD, Macintosh, Solaris,
OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX,
VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus,
وحتى الكمبيوتر الكفي !windows ce pocketpc

** لغة مفسرة :
**وذلك يتطلب شيئا من الشرح
برنامج مكتوب في لغة مجمعة/مترجمة مثل C أو C++ أو يتم تحويلها من مصدر اللغة. C أو C++ إلى اللغة التي يتكلمها جهازك "ثنائية الكود" (binary code i.e. 0s and 1s) باستخدام المترجم مع مختلف الخيارات والتعليمات. عند تشغيلك البرنامج، يقوم linker/loader بنسخ البرنامج من القرص الصلب إلى الذاكرة ويبدأ في تشغيله.

بايثون، من ناحية أخرى، لا تحتاج إلى الترجمة/التجميع إلى الكود الثنائي. فقط شغل البرنامج مباشرة من كود المصدر. داخليا، فإن بايثون يحول شفرة المصدر إلى شكل وسيط يسمى bytecodes ثم يترجم هذا إلى اللغة الأصلية لجهازك، ثم يشغله. كل هذا، في الواقع، يجعل من الأسهل بكثير استخدام بايثون حيث إنه ليس عليك أن تشعر بقلق من ناحية تجميع البرنامج، أو التأكد من صحة مكتبات الربط وتحميلها، الخ، الخ وهذا أيضا يجعل برامج بايثون الخاصة بك أكثر محمولية، بحيث يمكنك مجرد نسخ برنامج بايثون الخاص بك على جهاز كمبيوتر أخر، وبعدها يعمل!

لغة كائنية التوجه Object Oriented

بايثون تدعم البرمجة الإجرائية الموجهة/ procedure-oriented وكذلك البرمجة الكائنية الموجهة/ object-oriented. ففي اللغات التي تدعم البرمجة الإجرائية الموجهة/ procedure-orientedفإن البرنامج يتمحور حول الإجراءات أو الدوال التي ليست سوى قطعة من البرامج يمكن إعادة استخدامها. وفي لغات البرمجة الكائنية ، فإن البرنامج يتمحور حول الكائنات/ objects التي تجمع فيما بين البيانات والوظائف. ولغة بايثون قوية جدا ولكن بطريقة تبسيطية لعمل ***O**bject-**O**riented **P**rogramming* {oop}، وبخاصة عند مقارنتها بلغات كبيرة مثل C++ أو جافا.

قابلة للامتداد Extensible
إذا كنت في حاجة ماسة إلى قطعة من الكود ليعمل سريعا جدا أو تريد أن يكون لديك بعض القطع من خوارزمية لا تكون مفتوحة، يمكنك كتابة هذا الجزء من برنامجك بلغة C أو C++ وبعدها تستخدمه من برنامج بايثون الخاص بك.

** قابلة للتضمين Embeddable

يمكنك تضمين بايثون ضمن برامج C/C++ لإعطاء قدرات ال'scripting' لمستخدمي برنامجك.

**المكتبات الشاملة/المتسعة Extensive Libraries
مكتبة بايثون القياسية مكتبة ضخمة في الواقع. تذكر،ساعدك على عمل مختلف الأشياء العادية بما فيها: regular expressions, documentation generation, unit testing, threading, databases, web browsers, CGI, ftp, email, XML, XML-RPC, HTML, WAV files, cryptography, GUI (graphical user interfaces), Tk وغيرها من الأشياء التي تعتمد على النظام. تذكر ، كل هذا متاح دائما أينما يثبت بايثون. وهذا ما يسمى في فلسفة بايثون.(بطاريات الشحن الإضافية) 'Batteries Included'.

إلى جانب ذلك؛ بالنسبة للمكتبات القياسية؛ توجد العديد من المكتبات المتنوعة الأخرى عالية الجودة مثل: wxPython, Twisted, Python Imaging Library والكثير والكثير.

خلاصة
بايثون لغة مثيرة وقوية حقا . إنها في الحقيقة تجمع بين مزيج من حسن الأداء والميزات التي تجعل كتابة برامج بايثون تجمع بين كل من السهولة والمتعة.

لماذا ليس بيرل؟
إذا كنت لا تعرف فعلا، بيرل 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! !
وبعبارة أخرى، فإن بيرل برامجه سهلة عندما تكون صغيرة، وهو يبرع في البرامج الصغيرة والسكربتات والهاكات لإنجاز العمل.
وعلى أية حال؛ سرعان ما تصبح هذه البرامج جامدة بمجرد البدء في كتابة برامج أكبر، وأنا أتحدث من واقع تجربة كتابة برامج كبيرة بلغة بيرل في ياهو!
وبالمقارنة مع بيرل، فإن البرامج على بايثون هي بالتأكيد أسهل، وأكثر وضوحا، وأسهل في الكتابة وبالتالي أكثر قابلية للفهم وللصيانة.
أنا معجب بلغة بيرل وأقوم باستخدامها بشكل أساسي يوميالألأمور متنوعة ، ولكني كلما كتبت برنامجا، فدائما أبدأ التفكير في بايثون؛ حيث إنها أصبحت طبيعية جدا بالنسبة لي.
خضعت لغة بيرل لعدد كبير من التغييرات والهاكات، وتشعر أنها على غرار واحدة (ولكنها واحدة من جحيم) الهاك.
ومن المحزن أن إصدارة بيرل ٦ المقبلة لا يبدو أنها قامت بإجراء أي تحسينات تتعلق بهذا.
الميزة الوحيد الهامة جدا والتي أشعر بها في بيرل هي المكتبة الضخمة لـ CPAN (the Comprehensive Perl Archive Network) {الأرشيف الشامل لبيرل على الشبكة}وكما يوحي الاسم ،هو جمع مزيج من وحدات/ modules بيرل وهو ببساطة مذهل للعقل نظرا للحجم والعمق -- يمكنك القيام به عمليا بأي شيء يمكنك القيام به مع الحاسوب باستخدام هذه الوحدات/ modules. أحد الأسباب التي تجعل مكتبات بيرل أكثر مما عند بايثون هو أنه عمل حولها لوقت أطول بكثير من بايثون. ولعلي أقترح عمل نقل لموديلز بيرل إلى بايثون في موقع comp.lang.python .
كذلك؛ Parrot virtual machine هي مصممة لتقوم بتشغيل كل من لغة بيرل ٦ المعاد تصميمها تماما مثل بايثون وكذلك اللغات المفسرة الأخرى مثل Ruby و PHP و Tcl .
ما يعنيه ذلك بالنسبة لك هذا أنك ربما تكون قادرا على استخدام جميع وحدات بيرل من داخل بايثون في المستقبل، ولذا سيمنحك ذلك الأفضل في كل من أقوى مكتبة في العالم CPAN بالاشتراك مع لغة بايثون 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 **Eric S. Raymond** { ESR} **عن بايثون.**

- ***Eric S. Raymond***: مؤلف كتاب 'The Cathedral and the Bazaar' {الكاتدرائية والبازار} وهو أيضا الشخص الذي وضع مصطلح المصادر المفتوحة.

يقول: "/ Python has become his favorite programming language لقد أصبحت بايثون هي لغة البرمجة المفضلة لدي " وتعتبر هذه المقالة هي الملهم الحقيقي لي في أولى خطواتي في بايثون.

- ***Bruce Eckel*** : وهو مؤلف الكتب الشهيرة "التفكير بلغة جافا" 'Thinking in Java' و "التفكير بلغة c++" 'Thinking in C++' يقول : ليس هناك لغة قد جعلته أكثر إنتاجية من بايثون. ويقول : إنه ربما تكون بايثون هي اللغة الوحيدة التي تركز على جعل الأمور أسهل بالنسبة للمبرمج. اقرأ المقابلة الكاملة complete interview لمزيد من التفاصيل.
- ***Peter Norvig*** : هو معروف جيدا بأنه مؤلف لغة Lisp ومدير جودة البحث في جوجل (شكرا ل Guido van Rossum لإشارته إلى ذلك)
يقول: إن بايثون كانت دائما جزءا لا يتجزأ من Google، يمكنك التحقق من هذا التصريح في الواقع من خلال النظر في صفحة Google Jobs . والتي وضعت لغة بايثون في قائمة المعارف المطلوب معرفتها من قبل مهندسي البرمجيات.
- ***Bruce Perens*** : هو أحد المؤسسين ل opensource.org ومشروع userlinux.
Userlinux هدف لعمل توزيعة قياسية من لينكس مدعومة من بائعين متعددين .وقد ضرب بايثون المتنافسين مثلما فعلت بيرل وروبي لتصبح لغة البرمجة الرئيسية التي ستكون مدعومة من قبل Userlinux.

الفصـل الثانـي
تَنْصيب بايثـون

جدول المحتويات

لمستخدمي Linux/BSD- - لمستخدمي ويندوز - الخلاصة

إذا كنت تَستعمل توزيعة لينكس مثل فيدورا أو ماندريك أو {ضع اختيارَك هنا}، أو نظام BSD مثل FreeBSD، احتمال كبير أن يكون بايثون مثبتا على نظامك .
لاختِبار ما إذا كان بايثون موجودا بالفعل عندك على توزيعة لينكسَ، افتح برنامج الترمنال(مثل console أو Gnome terminal) وأدخل هذا الأمر

```
$ python -V
Python 2.3.4
```

$ هي علامة المحث/المؤشر في shell وذلك يختلف بالنسبة لك اعتمادا على إعدادات النظام لديك.
$ لذلك سوف أشير إلى المحث بهذا الرمز فقط

إذا رأيت بعض المعلومات عن إصدارة النسخة مثل ما هو معروض في الأمر أعلاه، وإلا فإنه عليك أن تثبت بايثون.

على أية حال، إذا حصلت على رسالة مثل هذه:

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

في تلك الحالة يكون معناه أن بايثون ليس مثبتا عندك، وهذا من المستبعد جدا لكنه محتمل.
في هذه الحالةِ، عِنْدَك طريقتان لتَنْصيب بايثون على نظامِك.

- تركيب binary packages باستخدام برامج إدارة الحزم التي تَجيء مع النظام، مثل yum في لينكس فيدورا, - urpmi في لينكس ماندراك، - apt-get في ديبيان لينكس , - pkg-add في نظام FreeBSD ، الخ.

مُلاحظة: سَتَحتاج اتّصالا بالإنترنت لاستِعْمال هذه الطريقة.
وبديلا عن ذلك، يُمْكِنُك أن تُحمّل حزم binaries مِن مكان آخر وبعد ذلك انسخها إلى جهازك وقم بتثبيتها.
يُمْكِن أن تركب بايثون بعمل كومبايل للكود المصدري source code وتثبيته. وسوف تمدك شبكة الإنترنت بأوامر التركيب .

لمستخدمي نظام ويندوز :

قم بزيارة موقع Python.org/download وحمّل آخر نسخة لبايثون مِن هذا الموقع (كان رقم النسخة : 2.3.4) حتى كتابة هذه الكلمات. وهي يبلغ حجمها فقط 9.4 ميغابايت في صورة مضغوطة جدا بالمقارنة مع أكثر لغات البرمجة الأخرى. و التركيب مثل أي برامج مبنية على بيئة الويندوز.

تنبيه:

عندما تَعطي خيارًا بعدم تثبيت أي مكوّنات إضافية، لا تختر Any ف يكون أحد هذه المكوّنات مفيدا لك، خصوصا IDLE .

الحقيقة المثيرة أن حوالي 70 % ممن قام بتحميل برامج بايثون مِن مستعملي ويندوز. بالطبع، هذه لا تَعطي صورة كاملة حيث أن كُل مستعملي لينكس تقريبا سَيكون عِنْدَهُم بايثون مثبتا على أنظمتِهم بشكل افتراضي .

استعمال بايثون في سطر أوامر ويندوز

إذا أردت أن تَكُون قادرا على استِعْمال بايثون مِن سطر أوامر ويندوز، فإنك تَحتاج لوَضْع الدليل PATH بشكل صحيح.

بالنسبة لويندوز 2000, 2003 ،XP، اضغط بالفأرة

Control Panel -> System -> Advanced -> Environment Variables.

اضغط على المتغيّر المسمّى PATH في قسم 'System Variables' ثم اختر EDIT وقم بإضافة: " C:\Python23 " (بدون أقواس "") في نهاية السطر المكتوب بالفعل هناك. بالطبع استعمل اسم الدليل المناسب لك.

للنسخ الأقدم مِن نظام النوافذ، يضاف السطر التالي إلى الملف C: \ Autoexec 'PATH=%PATH%;C:\Python23' (بدون أقواس "") ثم أعد تشغيل النظام.

بالنسبة لويندوز NT، يَستعمل ملف AUTOEXEC.NT.

الخلاصة

بالنسبة لنظام لينكس من المحتمل جدا أنك نصّبت بايثون على نظامِك. ما عدا ذلك، يُمْكِنُك أن تُركّبَه باستخدام برامج إدارة الحزم التي تَجيء مع توزيعتك.

بالنسبة لنظام ويندوز، يتم تثبيت بايثون بسهولة كذلك من خلال تَحميل ملف البرنامج وبالنقر مرتين عليه. ومن الآن فَصَاعِدَاً، سَنَفترض بأنّك نصّبت بايثون على نظامِك.

الفصل القادم، سَنَكْتب برنامجَنا الأول على بايثون.

الفصل الثالث
الخطوات الأولى

جدول المحتويات:

***المقدمة:**

سَنَرى الآن كَيف نشغل البرنامج التقليدي 'Hello World' في بايثون. سَيُعلّمُك هذا كَيف تَكْتبُ، تحفظ، وتشغّل برامج بايثون.
هناك اثنتان مِن طرق استخدام بايثون لتشغيل برنامجِك:-
**استخدام محث المفسر التفاعلي interactive interpreter prompt (من خلال أي برنامج كونسول)
** أو استخدام ملف مصدري. وسَنَرى الآن كَيف نَستعمل كلتا الطريقتين.

استعمال مؤشر المفسر the interpreter prompt

ابدأ interpreter على سطر الأوامر بكتابة كلمة python في الصَدَفَة {terminal / console} .
والآن اكتب: 'Hello World' متبوعا بمفتاح Enter. يَجِب أن تَرى الناتج عبارة عن: Hello World.
لمستخدمي ويندوز: ، يُمْكِنُك أن تشغل المفسر interpreter من سطر الأوامر {Dos} إذا وَضعت دليل المتغيّر PATH بشكل ملائم. وبديلا عن ذلك، يُمْكِنُك أن تَستعمل برنامج IDLE.
IDLE عبارة عن بيئةِ تطوير متكاملةِ صغيرة Integrated Development Environment.
(Click on Start -> Programs -> Python 2.3 -> IDLE (Python GUI
. مستعملو لينكس يُمْكِنُهم أن يَستعملوا IDLE أيضاً.
مُلاحظة:
هذه العلامة >>> إشارة لدُخُول محث البرنامج المفسر لبايثون prompt for entering Python statements. .
مثال 3.1. استعمال مؤشر مفسر بايثون **prompt Example 3.1. Using the python interpreter**

```
$ 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
>>>
```

لاحظ أن بايثون يَعطيك ناتج السطر فورا ! وأنت قد أدخلت فقط single Python statement. نحن نستعمل print (بشكل غير مفاجئ) لطبع أيّة قيمة value أعطيتها له.
هنا، نحن قد أعطينا له النَص "Hello World" وهذه تُطْبَع فورا على الشاشة.

كَيفية الخروج من بايثون :
للخُرُوج من مؤشر البرنامج prompt اضغط على مفتاحي Ctrl+ d إذا كنت تَستعمل IDLE أو تستخدم صَدَفَة BSD/LINUX. في حالة مؤشر سطر الأوامر بويندوز ، اضغط مفتاحي Ctrl+z متبوعة بمفتاح ENTER.

اخْتيار محرر النصوص

قَبْل أن نَنتقل لكِتابة برامج بايثون في الملفات المصدريةِ، نَحتاج محرّرا لكِتابَة الملفات المصدرية. إن اختيار محرّر أمر مهم في الحقيقة. يَجِب أن تختار المحرر كأنك تَختار السيارة التي تَشتريها. إن المحرّر الجيد سَيُساعدُك في كتابة برامج بايثون بسهولة، ويَجْعل رحلتَك مريحة أكثر ويُساعدَك لتَصِل إلى غايتِك (تَنَال هدفَك) بطريقة أسرع بكثير وأكثر أمانا.

إحدى المتطلبات الأساسية جدا هي syntax highlighting (ألوان بارزة للتراكيب) بحيث تكون كُل الأجزاء المختلفة للبرنامج ملونة colorized في بايثون حتى يتسنى لك أن تَرى برنامجَك وتَتصوّر كيفية عمله.

إذا كنت تَستعمل ويندوز، أقترح عليك استعمال IDLE. حيث إن IDLE يقوم بعمل syntax highlighting وأكثر بكثير مثل السَماح لك بتشغيل برامجِك ضمن IDLE ضمن أشياء أخرى.

مُلاحظة خاصّة: لا تستعمل Notepad - إنه اختيار سيئ لأنه لا يقوم بعمل syntax highlighting والأهم من ذلك أنه لا يَدْعم تثليم النَص indentation (أي تنظيم المسافات البادءة) وهو مهم جدا في حالتِنا كما سَنَرى لاحقا. المحرّرات الجيدة مثل IDLE (وأيضا VIM) ستساعدك آليا لعمَل ذلك.

إذا كنت تَستعمل Linux / FreeBSD، tg]d; فلديك الكثير مِن الخيارات بين المحرّرات. وإذا كنت مبرمجا خبيرا، فمن المؤكد أنك تستعمل VIM أو Emacs. ولا حاجة للقول بأنهما إثنان مِن المحرّرات الأقوى و سَنَستفيد من إستعمالهما لكِتابَة برامجِك على بايثون.

أنا شخصيا أستعمل VIM لأغلب برامجي. إذا كنت مبرمجا مبتدئا، حينئذ يُمْكِنُك أن تَستعمل KATE, وهي إحدى أدواتي المفضّلة. في حالة ما إذا كنت راغبا في قضاء وقت لتَعَلّم VIM أو Emacs، فإني أوصيك بشدة أن تَتعلّم اسْتِعْمال أي منهما حيث سَيَكُون ذلك مفيدا جدا لك في المدى البعيد.

إذا كنت ما تزال تُريد اسْتِكْشاف الخيارات الأخرى للمحرّر، انظر القائمة الشاملة لمحرّرات بايثون وحدد خيارك. يُمْكِنُك أن تَختار أيضا IDE (Integrated Development Environment) (بيئة تطوير متكاملة) لبايثون. شاهد القائمة الشاملة لـ IDE التي تدعم بايثون للمزيد من التفاصيل. عندما تَبْدأ بكِتابَة برامج كبيرة في بايثون ،فإن برامج IDEs يُمكن أن تكُون مفيد جدا في الحقيقة.

أُكرّر مرة أخرى، رجاء اختر محرّرا جيدا - فهو يَجْعل كتابة برامج بايثون وأكثر مرحَا وسهولة.

استخدام الملف المصدري

والآن دعنا نعود الى البرمجة. هناك تقليد أنه كلما كنت في سبيلك لتعلم لغة برمجة جديدة، اول برنامج تكتبه وتشغله هو برنامج 'Hello World' -- كل ما عليك فعله هو أن تقول: 'Hello World' عند تشغيله. وكما قال Simon Cozens[1]: " إنها بمثابة تعويزة تقليدية لأرباب البرمجة لمساعدتك على تعلم اللغة بشكل أفضل' .

ابدأ في اختيار المحرر، أدخل البرنامج التالي واحفظه باسم: helloworld.py

مثال: ٣.٢ استخدام الملف المصدري

Example 3.2. Using a Source File

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

(Source file: code/helloworld.py)(يشير المؤلف هنا لمكان الملف المصدري والذي يأتي مع الكتاب عند تحميلك له)

شغل هذا البرنامج عن طريق فتح الصدفة/الشل (Linux terminal or DOS prompt) وكتابة الأمر :

```
$ Python  Hello World
```

اذا كنت تستخدم IDLE، استخدم menu Edit -> Run Script او اختصار لوحة المفاتيح Ctrl+F5. والناتج output كما هو مبين أدناه.

Output

[1] واحد من كبار الهاكرز ومن أبرز رواد perl٦/parrot والمؤلف للكتاب المذهل "Beginning Perl"

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

اذا حصلت على الناتج كما هو مبين أعلاه، لك تهانينا! -- لقد نجحت في تشغيل أول برنامج لك في بايثون.وفي حال حصلت على خطأ ما، يرجى كتابة البرنامج المذكور بالضبط كما هو مبين أعلاه وتشغيل البرنامج مرة أخرى. علما أن بايثون يتميز بالحساسية لحالة الأحرف case-sensitive
مثال: كلمة print لا تساوي Print -- لاحظ الحرف الصغير p في الكلمة الأولى و الحرف الكبير P في الأخيرة.
أيضا، تأكد من عدم وجود فراغات او علامات شرطات قبل الحرف الأول في كل سطر -- وسنرى لماذا أن هذا أمر مهم في وقت لاحق.

***كيف يعمل:**

دعونا ننظر في أول سطرين من البرنامج. وتسمى هذه الكلمات : "التعليقات - comments "-- أي شيء مكتوب على يمين الرمز # هو تعليق و هو في الأساس أمر مفيد لقارئ هذا البرنامج .
بايثون لا يستخدم التعليقات باستثناء حالة خاصة من السطر الأول هنا. وهي تسمى shebang line وعندما يكون أول حرفين من الملف المصدري عبارة عن # # متبوعا باسم البرنامج فإن هذا يخبر لينكس\يونكس أن هذا البرنامج يجب ان يعمل مع هذا المفسر interpreter عند تنفيذ البرنامج.
وسوف يشرح هذا بالتفصيل في القسم التالي. علما أنه بإمكانك دائما تشغيل البرنامج على أي منصة platform من خلال تحديد المفسر interpreter مباشرة على سطر الأوامر مثل الأمر: python helloworld.py .

مهم

استخدام التعليقات أمر عقلاني 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 التي تلي Python statement -- تقوم فقط بطباعة هذا النص: 'Hello World'. إن كلمة print هي بالفعل عبارة عن operator معامل و'Hello World' يشار اليها باعتبارها سلسلة نصية string-- لا تقلق !!، سنبحث في هذه المصطلحات بالتفصيل فيما بعد.

برامج بايثون القابلة للتنفيذ Executable Python programs

وهذا لا ينطبق إلا على مستخدمي لينكس / يونكس ولكن قد يكون مستخدمو ويندوز لديهم بعض الفضول عن السطر الاول من البرنامج. أولا ، يتعين علينا اعطاء البرنامج تصريح بالتنفيذ executable permission باستخدام الأمر Chmod ثم تشغيل البرنامج المصدر.

```
$ chmod a+x helloworld.py
$ ./helloworld.py
Hello World
```

أمر chmod يستخدم هنا لتغيير حالة تصاريح الملف لجعله قابلا للتنفيذ باعطاء الاذن لجميع مستخدمي النظام.وبعدها فإننا ننفذ البرنامج مباشرة عن طريق تحديد موقع الملف المصدري. اننا نستخدم. / تشير إلى أن هذا البرنامج يقع في الدليل الحالي.
لجعل الأمور أكثر متعة ، يمكنك فقط إعادة تسمية الملف الى helloworld وتشغيله على النحو : . / Helloworld وسيظل يعمل لأن النظام يعرف ان عليه تشغيل البرنامج باستخدام المفسر المحدد في السطر الاول في الملف المصدر.
أنت الآن قادر على تشغيل البرنامج ما دمت تعرف بالضبط مسار البرنامج -- ولكن ماذا لو كنت تريد القدرة على تشغيل البرنامج من اي مكان؟ يمكنك ان تفعل ذلك من خلال تخزين البرنامج في واحدة من الادله 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 الادله الواردة في المسار PATH.

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

-اكتب الأمر

-قم بنسخ البرنامج لأحد الأدلة الناتجة من الأمر السابق

-يمكنك الآن تشغيل البرنامج من أي مكان دون الحاجة لذكر مساره

يمكننا عرض متغير المسار PATH باستخدام الأمر *echo* وتقديم الرمز *$* قبل اسم المتغير لتشير إلى shell ونحن بحاجة إلى قيمة هذا المتغير هكذا:

$ echo $PATH

ونحن نرى أن home/swaroop/bin// هو أحد الأدلة في المسار PATH حيث swaroop هو اسم المستخدم الذي أستخدمه على النظام عندي. سيكون هناك عادة دليل مماثل similar directoryلاسم المستخدم الخاص بك على جهازك. وكبديل 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 خلق أوامر لنفسك مثلها مثل الأمر cd أوغيره من الأوامر التي تستخدمها في طرفية لينكس أو الدوس.

تنبيه: W.r.t. Python يمكن تسميته برنامج او سكريبت او تطبيق وجميعها تعني نفس الشيء.

الحصول على المساعدة

إذا كنت بحاجة إلى معلومات بشكل سريع عن أي دالة أو بيان statement في بايثون ، يمكنك استخدام وظيفة المساعدة المدمجة في البرنامج . هذا مفيد جدا وخصوصا عند استخدام مؤشر المفسر interpreter prompt. فعلى سبيل المثال شغل help(str) -- هذا الأمر يعرض مساعدة عن الطبقة str class والتي تستخدم لتخزين كل نص (الجمل) تستخدمه في برنامجك. الطبقات Classes سيتم شرحها بالتفصيل في الفصل المتعلق بالبرمجة الشيئية الموجهة object-oriented programming

ملاحظة:
اضغط q للخروج من المساعدة

وبالمثل، يمكنك الحصول على معلومات عن أي شيء تقريبا في بايثون. استخدم () helpلمعرفة المزيد حول استخدام ' طريقة المساعدة نفسها!

في حال كنت بحاجة إلى الحصول على مساعدة عن معامل operator مثل print، فأنت بحاجة إلى تحديد متغير البيئة PYTHONDOCS environment variable 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 يفهم أنني اريد استحضار مساعدة حول 'print' وأنني لا منه طباعة أي شيء.

علما أنني استخدمت الموقع الأول للبرنامج و هو المستخدم في فيدورا 3 -- و قد تكون مختلفة طبقا للتوزيعة أو النسخة.

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

الفصـــل الرابــع
الأساسيـــات

قائمة المحتويات

إن طباعة 'Hello World' فقط لا يكفي، أليس كذلك ؟! هل تريد أن تفعل أكثر من ذلك -- تريد أن تأخذ بعض المدخلات، و التلاعب بها والحصول على شيء منها. يمكننا أن نحقق هذا في بايثون باستخدام الثوابت والمتغيرات constants and variables.

الثوابت الحرفية Literal Constants

مثال الثابت الحرفي هو: العدد 5 ، 1،23 ، 3 - 9.25e أو جملة مثل 'This is string ' او " It's a string!". وهذه تسمى الحرفية literal لأنها حرفية – وأنت تستخدم قيمتها الحرفية . الأمر الثاني: أنها تمثل نفسها في حد ذاتها ولا شيء آخر -- وهي ثابت constant لأن قيمتها لا يمكن أن تتغير. لذلك، فهذه كلها يشار اليها بأنها الثوابت الحرفية lliteral constants.

الأعداد:

الأعداد في بايثون أربعة أنواع:- أعداد صحيحة integers، أعداد صحيحة طويلة long integers، أعداد الفاصلة العائمة, floating point وأعداد مركّبة complex numbers.

أمثلة على الأعداد الصحيحة: 2 التي هي عدد صحيح فقط.
الأعداد الصحيحة الطويلة هي فقط أعداد صحيحة ولكنها أكبر.
أمثلة أعداد الفاصلة العائمة/, floating point (أو float اختصارا) 3.23 E-4 .,52.3,E-4 يُشير إلى 10^{-4} في هذه الحالةِ، E-4*52.3 تعني $10^{-4}*52.3$

أمثلة الأعداد المركّبة (-5+4j) و (2.3 - 4.6j)

السلاسل النصية strings

strings : هي سلسلة من الحروف. وهي أساسا مجرد مجموعة من الكلمات.
تقريبا أستطيع أن أضمن انك ستقوم باستخدام strings في كل جانب تقريبا من برامج بايثون التي تكتبها، لذلك عليك الانتباه إلى الجزء التالي. وإليك كيف تستخدم الجمل النصية في بايثون:

** استخدام علامات الاقتباس المفردة (‘)
يمكنك تحديد النص string باستخدام علامة اقتباس مثل 'Quote me on this' جميع المساحة البيضاء مثل الفراغات وال tabs تحفظ كما هي.
** استخدام علامات الاقتباس المزدوجة: double quotes (")
الجمل ضمن علامات الاقتباس المزدوجة double quotes " " تعمل بنفس الطريقة تماما كما في الجمل التي ضمن علامات الاقتباس المفردة ‘ ’ single quotes. مثال على ذلك "What's your name?"

**استخدام علامة الاقتباس الثلاثية ‘‘‘ ’’’ triple quotes
يمكنك تحديد جمل متعددة الأسطر من السلاسل النصية باستخدام علامات الاقتباس الثلاثية. ويمكنك استخدام علامة الاقتباس المفردة والمزدوجة بحرية ضمن علامة الاقتباس الثلاثية.
على سبيل المثال:

```
•
• '''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." '''
```

Escape Sequence

افترض أنك تريد أن تكون الجملة تحتوي على علامة اقتباس فردية (')، كيف لك أن تحدد هذه الجملة ؟ على سبيل المثال، the string is What's your name لا يمكنك تحديد هذه العبارة(What's your name) بعلامة اقتباس مفردة ‘ لأن بايثون سوف يرتبك من حيث أين تبدأ الجملة وأين تنتهي. لذا سيتعين عليك أن تجعل علامة الاقتباس الواحدة لا تشير لنهاية النص (لئلا يعتقد بايثون أن الجملة انتهت عند علامة الاقتباس الأولى). هذا ويمكن إنجاز ذلك بمساعدة ما يسمى *escape sequence* (**سلاسل الهروب**). اجعل علامة الاقتباس المفردة {قبل نهاية النص} هكذا \ '-- انتبه إلى الشرطة المائلة الخلفية backslash. \.

الآن، يمكنك تحديد الجملة النصية بعلامة اقتباس مفردة كما يلي: ***'What\'s your name?'***.

هناك طريقة اخرى لتحديد هذه السلسله ستكون بتحديد ***"What's your name?"*** اي استخدام اقتباس مفرد بداخل اقتباس مزدوج. وبالمثل، عليكم استخدام escape sequence من أجل استخدام علامة الاقتباس المزدوجة ذاتها في حال كانت الجملة الأصلية محاطة بعلامة اقتباس مزدوجة. كذلك، عليك ان تشير بالشرطة المائلة الخلفية ذاتها backslash الشرطة باستخدام escape sequence ||.

ماذا لو أردت أن تحدد سلسلة نصية بها سطران ؟ ويتمثل أحد الطرق باستخدام اقتباس ثلاثي ‘‘‘- ’’’ كما هو مبين أعلاه أو يمكنك استخدام escape sequence. للسطر الجديد بكتابة - \ n لتشير الى بدء سطر جديد. ومثال على هذا ***This is the first line\nThis is the second line***

. هناك فائدة أخرى لـ escape sequences هو ال \t - tab . وهناك العديد من سلاسل الهرب escape sequences ولكني أشرت فقط إلى أكثرها منفعه هنا.

شيء واحد علينا أن نلاحظه في الجملة، هو أن الشرطة المائلة الخلفية \ backslash في نهاية الجملة تشير الى أن السلسلة النصية مستمرة في السطر المقبل، ولكن بدون اضافة سطر جديد. على سبيل المثال،

المتغيرات Variables

إن استخدام الثوابت الحرفية literal constants فقط يمكن أن يصبح سريعا أمرا مملا -- ونحن بحاجة الى طريقة ما لتخزين المعلومات والتلاعب في أي منها ايضا. وهذا حيث تظهر المتغيرات في الصورة. المتغيرات بالضبط تدل على اسمها - لأنه يمكن ان تتفاوت قيمتها اي يمكنك ان تخزن اي شيء باستخدام متغير Variable. المتغيرات ليست سوى اجزاء محجوزة من ذاكرة الكمبيوتر الخاص بك حيث تخزن بعض المعلومات. بعكس الثوابت الحرفية literal constants، أنت بحاجة إلى طريقة ما للوصول الى هذه المتغيرات ، وبالتالي اعطائها أسماء.

تسمية المعرف Identifier Naming

المتغيرات هي امثلة على المعرفات . المعرفات هي أسماء تعطى لتعريف شيء ما. وهناك بعض القواعد عليك اتباعها لتسمية المعرفات :

**الحرف الأول من المعرف لا بد أن يكون من من الحروف الابجدية كبيرة أو صغيرة (upper or lowercase) او شرطة منخفضة underscores ('_').

** بقية اسم المعرف يمكن ان تتكون من الحروف الكبيرة أو الصغيرة (upper or lowercase) ، و شرطة منخفضة underscores ('_'). او ارقام (0-9).

** أسماء المعرف حساسة لحالة الأحرف case-sensitive. على سبيل المثال ، Myname و myname ليست بدرجة واحدة. نلاحظ الحرف الكبير M في الكلمة الأولى والحرف الصغير m في الأخرى ان تتغير هذه الاخيرة.

امثلة لأسماء معرفات صالحة: i, __my_name, name_23 and a1b2_c3 ،
(نلاحظ أن المعرف الأول حرف هجائي –والثاني يبدأ ب underscore – والثالث أوله حرف أبجدي ..... إلخ)
امثلة لأسماء معرفات غير صالحة : 2things, this is spaced out and my-name
(نلا حظ أن المعرف الأول يبدأ برقم – والثاني يحتوي على مسافات وفراغات - والثالث يحتوي على شرطة - dash وليس underscore)

أنواع البيانات... Data Types

المتغيرات Variables يمكنها حمل قيم لأنواع مختلفة من البيانات تسمى Data Types. الأنواع الأساسية هي: الأعداد - وسلاسل النصوص strings التي ناقشناها من قبل . وفي الفصول القادمة سنرى كيفية إنشاء الأنواع الخاصة بنا باستخدام الطبقات/ classes

الكائنات Objects ....

تذكر أن بايثون يشير إلى أي شيء مستخدم في البرنامج على أنه كائن object، وهذا هو المقصود بمعناه العام . فبدلا من أن نقول عنه " شيء - something) نقول عنه (كائن object)

**ملاحظة لمستخدمي البرمجة الكائنية الموجهة: Object Oriented Programming users

بايثون لغة برمجة قوية في مجال البرمجة الكائنية الموجهة بمعنى إن كل شيء عبارة عن object سواء الأعداد ، النصوص،وحتى الدوال functions .

Save the following سنرى الآن كيف نستخدم المتغيرات مع الثوابت الحرفية Variables literal constants احفظ المثال التالي في ملف ثم شغل البرنامج .

كيف نكتب البرامج في بايثون: How to write Python programs

من الآن فصاعدا ، هناك إجراءات قياسية موحدة لحفظ وتشغيل برامج بايثون على النحو التالي :

1.افتح محرر النصوص المفضل لديك.

2.أدخل كود البرنامج المعطى لك في المثال التالي.

3.احفظ الملف بالاسم المبين في أول كود البرنامج المذكور في التعليق اتبع ما اتفقنا عليه بأن تكون جميع البرامج المحفوظه بامتداد .Py

4.تشغيل مفسر بايثون متبوعا باسم البرنامج بالأمر التالي **python *program.py*** او استخدام IDLE لتشغيل البرامج. يمكنك ايضا استخدام طريقة الملفات التنفيذية executable ، كما هو موضح في وقت سابق.

Example 4.1. Using Variables and Literal constants

```
# 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
```

Output

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

كيفية عمل البرنامج How It Works

إليك كيف يعمل هذا البرنامج. أولا ، قمنا بإسناد ثابت حرفي (literal constant) 5 الى المتغير الأول i باستخدام المعامل operator (=). هذا السطر يسمى تصريح statement لأنها تبين وتعين شيئا ما ينبغي القيام به ، وفي هذه الحالة ، ربطنا اسم المتغير الأول i مع القيمه 5. بعد ذلك طبعنا قيمة i باستخدام التصريح printالذي يقوم فقط وبشكل طبيعي لا يثير الدهشه بطباعة قيمة المتغير i على الشاشة.

فإن أضفنا 1 الى القيمه المخزنة في المتغير الأول وتخزينه بأثر رجعي. ثم طباعته فالمتوقع أن نحصل على القيمه 6.

وبالمثل ، فاننا أسندنا literal string للمتغير s ثم طباعته.

Note for C/C++ Programmers..... ++ملاحظة لمبرمجي سي\سي

المتغيرات Variable تستخدم فقط بإسنادها إلى قيمة. وليس هناك حاجة للإعلان declaration عن نوع التصاريح أو المعرفات المستخدمة.

السطور المادية والسطور المنطقية Logical and Physical Lines...

السطر الفعلي (المادي) هو ما تراه عندما تكتب البرنامج. والسطر المنطقي هو ما يراه (يفهمه) بايثون كتصريح واحد single statement. بايثون تفترض ضمنا ان كل سطر فعلي مادي Physical يقابله سطر منطقي يتطابق معه Logical
مثال على السطر المنطقي: 'print 'Hello World -- اذا كان مكتوبا على السطر في حد ذاته (كما ترونه في المحرر)، هذا ايضا يتطابق مع سطر مادي.
ضمنيا، تشجع بايثون على استعمال تصريح واحد single statement لكل سطر مما يجعل الكود أكثر سهولة في القراءة
.
اذا كنت تريد أن تحدد أكثر من سطر منطقي واحد على سطر مادي، عليك ان تحدد هذا صراحة باستخدام الفاصلة المنقوطة (;) وهو ما يشير الى نهاية السطر المنطقي أو التصريح statement. على سبيل المثال،

```
i = 5
print i
```

ويكون بنفس الفاعلية لو كتبته هكذا:

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

وبنفس الشكل يمكن كتابته كما يلي :

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

أو حتى هكذا:

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

ومع ذلك، فانني اوصي بقوة ان تثبت على كتابة سطر منطقي واحد فقط لكل سطر مادي واحد. استخدام اكثر من سطر مادي واحد لكل سطر منطقي الا اذا كان السطر المنطقي طويل حقا. والفكرة هي تجنب الفصلة المنقوطة قدر الإمكان نظرا لأنها تؤدي الى مزيد من الوقت في قراءة الكود. في الحقيقة، انا لم تستخدمها أو حتى رأيت الفاصله المنقوطه في برنامج لبايثون.

مثال على كتابة سطر منطقي يمتد إلى عدة أسطر مادية كما يلي. هذا ويشير بوضوح الى انضمام السطر.

```
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
print i          هي بالضبط مثل
```

احيانا هناك افتراض ضمني انك لا تحتاج الى استخدام الشرطة العكسية المائلة backslash (\ ). وهذا هو الحال عندما يستخدم السطر المنطقي بين أقواس هلالية (parentheses) او مربعة [square brackets] أو مجعدة { curly }. وهذا يسمى سطر الانضمام الضمني **implicit line joining**. يمكنك أن ترى هذا يعمل عندما نكتب باستخدام القوائم في الفصول القادمة.

Indentation :

{مصطلح معناه تنظيم المسافات البادءة والفراغات وتهذيبها، وحرفيا تسمى التثليم والتشذيب والتقليم !! }

الفراغات البيضاء مهمة في بايثون. وبالفعل ،الفراغات البيضاء في بداية السطر أمر مهم. وهذا ما يسمى *Indentation*

. الفراغات البيضاء الأساسية(المسافات والشرطات) في بداية السطر المنطقي تستخدم في تحديد مستوى التثليم indentation level للسطر المنطقي، والذي بدوره يستخدم لتحديد تجمع من البيانات. Statements

وهذا يعني ان البيانات statements التي تسير جنبا الى جنب يجب ان يكون لها نفس التثليم indentation. كل مجموعة من البيانات تسمى الكتلة **block**. وسنرى أمثلة على مقدار أهمية **blocks** في فصول لاحقة.
أمر واحد عليك أن تتذكره وهو أن التثليم indentation الخاطئ يمكن أن يؤدي إلى أخطاء.على 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
```

وعند 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 أشير إليه من بايثون حيث يخبرنا بأن التركيب النحوي syntax في هذا البرنامج غير صحيحة. مثل كون البرنامج غير مكتوب على الوجه الصحيح. معنى هذا أنك لا تستطيع أبدا بصورة اعتباطية بداية كتلة block جديدة من البيانات (باستثناء الكتلة الرئيسية التي تستخدمها دائما ، بطبيعة الحال). الحالات التي يمكنك فيها استخدام الكتل الجديدة سيتم تفصيلها في فصول لاحقة مثل التحكم في تدفق البيانات..

How to indentكيف تقوم بعمل التثليم

لا تستخدم مزيجا من العلامات (tabs and spaces) للتثليم فضلا عن أنها لا تعمل على النحو الصحيح عبر مختلف المنصات. وأوصي بشدة أن تستخدم tab واحدة او اثنتين أو أربع مسافات لكل مستوى indentation .

اختر أحد من هذه الأساليب الثلاثة للتثليم. والاهم من ذلك ، ان تختار واحدة واستخدامها باستمرار. indentation ما هو إلا استخدام الاسلوب فحسب.

الخلاصة

الآن بعد أن مررنا على كثير من التفاصيل الدقيقة، يمكننا الانتقال الى مزيد من الأمور الأهم مثل التحكم في تدفق البيانات control flow statements. كن على ثقة أنك ستصبح مرتاحا لما سوف تقرؤه في هذا الفصل.

الفصل الخامس

العوامل والتعبيرات

Operators and Expressions

جدول المحتويات

** **مقدمة:**

أغلب البيانات statements وهي (logical lines) التي ستقوم بكتابتها تحتوي على تعبيرات **expressions.**

كمثال بسيط لأحد التعبيرات ٢+٣. أحد التعبيرات يمكن تقسيمها إلى عوامل وحدود operators and operands

Operators: هي وظيفة (كالجمع والقسمة والطرح وخلافه) تقوم بعمل شيء ما ويمكن تمثيلها بأحد الرموز مثل + أو أحد الكلمات المفتاحية/ keywords.

Operators: تتطلب بعض البيانات تسمى المعاملات/operands. وفي هذه الحالة تعتبر ٢, 3 عبارة عن operands

Operators :سوف نقوم بجولة مختصرة حول العوامل operators . واستخداماتها

تلميح : يمكنك حساب المقادير المعطاة في الأمثلة باستخدام المفسر interpreter مباشرة . على سبيل المثال لاختبار التعبير 2+3 استخدم مؤشر المفسر المتفاعل الخاص

Python interpreter prompt ببايثون

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

Table 5.1. Operators and their usage جدول يبين العوامل الرياضية واستخداماتها

Operator	Name	Explanation	Examples
+	Plus	Adds the two objects	3 + 5 gives 8. 'a' + 'b' gives 'ab'.
-	Minus	Either gives a negative number or gives the subtraction of one number from the other	-5.2 gives a negative number. 50 - 24 gives 26.
*	Multiply	Gives the multiplication of the two numbers or returns the string repeated that many times.	2 * 3 gives 6. 'la' * 3 gives 'lalala'.
**	Power	Returns x to the power of y	3 ** 4 gives 81 (i.e. 3 * 3 * 3 * 3)
/	Divide	Divide x by y	4/3 gives 1 (division of integers gives an integer). 4.0/3 or 4/3.0 gives 1.3333333333333333
//	Floor Division	تعيد ناتج القسمة بدون باقي	4 // 3.0 gives 1.0
%	Modulo	Returns the remainder of the division	8%3 gives 2. -25.5%2.25 gives 1.5 .
<<	Left Shift	Shifts the bits of the number to the left by the number of bits specified. (Each number is represented in memory by bits or binary digits i.e. 0 and 1)	2 << 2 gives 8. - 2 is represented by 10 in bits. Left shifting by 2 bits gives 1000 which represents the decimal 8.
>>	Right Shift	Shifts the bits of the number to the right by the number of bits specified.	11 >> 1 gives 5 - 11 is represented in bits by 1011 which when right shifted by 1 bit gives 101 which is nothing but decimal 5.
&	Bitwise AND	Bitwise AND of the numbers	5 & 3 gives 1.
|	Bit-wise OR	Bitwise OR of the numbers	5 | 3 gives 7
^	Bit-wise	5 ^ 3 gives 6	

Operator	Name	Explanation	Examples
	XOR		
~	Bit-wise invert	The bit-wise inversion of x is -(x+1)	~5 gives -6.
<	Less Than	Returns whether x is less than y. All comparison operators return 1 for true and 0 for false. This is equivalent to the special variables True and False respectively. Note the capitalization of these variables' names.	5 < 3 gives 0 (i.e. False) and 3 < 5 gives 1 (i.e. True). Comparisons can be chained arbitrarily: 3 < 5 < 7 gives True.
>	Greater Than	Returns whether x is greater than y	5 < 3 returns True. If both operands are numbers, they are first converted to a common type. Otherwise, it always returns False.
<=	Less Than or Equal To	Returns whether x is less than or equal to y	x = 3; y = 6; x <= y returns True.
>=	Greater Than or Equal To	Returns whether x is greater than or equal to y	x = 4; y = 3; x >= 3 returns True.
==	Equal To	Compares if the objects are equal	x = 2; y = 2; x == y returns True. x = 'str'; y = 'stR'; x == y returns False. x = 'str'; y = 'str'; x == y returns True.
!=	Not Equal To	Compares if the objects are not equal	x = 2; y = 3; x != y returns True.
not	Boolean NOT	If x is True, it returns False. If x is False, it returns True.	x = True; not y returns False.
and	Boolean AND	x and y returns False if x is False, else it returns evaluation of y	x = False; y = True; x and y returns False since x is False. In this case, Python will not evaluate y since it knows that the value of the expression will has to be false (since x is False). This is called short-circuit evaluation.
or	Boolean OR	If x is True, it returns True, else it returns evaluation of y	x = True; y = False; x or y returns True. Short-circuit evaluation applies here as well.

أسبقية العامل Operator Precedence

إذا كان لديك تعبير عن عملية حسابية مثل 2 + 3 * 4 هل تقوم بعملية الجمع أولا أم بعملية الضرب ؟!

علم الرياضيات في المدرسة الثانوية يخبرنا أن عملية الضرب يجب إجراؤها أولا - ذلك معناه

أن عملية الضرب multiplication لها أسبقية أعلى higher precedence من عملية الجمع operator addition.

الجدول التالي يعطينا بيان بأسبقية العامل في بايثون بدءا بالأسبقية الأدنى (أقل إلزاما) ثم الأسبقية الأعلى (أكثر إلزاما).
ذلك معناه أنه بداخل التعبيرات المعطاة فإن بايثون سيقوم بحساب قيمة العوامل الأدنى في الجدول قبل العوامل الأعلى في قائمة الجدول.

الجدول التالي (يبدو وكأنه دليل ومرجع في بايثون) أعطيتها لك على سبيل زيادة العلم من أجل الكمال، لذلك أنصحك به.
استخدم الأقواس ( ) لتجميع العوامل والحدود/المعاملات operators and operands لتحديد الأسبقية للعامل بوضوح ولتجعل برنامجك أسهل في القراءة قدر الإمكان. على سبيل المثال، 2 + (3 * 4) تحدد بشكل أكثر وضوحا من
2 + 3 * 4 . وهكذا مع كل شيء بعد ذلك. الأقواس ( ) يجب أن تستخدم بحساب وبعقلانية ولا نبغي أن تكون زائدة عن الحد. (مثل ٢+(٣+٤)).

Table 5.2. Operator Precedence

Operator	Description
lambda	Lambda Expression
or	Boolean OR
and	Boolean AND
not x	Boolean NOT
in, not in	Membership tests
is, is not	Identity tests
<, <=, >, >=, !=, ==	Comparisons
|	Bitwise OR
^	Bitwise XOR
&	Bitwise AND
<<, >>	Shifts
+, -	Addition and subtraction
*, /, %	Multiplication, Division and Remainder
+x, -x	Positive, Negative
~x	Bitwise NOT
**	Exponentiation
x.attribute	Attribute reference
x[index]	Subscription
x[index:index]	Slicing
f(arguments ...)	Function call

Operator	Description
(expressions, ...)	Binding or tuple display
[expressions, ...]	List display
{key:datum, ...}	Dictionary display
`expressions, ...`	String conversion

.العوامل التي لم نمر خلالها سيتم شرحها في الفصول اللاحقة
العوالم المتساوية في الأسبقية وضعت في نفس الصف في القائمة في الجدول المبين - أعلاه. على سبيل المثال + و
لهما نفس الأسبقية.

Order of Evaluation:
افتراضيا؛ فإن جدول أسبقية العوامل يقرر أي عامل له أسبقية قبل الآخر.لذلك إذا أردت تغيير أمر الأسبقية التي يجري تقييمها يمكنك استخدام الأقواس ( )، على سبيل المثال إذا أردت أن تكون عملية الجمع مقيمة قبل عملية الضرب في أحد التعبيرات حينئذ يمكن استخدام الأقواس مثل : (2 + 3) * 4 .

الارتباطية Associativity :
العوامل المرتبطة عادة من اليسار إلى اليمين كمثال العوامل المشتركة في الأسبقية تقيم بنمط من اليسار إلى اليمين. على سبيل المثال ٢ + 3 + 4 تقيم مثل (2 + 3) + 4.
بعض العوامل مثل العوامل المخصصة تحمل ترابطية من اليمين إلى اليسار مثل: a = b = c تعامل باعتبارها a = (b = c).

التعبيرات... Expressions
استخدام التعبيرات

Example 5.1. Using Expressions

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

length = 5
breadth = 2

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

Output : الناتج

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

```
Perimeter is 14
```

كيف يعمل :

{أسندنا القيمة 5 إلى المتغير Length و القيمة 2 إلى المتغير breadth ثم قمنا بتخزين حاصل ضربهما في المتغير الثالث كالتالي (area = length*breadth) }

- قيمة طول وعرض المستطيل تخزن أو تسند إلى المتغيرات (length -- breadth) بإعطائها نفس الاسم. ونحن نستخدمها لحساب مساحة ومحيط المستطيل بمساعدة التعبيرات. سقوم بتخزين نتيجة التعبير عن length * breadth في المتغير area ثم نقوم بطباعته باستخدام البيان print. اما في الحاله الثانية، فإننا مباشرة باستخدام قيمة التعبير 2 * (length + breadth) في البيان/التصريح print.

أيضا، لاحظ كيف يتميزبايثون بأناقة وجمال الكتابة في طباعة الناتج. ورغم اننا لم نحدد مسافة بين 'Area is' والمتغير area.
بايثون يضعها لنا كذلك من أجل الحصول على ناتج نظيف ولطيف ويصبح البرنامج أكثر قابلية للقراءة بهذه الطريقة (حيئذ لن نشعر بالقلق حيال المسافات في الناتج) وهذا مثال على كيفي أن بايثون يجعل الحياة سهلة بالنسبة للمبرمج !! .

الخلاصة:

قد رأيت كيفية إستخدام العوامل operators والحدود الحسابية/المعاملات operands والتعبيرات expressions والتي تنبني أساسا على كتل Blocks أي برنامج.
وفيما يلي سنرى كيف نقوم باستخدامها في برامجنا باستخدام التصاريح /البيانات.

الفصل السادس

Control Flow

جدول المحتويات

* **مقدمة :**

في البرامج التي رأيناها الان هناك دائما سلاسل من البيانات ؛ وبايثون بأمانة و بإخلاص ينفذ كلا منها في نفس الأمر. ماذا لو اردت تغيير طريقة انسياب العمل على سبيل المثال ، تريد من البرنامج اتخاذ بعض القرارات والقيام باشياء مختلفة تبعا للمواقف المختلفة مثل طباعة كلمة ' Good Morning 'أو ' Good Evening '
حيث يتوقف ذلك على ما هو الوقت الآن من اليوم ؟
وكما خمنت أنت فإن ذلك يتوقف على السيطرة على أداة تحكم تدفق البيانات في بايثون مثل : if –for -while

The "if" statement :

يستخدم للتحقق من الشرط ، وإذا كان الشرط true ، فإننا نشغل كتلة من البيانات (تسمى if-block) ، وإلا فإننا نشغل عملية اخرى لكتلة بيانات (تسمى else-block). والبند " else " اختياري .

Example 6.1. Using the if statement

```
#!/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
```

Output

```
$ 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 ، فاننا نأخذ التخمينات من المستخدم ومعرفة ما اذا كان هذا هو العدد الذي لدينا. نحن أسندنا المتغير " number" إلى أي عدد صحيح اننا نريد نريده أن يقول 23. ثم ، أخذنا تخمين المستخدم باستخدام الدالة raw_input ().
الدوال ما هي إلا أجزاء من البرامج يمكن اعادة استخدامها. سنقوم بالمزيد من القراءة عنها في الفصل التالي.
قمنا بإعطاء جملة نصية للدالة المدمجة " raw_input " والتي تقوم بطباعة النص إلى الشاشة وتنتظر المدخلات من المستخدم . وبمجرد أن ندخل أي شيء والضغط على enter تعيد إلينا الدالة المدخلات وهي في حالة raw_input عبارة عن نص ، بعدها يحول هذا النص إلى عدد صحيح باستخدام " int " ثم نخزنه في المتغير " guess " . في الحقيقة أن " int " عبارة عن طبقة " class " ولكن كل ما عليك أن تعرفه حقا الآن أنك تستطيع تحويل النص إلى عدد صحيح (على فرض أن الجملة تحتوي على عدد صحيح داخل النص) .
بعد ذلك قارنا بين تخمين " guess " المستخدم وبين العدد الذي اخترناه "23" . فإذا كان مساويا له يقوم البنامج بطباعة رسالة بنجاح العملية ،لاحظ أننا نستخدم مستويات للتثليم indentation لنخبر بايثون عن البيانات التي تنتمي إلى أية كتلة . ذلك يبين سبب أهمية التثليم indentation في بايثون . أتمنى أنك تثبت على قاعدة: " ضغطة tab واحدة لكل indentation level " . فهل ستفعل ذلك ؟!

نلاحظ أن البيان " if " يحتو ي على colon (:) في النهاية – نحن نشير إلى بايثون بأن كتلة block البيانات مستمرة .
بعد ذلك نقوم بالتحقق من كون التخمين أقل أو أكبرمن ذلك العدد "23". ما يجب علينا

هنا هو استخدام البند " elif" الذي يجمع بين اثنين من البيانات ذوي علاقة ب -if : elif
else داخل بيان مشترك if-elif-else .
ذلك يجعل البرنامج أسهل ويقلل من كمية indentation المطلوبة .
يجب أنت تكون بيانات elif و elseتحتوي على (:) في نهاية السطر المنطقي متبوعة كتلة مناظرة من البيانات
(مع التثليم المناسب بالطبع !)
* يمكنك كتابة البيان " if " مرة أخرى داخل كتلة if-block وهكذا تسمى البيان المتداخل لـ if .
* تذكر ان الأجزاء " elif " و " else " اختيارية . وأقل ما يمكن من كتابة البيان if هو :

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

بعدما ينتهي بايثون من تنفيذ البيان if كاملا بما فيها البنود المندرجة تحتها " elif " و " else " ، ثم يذهب منطلقا إلى البيان التالي من محتويات كتلة البيانات if . و في هذه الحالة الكتلة الرئيسة التي تنفذ من البرنامج تبدأ ثم يليها البيان 'print 'Done . بعدها يرى بايثون نهاية البرنامج ومن ثمّ ثم ينهيه بمنتهى البساطة .

ورغم ان هذا البرنامج بسيط جدا ، ولقد نبهت هناك إلى الكثير من الاشياء التي عليك أن تلاحظها حتى في هذا البرنامج البسيط. وهذه كلها أشياء جميلة وبسيطة (وبسيطة بشكل مفاجئ بالنسبة لأولئك القادمين بخلفية برمجية من لغة C/C++) ويتطلب منك ان تصبح مدركا لكل هذه البدايات ، ولكن بعد ذلك ، سوف تصبح مرتاحا معها وسوف بأنها مألوفة لديك .

ملاحظه للج / ج + + مبرمجين
ليس هناك أي تبديل في بيان بايثون. يمكنك استخدام احدى elif.. اذا.. بيان آخر الى ان تفعل الشيء نفسه (وفي بعض الحالات ، استخدام القاموس لنفعل ذلك بسرعة)

ملاحظة لمبرمجي C/C++

لا يس هناك بيان التبد يل switch statement في بايثون . لـ كن يم كن ا ستخدام البيانات (do it quickly) if..elif..else لعمل نفس الشيء (وفي بعض الحالات ، يستخدم القاموس لعمل ذلك بشكل سريع)

* **البيان " while " ..... The while statement "**

البيان " while " يسمح لك مرارا وتكرارا بتنفيذ كتلة من البيانات ما دام الشرط حقيقيا true. البيان " while " هو مثال لما يسمى التدوير" أو التحليق " looping statement . البيان " while " يمكن أن يشمل بند اختياري وهو " else "
استخدام البيان " 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'
```

Output : الناتج

```
$ 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 البرنامج :**

في هذا البرنامج ،ما زلنا نلعب لعبة التخمين guessing ، لكن الميزة هنا هوأن المستخدم يسمح له بالاحتفاظ بتخمينه إلا إذا كان تخمينه صحيحا ليست هناك حاجة الى لتنفيذ البرنامج مرارا وتكرارا عند كل تخمين - كما فعلنا في سابقا -. هذا يشرح بوضوح استخدام البيان " while " :

- قمنا بتحريك البيان " raw_input " و if " " الى داخل الحلقة while loop .
  وجعلنا المتغير "unning " إلى true قبل تشغيل الحلقة while loop .
- أولا : قمنا بجعل المتغير "unning " True"وبعدها شرعنا في تنفيذ عملية المقارنة من خلال كتلة بيانات while-block .
- بعد تنفيذ هذه الكتلة while-block ، يتم التحقق من الشرط مرة ثانية و في

هذه الحالة هو المتغير " running" .

- فإذا كان المتغير حالته "true" "، نقوم بتنفيذ كتلة البيان"while-block" ثانية , وإلا فسنواصل تنفيذ الكتلة الاختيارية "else-block"" وبعدها ننتقل إلى البيان التالي .
- يجري تنفيذ الكتلة else block عندما يصبح الشرط في الحلقة while loop خاطئا "False" – وربما تكون هذه أول مرة يتم فيها التحقق من الشرط . إذا كان هناك البند " else" للحلقة " while loop"، فهو ينفذ دائما إلا إذا كان لديك حلقات while loops والتي تتكرر باستمرار إلى الأبد بدون الخروج منها .
- "True"" و " False" تسمى Boolean types ويمكنك ان تعتبرها معادلا لقيمة 1 و 0 على التوالي. ومن المهم استعمالها حيثما يكون الشرط او التحقق مهما وليس المهم هو القيمه الفعليه مثل 1.

- " else-block " تكون بالفعل زائدة عن الحاجة عندما يمكنك وضع البيانات التابعة لها في نفس الكتلة

(كما في حالة البيان while statement) بعد أن تقوم while بنفس الأثر .

ملاحظة لمبرمجي C/C++

تذكر أنه يمكنك الحصول على البند "else" للحلقة " "while

الحلقة " for" .... The for loop :

" for" في البيان هي بيان حلقي looping آخر وهي تتكرر من خلال سلسلة متتابعة من الكائنات objects
مثل الذهاب خلال كل عنصر في سلسلة متتابعه. وسنتعرف على المزيد عن السلاسل بالتفصيل في فصول لاحقة كل ما عليك معرفته الآن هو ان التتابع هو مجرد مجموعة من العناصر المطلوبة.

Example 6.3. Using the for statement

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

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

Output

```
$ python for.py
```

```
1
2
3
4
The for loop is over
```

كيفية عمل البرنامج : * **

- في هذا البرنامج ؛ قمنا بطباعة سلسلة من الأعداد . وشغلنا السلسلة من الأعداد باستخدام الدالة الداخلية " range"
- أسندنا المدى المكون من رقمين (1،5) إلى المتغير "i"
- ما قمنا به هنا أننا أعطيناه رقمين"1،5" بينهما مدى يعيد لنا سلسلة من الأرقام بداية من العدد الأول ثم يتصاعد حتى يصل إلى العدد الثاني على سبيل المثال ، مجموعة (1،5) يعطي سلسلة مكونة من [1،2،3،4] افتراضيا ، ياخذ المدى range خطوة بقيمة 1 واذا أعطيته عددا ثالثا تكون الخطوة بمقدار ذلك العدد، على سبيل المثال: المجموعة (1،5،2) تعطي [1،3]. {حيث أن المدى محصور بين 1,5 والقفزة أو الخطوة بمقدار 2 }تذكر ان المدى يمتد حتى العدد الثاني- أي إنه لا يشمل العدد الثاني .
- بعد ذلك تتكرر الحلقة "for" خلال ذلك المدى

```
for i in range(1, 5):
```

تعادل :

```
for i in range [1, 2, 3, 4]:
```

- والتي تشبه إسناد كل عدد (أو كائن- object) إلى المتغير i واحدا منها في كل مرة ، بعدها يتم تنفيذ كتلة البيانات لكل قيمة لـ i .و في هذه الحالة نقوم فقط بطباعة القيمة في كتلة البيانات .
- تذكر أن الجزء " else" اختياري . وحينما يضاف ،فهو دائما ينفذ فور انتهاء الحلقة " for"إلا إذا حدث خروج مصدق بواسطة البيان " break" {سيتم شرحه بعد قليل}.
- تذكر ان " for" داخل الحلقة التكرارية loop تعمل مع اي سلسلة. وهنا لدينا قائمة من الأعداد يتم تشغيلها من خلال الدالة الداخلية " range"، ولكن على العموم يمكننا استخدام أي نوع من السلاسل لأي نوع من الكائنات {أو العناصر}! . وسوف نقوم بشرح لهذه الفكرة بالتفصيل في الفصول القادمة .

ملاحظة لمبرمجي C/C++/Java/C#

الحلقة " for" في بايثون تختلف اختلافا جذريا عن C/C++/. مبرمجي C# سوف يلاحظون ان هذه الحلقة في بايثون مشابهة لحلقة "" foreach في C# . مبرمجو Java سوف يلاحظون أيضا ان نفس الشبه في العبارة :

"for (int i : IntArray)" موجود في Java 1.5 ، إذا اردت ان تكتب "for (int i = 0; i < 5; i++)" في C/C++ ،ففي بايثون تكتب فقط "for i in range(0,5)" . وكما ترون ، فان كتابة الحلقة في بايثون اكثر بساطة واقل تعبيرا وعرضة للخطأ.

* **البيان " The break statement " break "**

break statement : يستخدم في الخروج من الحلقة التكرارية .كمثال؛ وقف تنفيذ الحلقة حتى ولو لم يصبح شرط الحلقة False أو أن سلسلة العناصر لم تتكرربالكامل .
من الملاحظات المهمة أنك لو قمت بالخروج من حلقة " for" أو " while" ؛ فإنه بالمثل لن يتم تنفيذ كتلة البيانات الخاص بـ " else" .

* **استخدام break statement**

Example 6.4. Using the break statement

```
#!/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'
```

Output

```
$ 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
```

*** كيفية عمل البرنامج :**

- في هذا البرنامج قمنا بأخذ المدخلات من المستخدم مرارا وتكرارا ثم طبعنا length المدخلة في كل مرة.
  وقد قمنا بتوفير شرطا خاصا لوقف البرنامج من خلال فحص ما إذا كان المدخل من المستخدم هو " quit"
  وأوقفنا عمل البرنامج عن طريق الخروج من الحلقة والوصول إلى نهاية البرنامج .
- يمكن اكتشاف طول السطر المدخل {عدد الحروف بما فيها المسافات} باستخدام الدالة الداخلية "len" .
- تذكر أن break statement يمكن استخدامها مع الحلقة " for" بشكل جيد .

قصيدة شعر " G2" لبايثون !!

استخدمت في هذا المثال مقطوعة شعرية صغيرة 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!
```

: The continue statement

يستخدم The continue statement في إبلاغ بايثون بأن يتخطى بقية ما ورد في كتلة الحلقة الحالية ويواصل تكرار الحلقة .

Example 6.5. Using the continue statement

```
#!/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...
```

Output

```
$ python continue.py
Enter something : a
Enter something : 12
Enter something : abc
Input is of sufficient length
```

```
Enter something : quit
```

*** كيفية عمل البرنامج :**

- في هذا البرنامج ؛ قبلنا المدخل من المستخدم ، ولكن نفذناه فقط عندما كان 3 أحرف على الأقل .لذا استخدمنا الدالة الداخلية " len" للحصول على طول العبارة ،فإذا كان الطول أقل من 3 أحرف ؛ نقوم بعمل skip بقية البيان الموجود في الكتلة باستخدام العبارة " continue". بخلاف بقية البيانات في الحلقة والتي يتم تنفيذها ، ويمكننا عمل أي نوع نريده من العمليات هنا.
- لاحظ أن عبارة " continue" تعمل مع الحلقة " for" بشكل جيد .

*** الخلاصة :**

قد رأينا كيفية استخدام ثلاث أدوات للتحكم في تدفق البيانات : (if – while – for) و البيانات المرتبطة
بها (break و continue).وتلك بعض من أكثر الأجزاء المستخدمة عادة في بايثون ؛ وكونك مرتاحا ومتآلفا معها أمر ضروري . وفيما يلي ؛ سنرى كيف ننشئ ونستخدم الدوال functions .

الفصل السابع
Functions الدوال

قائمة المحتويات

***مقدمة :**

الدوال : هي أجزاء من البرامج يمكن إعادة استخدامها. انها تسمح لك بإعطاء اسم ما لكتلة من البيانات ، ويمكنك استخدام هذه الكتلة في اي مكان من البرنامج ، واي عدد من المرات.وهذا يعرف بما يسمى استدعاء الدالة .ونحن بالفعل قد استخدمنا بعضا من الدوال الداخلية مثل الدالة "len" والدالة "range".

تعرف الدالة بالكلمة المفتاحية " def". متبوعة باسم المعرف للدالة ثم زوجين من الأقواس الهلالية () ، والتي ربما يرفق معها بعض أسماء المتغيرات وينتهي السطر بنقطتين (:) . يعقب ذلك كتلة من البيانات والتي بدورها تشكل جزءا من هذه الدالة . والمثال الذي يبين ذلك في غاية البساطة فعلا :

تعريف دالة :

Example 7.1. Defining a function

```
#!/usr/bin/python
# Filename: function1.py

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

sayHello() # call the function
```

Output

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

***كيف يعمل البرنامج :**

- نقوم بتحديد دالة مسماة sayHello باستخدام التركيب كما هو موضح أعلاه. هذه الدالة لا تأخذ أي بارامترات {قيم} ، وبالتالي لا توجد إعلان عن متغيرات بين القوسين. بارامترات الدالة توضع فقط للدالة حتى نتمكن من تمرير قيم مختلفة لها ونعود إليها لمقارنتها مع النتائج.

* **محددات الدالة Function Parameters**

الدالة يمكن ان تأخذ بارامترات ، والتي ليست سوى قيم تستخدم لهذه الدالة . هذه البارامترات تشبه المتغيرات غير أن قيم المتغيرات يتم تحديدها عندما نستدعي الدالة ، ولا يسند لها قيم بداخل الدالة .

البارامترات محددة داخل زوج من الأقواس() الخاصة بتعريف الدالة ، ومفصولة بنقطتين(:) . عندما نقوم باستدعاء الدالة ، نعطيها القيم بنفس الطريقة .

ملاحظة الأمر المهم أن الأسماء المعطاة في تعريف الدالة تدعى parameters ، بينما القيم التي تعطى داخل الدالة تدعى arguments .

استخدام محددات الدالة :

Example 7.2. Using Function Parameters

```
#!/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
```

Output

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

كيف يعمل البرنامج

- هنا ، عرفنا الدالة باسم printMax حيث أخذنا اثنين من القيم (بارامترات) هي (a , b). واستنتجنا العدد الأكبر باستخدام بسيط لعبارتي " if..else " وبعد ذلك طباعة العدد الأكبر.

•في اول استخدام لـ printMax ، نحن نعرض مباشرة الأعداد {٣‘٤} وهي (arguments)
وفي الاستخدام الثاني ، نقوم باستدعاء الدالة باستخدام المتغيرات {x,y}.تجعل
printMax(x, y)قيمة الوسيط x تسند إلى البارامتر a و قيمة الوسيط y تسند إلى بارامتر
b . الدالة The printMax تعمل نفس الشيء في كل الحالات.

* المتغيرات المحلية Local Variables

عندما تقوم بالإعلان عن المتغيرات داخل تعريف الدالة ، لا تكون مرتبطة بأي حال من الأحوال مع متغيرات أخرى بنفس الاسم خارج تعريف الدالة . أسماء المتغيرات تعتبر محلية local داخل الدالة ، وهذا ما يسمى نطاق المتغير. جميع المتغيرات لها نطاق داخل الكتلة ، و يتم الإعلان عنها في السلسلة النصية starting في بداية تعريف الاسم .

Example 7.3. Using Local Variables 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
```

Output

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

* كيف يعمل البرنامج :

•في هذه الدالة وهي المرة الاولى التي نستخدم قيمة اسم x .
بايثون يستخدم قيمة المحدِد parameter الذي اعلن عنه في الدالة .

•بعد ذلك أسندنا القيمة ٢ إلى x ، الاسم x يعتبر محليا local في الدالة التي لدينا ؛ لذا عندما نغير قيمة x في الدالة ،تصبح x المحددة في الكتلة الرئيسة بلا أي مساس .

•في بيان print الأخير نؤكد أن قيمة x في الكتلة الرئيسة لن تمس بالفعل .

* **استخدام البيان العمومي Using the global statement**

اذا اردت إسناد قيمة الى الاسم المحدد خارج الدالة ، حينئذ عليك أن تبلغ بايثون أن الاسم ليس محليا local ،

ولكنه عمومي global . ونحن نفعل ذلك باستخدام global statement. ومن المستحيل اسناد قيمة الى متغير محدد خارج الدالة دون استخدام global statement .

يمكنك استخدام هذه القيم لكل من المتغيرات المحددة خارج الدالة (بافتراض عدم وجود المتغير الذي يحمل نفس الاسم داخل الدالة). غير ان هذا الأمر لأ نشجع عليه وينبغى تجنبها لانه يصبح غير واضح بالنسبة الى قارئ هذا البرنامج . كما يوجد ذلك في تعريف المتغيرات .

باستخدام global statement يشيربوضوح أن المتغير معرف في الكتلة الخارجية.

Example 7.4. Using the global statement

```
#!/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
```

Output

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

* **كيف يعمل البرنامج** :

•global statement يستخدم للإعلان بأن x هو متغير عمومي global ؛ ومن هنا ، فاننا عندما نسند قيمة الى x داخل الدالة ، فإن هذا التغيير يظهر عندما نستخدم قيمة x في الكتلة الرئيسية.

•يمكنك تحديد اكثر من متغير global واحد باستخدام نفس البيان global. على سبيل المثال :

```
global x, y, z
```

* **القيم الافتراضيه للوسائط Default Argument Values**

في بعض الدوال ، قد ترغب في جعل بعض محدداتها parameters كوضع اختياري واستخدام القيم الافتراضية اذا كان المستخدم لا تريد توفير القيم لهذه المحددات . ويتم ذلك بفضل مساعدة قيم ال argument . يمكنك تحديد القيم الافتراضيه لل argument للبارامترات بإتباع اسم المحدد parameter name في تعريف الدالة بـ علامة (=) منبوعة بالقيمة الافتراضية .

علما بأن القيمه الافتراضيه ل argument ينبغي ان يكون ثابتا constant. وسوف يتم شرح ذلك بشيء أكثر
من التفصيل في فصول لاحقة. اما الآن ، فقط تذكر هذا.

Example 7.5. Using Default Argument Values.... استخدام القيم الافتراضية

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

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

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

Output

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

* **كيف يعمل البرنامج** :

•الدالة اسمها say تستخدم لطباعة جملة ما عددا من المرات كما نريد.
واذا لم نعطها أية قيمة ، فالوضع الافتراضي هو طباعة الجملة لمرة واحدة فقط .
•نحقق ذلك عن طريق تحديد قيمة ل argument من 1 الى عدد مرات المحدد parameter .
•في اول استخدام من say ، نعطي النص فقط وهي تقوم بطباعة الجملة مرة واحدة. في المرة الثانية من استعمال say ، ونحن نقوم بإعطائه كلا من النص و 5 argument على حد سواء والتي تنص على أننا نريد تكرار الجملة خمس مرات .

* مهم *

هذه البارامترات التي في نهاية parameter list يمكن أن تُعطى قيم argument افتراضية ؛ ولا يمكن أن يكون لديك باراميتر مع rgument افتراضي قبل باراميتر بدون argument افتراضي عند طلب البارامترات المعلن عنها في قائمة بارامتر الدالة .

وذلك بسبب أن قيم تسند إلى البارامترات حسب وضعيتها position . على سبيل المثال : def func(a," b=5) " صالحة

بينما " def func(a=5, b) " لا تصلح .

* **Keyword Arguments :**

اذا كان لديك بعض الدوال مع العديد من البارمترات وتريد أن تحدد بعضها فقط ، حينئذ يمكنك ان تعطي قيما لكل البارامترات عن طريق تسميتها -- وهذا ما يسمى keyword arguments - نحن نستخدام هذا الاسم (keyword) بدلا من الموضع position (الذي كنا قد استخدمناه طوال الوقت) لتحديد الـ arguments الخاصة بالدالة .

وذلك الأمر له ميزتان : - الأولى ؛ استخدام الدالة يكون أسهل حيث لسنا في حاجة الى الانشغال بأمر هذه الـ arguments .

الميزة الثانية ؛ أننا نستطيع إعطاء قيم للـ arguments التي نريدها كما نشاء ، ونمدها بـ parameters أخرى تحتوي على قيم افتراضية لـ arguments .

Example 7.6. Using Keyword Arguments

```
#!/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)
```

Output :

```
 $ 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 البرنامج** :

- هذه دالة اسمها func تحتوي على محدد parameter واحد(a) بدون قيمة افتراضية لل argument ، تليها 2 {parameter {b,c مع قيم افتراضية لـ argument {حيث b=5, c=10}.

- في الاستخدام الأول ؛ func(3, 7) ، البارامتر a يأخذ القيمة ٣ ، والبارامتر b يأخذ القيمة الافتراضية ٥ والبارامتر c يأخذ القيمة ١٠.
- في الاستخدام الثاني func(25, c=24) ، المتغير a له القيمة الموجودة في أول موضع في قوس الـ argument {وهي=٢٥} والمتغير c يحصل على القيمة ٢٤ بينما المتغير b يحصل على قيمته الافتراضية ٥
- في الاستخدام الثالث : func(c=50, a=100)؛استخدمنا keyword arguments كاملة لتحديد القيم

لاحظ أن القيمة المحددة للباراميتر c موضوع قبل a على رغم أننا قمنا بتحديد a قبل c عندما قمنا بتعريف الدالة .

:* The return statement

يستخدم التعبير **return ""** لإرجاع الدالة مثل الخروج من الدالة . ويمكننا اختياريا أن نرجع القيمة من الدالة على الوجه المطلوب.

Example 7.7. Using the literal statement

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

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

print maximum(2, 3)
```

Output

```
$ python func_return.py
3
```

* كيف يعمل البرنامج :

- الدالة maximum ترجع لنا الحد الأكبرمن البارامترات ، وهي في حالتنا الأعداد المعطاة للدالة {2, 3} وهي تستعمل بيانات بسيطة هي if..else .للعثور على قيمة العدد الأكبر وبعدها تعيد لنا تلك القيمة .

ملاحظة :عندما تكون return بدون أية قيمة تكون مساوية لـ None .

•None تعتبر نوع خاص في بايثون يمثل العدم . على سبيل المثال ؛ تستخدم للإشارة إلى متغير لا يحمل أي قيمة إذا كان يحمل قيمة مقدارها None .
•كل دالة تحتوي ضمنيا على إرجاع البيان None عندما تنتهي دون أن تكتب بيانها الخاصة ب return.
و يمكنك رؤية ذلك بطباعة someFunction() عندما تكون الدالة someFunction() لا تستخدم البيان return
كما يلي :

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

يستخدم البيان pass في بايثون ليشير إلى كتلة فارغة من البيانات .

* DocStrings

بايثون له ميزة أنيقة تدعى الوثائق النصية " ***documentation strings*** " والتي يشار إليها عادة من خلال اسمها المختصر " DocStrings"
Docstrings : هي أداة هامة يجب عليك أن تستفيد منها حيث إنها تساعد على توثيق البرنامج بشكل أفضل ، وتجعله أكثر سهولة للفهم .
وبشكل مدهش ، يمكننا حتى من الحصول على دعم من docstring ، عندما يقوم البرنمج بالعمل بالفعل !!

* استخدام docstrings :

Example 7.8. Using DocStrings

```
#!/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__
```

Output

```
$ python func_doc.py
5 is maximum
Prints the maximum of two numbers.

    The two values must be integers..
```

* **كيف يعمل البرنامج** :

- string السطر المنطقي الأول للدالة هو docstring للدالة . لاحظ أن DocStrings تنطبق أيضا على modules و classes التي سوف نقوم بشرحها في فصول خاصة .
- الاتفاقية المتبعة في docstring هي أن الجمل متعددة الأسطر في أول سطر تبدأ بحرف كبير capital وتنتهي بنقطة(.) . بعد ذلك السطر الثاني فارغ متبوعا بجملة من الشرح المفصل في السطر الثالث .
وننصحك بشدة أن تتبع هذه الاتفاقية في كل docstrings من أجل ما تبذله من الدوال المهمة .
- يمكننا الوصول الى docstring الخاصة بالدالة printMax باستخدام __doc__ (لاحظ الشرطة المنخفضة المزدوجة __ (double underscores) مسند إلى (اسم منتمي_إلى name (belonging__to الدالة .

* فقط تذكر ان بايثون يعتبر كل شيء أنه كائن "object" وهذا يشمل الدوال أيضا .
وسوف نتعلم المزيد عن الكائنات "objects" في الفصل المتعلق بالطبقات "classes " .

* اذا كنت قد استخدمت خاصية help() في بايثون ، فلابد أنك رأيت بالفعل طريقة استخدام docstrings بالفعل . * كل ما عليك هو مجرد استحضار __doc المنتمية لهذه الدالة وتعرضها لك بأسلوب مهذب وأنيق .
* يمكنك استكشاف ذلك من خلال الدالة المبينة أعلاه – فقط أضِف : help(printMax) في برنامجك . وتذكر أن تضغط مفتاح q للخروج من المساعدة .
الأدوات الآلية يمكنها استرجاع الوثائق من برنامجك بهذه الطريقة. لذا ، فإنني أوصي بقوة إن كنت تستخدم docstrings لأية دالة غيرتافهة تكتبها. الأمر pydoc الذي يأتي مع بايثون يعمل بالمثل لاستخدام help() عن طريق docstrings .

* **خلاصة**
لقد رأينا الكثير من الجوانب المتعلقة بالدوال ، ولكن الملاحظ أننا ما زلنا لم نغطي كافة جوانبها .
ورغم ذلك فقد قمنا بالفعل بتغطية معظم الأمور التي تتعلق بدوال بايثون الأساسية اليومية . وفيما يلي ؛ سوف نرى كيف نقوم بإنشاء Python modules .

□ □ □

الفصل الثامن

Modules

* **مقدمة**

قد رأيت كيف يمكنك إعادة استخدام الكود في برنامجك عن طريق تعريف الدوال مرة واحدة. ماذا لو اردت اعادة استخدام عدد من الدوال في البرامج الاخرى التي تكتبها؟ نعم ! كما قد خمنت ، الجواب هو النماذج modules. النموذج module : هو في الاساس ملف يحوي جميع الدوال والمتغيرات التي قمت بتعريفها. ولإعادة استخدام هذا النموذج في برامج اخرى ،يجب أن يكو اسم ملف الوحدة module بامتداد Py. . الموديل يمكن استيرادها من قِبل برنامج آخر للاستفادة من وظيفته. وهذه هي الطريقة التي يمكننا أن نستخدم مكتبيات بايثون القياسية بشطل صحيح. أولا ، سوف نرى كيفية استخدام المكتبات القياسيه للموديلز.

استخدام sys module :

Example 8.1. Using the sys module

```
#!/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'
```

Output

```
$ 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 البرنامج :__

- في البداية قمنا باستيراد sys module باستخدام التعبير " import" في الأساس هذا يترجم لنا بمعنى إخبار بايثون أننا نريد استخدام ذلك الموديل . الموديل sys يحتوي وظيفة مرتبطة مع مفسر بايثون والبيئة الخاصة به .

- عندما ينفذ بايثون الموديل المستورد sys ،بعدها يبحث عن الموديل sys.py في أحد الأدلة الموجودة في القائمة الخاصة بـ المتغير sys.path {سبق شرحه} . فإذا وجد الملف عندئذ يتم تشغيل البيانات لموجودة في الكتلة الرئيسة الخاصة بالموديل وبعدها يصبح الموديل معدا للاستعمال .

ملاحظة : هذه الافتتاحية تتم فقط عند أول مرة يستدعى فيها الموديل .كذلك اعلم أن 'sys' اختصار ل 'system'

المتغير "argv" في "sys" module" يشير إلى استخدام notation dotted {اسم من كلمتين موصولين بنقطة} sys.argv

- أحد مميزات هذا الأسلوب أن الاسم لا يشتبه مع أية متغير "argv" آخر مستخدم في برنامجك . وكذلك فإنه يدل بوضوح على كون ذلك الاسم جزءا من الموديل sys .

- المتغير sys.argv عبارة عن قائمة من الجمل النصية (سيتم شرح ذلك بالتفصيل في أقسام لاحقة).وبشكل أكثر تحديدا فإن sys.argv يحتوي على قائمة من arguments بسطر الأوامر. مثلا : ال arguments الممررة إلى برنامجك تستخدم سطر الأوامر .

arguments لكتابة وتشغيل برامجك ؛ ابحث عن طريقة لتحدد سطر أوامر ال IDE إذا كنت تستخدم لبرنامجك من خلال القوائم .

- وهنا ؛ عندما ننفذ برنامج بايثون المسمى using sys.py we are arguments_ ، فقد قمنا بتشغيل الموديل sys.spy مع الأمر python والأشياء الأخرى التابعة له عبارة عن arguments تم تمريرها إلى البرنامج . يقوم بايثون بتخزينها لنا في المتغير sys.argv .

- تذكر أن اسم البرنامج {السكربت} الشغال الذي يعمل دائما أول argument في قائمة sys.argv . لذلك في هذه الحالة سيكون لدينا 'using_sys.py' كـ [sys.argv[0] و 'we' كـ [sys.argv[1] و ' are' كـ [sys.argv[2] و 'arguments' كـ [sys.argv[3] . لاحظ أن بايثون يبدأ العدّ من 0 وليس من 1.

- المتغير sys.path يحتوي على قائمة من أسماء الأدلة التي يتم استيراد الموديلز منها . مع ملاحظة أن أول عبارة في sys.path فارغة – هذه العبارة الفارغة تشير إلى أن الدليل الحالي هو كذلك جزء من

sys.path ، والذي هو نفسه متغير البيئة PYTHONPATH environment variable. ذلك معناه أنه يمكنك مباشرة استيراد الموديلز الموجودة في الدليل الحالي . خلافا لذلك فينبغي عليك أن تضع الموديل في أحد هذه الأدلة الموجودة في قائمة sys.path .

***ملفات Byte-compiled .pyc**

استيراد الموديل أمر نفيس نسبيا ، لذا فغن بايثون يقوم بعمل بعض الحيل ليجعلها تعمل بشكل أسرع . أحد هذه الطرق هي إنشاء ما يسمى بملفات Byte-compiled .pyc وهي بامتداد .pyc الذي يرتبط بالبيئة الذي يحول بايثون البرامج إليها {.py} (تذكر الجزء الذي تحدثنا فيه عن كيفية عمل بايثون) هذا الملف بامتداد .pyc يستخدم عندما نستدعي الموديل في المرة الثانية من برنامج مختلف- وسيكون أكثر سرعة حيث أن الجزء من العملية المطلوب من استيراد الموديل قد تم عمله بالفعل . وبالمثل فإن هذه الملفات byte-compiled هي مستقلة عن المنصة platform-independent . وبذلك نكون قد عرفنا ما هي ملفات .pyc.

*** البيان "from..import":**

إذا أردت مباشرة استيراد المتغير "argv" إلى برنامجك (لتجنب كتابة sys. في كل مرة) عنده يمكنك استخدام عبارة "from sys import argv" . إذا اردت استيراد كل الأسماء المستخدمة في الموديل sys يمكنك استخدام عبارة "from sys import *" . وهذا يعمل مع كل الموديلز . 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 أي أسماء.

*** A module's __name__ :**

كل module **يحمل اسما ومجموعة من البيانات في الموديل يمكن استخراجها من خلال اسم ذلك ال** module . وذلك سهل المنال في حالة خاصة قد تم شرحها سابقا ؛ وذلك عندما نستورد ال module في المرة الأولى ، يتم تشغيل الكتلة الرئيسة من ال module .

ماذا لو أردت أن تشغل الكتلة فقط عندما يكون البرنمج مستخدما بها هي نفسها وليس مستوردا من module آخر ؟ هذا يمكن تحققه باستخدام __name__ مسندا إلى اسم الموديل .

Using a module's __name__ *

Example 8.2. Using a module's __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'
```

Output

```
```

```
$ python using_name.py
This program is being run by itself

$ python
>>> import using_name
I am being imported from another module
>>>
```

*كيف يعمل البرنامج :

كل موديل في بايثون يحمل اسم __name__ محدد وهذا الاسم '__main__' يتضمن ذلك الموديل يصبح قائما بذاته من خلال المستخدم ويمكننا بالمثل عمل الأحداث المناسبة .

*عمل Modules خاصة بك :

إنشاء موديلز خاصة بك أمر سهل ، وسوف تقوم بعمل ذلك على طول الخط. كل برنامج لبايثون يعتبر موديل كذلك . فقط عليك التأكد من احتوائه على امتداد .py . والمثال التالي سيوضح لك ذلك .

Example 8.3. How to create your own module

```
#!/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 موديل بسيط .وكما ترى ؛ فلا يوجد شيء خصوصي بالمقارنة بما اعتدناه في برامج بايثون . وفيما يلي سوف نرى كيف نستخدم هذا الموديل في برامج بايثون الأخرى .

تذكر أن هذا الموديل يجب أن يوضع في نفس الدليل الذي يعمل عليه البرنامج ، أو أن الموديل يجب أن يكون في أحد الأدلة الموجودة في قائمة sys.path .

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

import mymodule

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

Output

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

***كيف يعمل البرنامج :**

اسم من كلمتين بينهما نقطة}للوصول إلى عناصر الموديل .{ dotted notation لاحظ أننا نستخدم نفس ال notation ويجيد بايثون إعادة استخدم نفس الـ لإضفاء شعور"بايثوني" مميز عليها ، لذا ليس علينا أن نظل . نتعلم طرقا جديدة لصنع الأشياء

from..import*:

Here is a version utilising the from..import syntax.

```
#!/usr/bin/python
# Filename: mymodule_demo2.py

from mymodule import sayhi, version
# Alternative:
# from mymodule import *

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

الناتج لـ mymodule_demo2.py مثل الناتج من mymodule_demo.py.

***الدالة dir()** :

يمكنك استخدام الدالة الداخلية المدمجة dir() لعمل قائمة للمعرفات تحددها الموديل . هذه المعرفات هي الدوال ، والمتغيرات ، والطبقات classes المعرفة في الموديل .

عند إعطائك الموديل اسما لدالة dir() ، فهي تعيد لنا قائمة الأسماء المعرفة في الموديل . وعند عدم وجود أية argument متاحة لها ، تعيد لنا قائمة بالأسماء المعرفة في الموديل الحالي .

*** استخدام الدالة dir** :

Example 8.4. Using the dir function

```
$ 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 البرنامج :

في البداية رأينا استخدام dir مع الموديل المستورد sys. يمكننا رؤية قائمة ضخمة من العناصر التي تشتمل عليها .

بعدها قمنا باستخدام الدالة dir بدون تمرير أي بارامترات إليها – افتراضيا ؛ هي تعيد إلينا قائمة من العناصر المنتمية للموديل الحالي .

ومن أجل ملاحظة dir في هذا العمل ، قمنا بتعريف متغير جديد "a" وإسناده إلى قيمة وبعدها نقوم بفحص dir ، وسنلاحظ أن هناك قيمة مضافة إلى القائمة بنفس الاسم {a}. نقوم بحذف القيم المنتسبة للمتغير في الموديل الحالي باستخدام عبارة "del" وسوف ينعكس هذا التغير مرة ثانية في ناتج الدالة dir .

ملاحظة على del – هذه العبارة تستخدم لحذف اسم متغير بعد أن يتم عمل المتغير، وهي في هذه الحالة : "del a" ، ولا يمكنك على المدى الطويل الوصول إلى المتغير a – وكأنها لم تكن موجودة على الإطلاق من قبل

* الخلاصة :

الموديل أمر مفيد لأنها تمدك بخدمات ووظائف يمكننا إعادة استخدامها في برامجنا . والمكتبة القياسية التي تأتي مع بايثون تعتبر مثال على الموديلز . وقد رأينا كيف نستخدم هذه الموديلز وإنشاء الموديلز الخاصة بنا كذلك .

وفيما يلي سوف نتعلم بعض المفاهيم المهمة والتي تدعى "structures data".

الفصل التاسع : هياكل البيانات

Data Structures

* **مقد مة**

هياكل البيانات هي أساسا مجرد هياكل{لتنظيم البيانات والملفات} يمكنها حمل بعض البيانات معا. وبعباره اخرى

، فهي تستخدم لتخزين مجموعة من البيانات ذات الصلة ، وهناك ثلاثة أنواع من هياكل البيانات مدمجة في بايثون -- القائمة list، وقائمة tuple ، والقاموس dictionary. وسنرى كيفية استخدام كل منها ، وكيف انها تجعل الحياة ا سهل .

القائمة list:

أحد هياكل البيانات التي تحمل مجموعة من العناصر ذات الصلة ، فمثلا يمكنك ان تخزن سلسلة من: list القائمة البنود فى قائمة. هذا الأمر سهل التصور كما لو كنت تفكر في قائمة للتسوق لديك فيها عناصر تعدها للشراء ، فيما عدا أن من المحتمل أن لديك كل عنصر في سطر منفصل في قائمة التسوق ، في حين أن بايثون يضع فاصلة بين فيما بينها.

قائمة العناصر ينبغي ان تكون بين قوسيم مربعين [ ] حتى يفهم بايثون انك تريد تحديد قائمة. بمجرد ان تقوم بانشاء قائمة ، يمكنك اضافة او ازالة او البحث عن البنود الواردة في القائمة. وحينئذ ، يمكننا ان نضيف أونحذف البنود ، ونحن نقول ان القائمة هي نوع بيانات قابلة للتغيير {المتغيرة أو المتقلبة } اي أن هذا النوع يمكن تغييره.

Objects and Classes {مقدمة سريعه للكائنات والطبقات {الفصائل
وتسند i بوجه عام ، فإنك عندما تستخدم المتغير Objects , Classes رغم أنني قد كنت أخرت وحتى الآن مناقشة في الواقع يمكنك . int {أو الفئة}class تابع للفصيلة "i" Object إليه قيمة ما ولنقل مثلا أنها 5 ، فإنك أنشأت كائن أن تشتمل على أساليب مثل الدوال المحددة للاستخدام class لتفهم القوائم بشكل افضل. ويمكن لل help(int) ان ترى فقط . يمكنك استعمال وظيفة هذه الأجزاء فقط عندما يكون لديك كائن لهذه ال "class" بوجه خاص مع هذه الفئة التي تسمح لك بإضافة عنصر الى نهاية القائمة. ""list على سبيل المثال ؛ توفر بايثون طريقة لإرفاق الفئة class "الى قائمة string سنضيف سلسلة نصية mylist.append('an item') : {على سبيل المثال {هذه الصيغة mylist" لاحظ dotted notation {الاسم المنقط } للوصول الى أساليب ال objects.

وهي ليست سوى متغيرات احددة لاستخدامها فيما يخص تلك **fields** يمكن ان يكون لها ايضا حقول class الفئة من تلك الفئة. الحقول أيضا object الفئة فقط. يمكنك استخدام هذه المتغيرات / الأسماء فحسب عندما يكون لديك mylist.field ، على سبيل المثال ، dotted notation يمكن الوصول إليهامن خلال عليها الاسم المنقط .

استخدام القوائم* :

Example 9.1. Using lists

```
#!/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
```

Output

```
$ 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 البرنامج

نقوم بتخزين السلاسل النصية ، shoplist عبارة عن قائمة تسوق لشخص ذاهب الى السوق. في shoplist المتغير لأسماء العناصر التي سيشتريها ، لكن تذكر أنه يمكنك اضافة اي نوع من الكائنات على القائمة بما في ذلك strigs الاعداد وحتى القوائم الأخرى.

للتكرار خلال قائمة البنود. الآن ، لا بد انك ادركت ايضا ان القائمة هي "" for..in ولدينا ايضا استخدم الحلقة . في قسم لاحق sequence وسوف نتناقش بخصوص الممتسلسلة . sequence ممتسلسلة

" لمنع الطباعة التلقاءية لفاصل الأسطر بعد كل عبارة print " " لاحظ أننا نستخدم فاصلة "," في نهاية عبارة .هذه قد تعتبرطريقة قبيحة لفعل ذلك ، ولكنها بسيطة وتجعلنا ننجز المهمة " print

كما سبق ان ناقشناها من ، list object إلى append بعدها ، قمنا باضافة بند الى قائمة باستخدام طريقة الإرفاق قبل. ثم ، نتحقق من ان البند قد تم بالفعل اضافته الى القائمة عن طريق طبع محتويات القائمة وذلك ببساطة بتمرير .التي تقوم بطبعها لنا بطريقه أنيقة مرتبة print القائمة عن طريق عبارة

للقائمة. نفهم من ذلك ان هذا الاسلوب يؤثر على القائمة نفسها ولا يعيد sort ثم ، نقوم بترتيب القائمة باستخدام طريقة وهذا ما نعنيه بالقول ان القوائم قابلة للتغيير . strigs القائمة المعدلة -- وهذا يختلف عن طريقة عمل السلاسل النصية

ثابتة strigs ، وأن السلاسل النصية.

هنا . "del" ثم ، عندما ننتهي من شراء بند في السوق ، ونريد إزالته من القائمة. نحقق ذلك عن طريق استخدام البيان بحذفه لنا من القائمة. نحدد ما نريد ازالته وهو البند "del" ، نشير الى البند الذي في القائمة و نريد ازالته يقوم البيان). تذكر أن بايثون يبدأ العد من 0) (del shoplist[0]الأول من القائمة ، وبالتالي نحن نستخدم
لاستكمال التفاصيل help(list) انظر ، list object اذا كنت تريد ان تعرف كل الاساليب التى حددتها عن طريق.

***Tuple** :

Tuples مثل القوائم الا انها ثابتة مثل الجمل النصية stringsاي لا يمكنك تعديل tuples. tuples تعرف عن طريق تحديد بنود منفصلة ، بينها فصلات ,, داخل زوج من الأقواس ( ). Tuples عادة ما تستخدم في الحالات التي يكون فيها البيان او الدالة يحددها المستخدم - يمكن أن نفترض بأمان أن مجموعة من القيم أي tuple من القيم المستخدمة لا تتغير.

استخدام tuples

Example 9.2. Using Tuples

```
#!/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]
```

Output

```
$ 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 البرنامج**

المتغير zoo يشير الى tuple من البنود. ونحن نرى ان الدالة len يمكن استخدامها للحصول على طول tuple وهذا يدل أيضا على أن tuple هي ممتسلسلة sequence كذلك. الآن ننقل هذه الحيوانات إلى new_ zoo حيث أن الzoo القديمة تصبح مغلقة . لذا تحتوي new_zoo tuple على بعض الحيوانات الموجودة بالفعل جنبا إلى جنب مع الحيوانات التي جلبت من ال zoo القديمة . ,وبالنظر إلى واقع الأمر نلاحظ أن ال tuple داخل tuple

لا تفقد هويتها ،

يمكننا الوصول إلى العناصرفي ال tuple عن طريق تحديد موضع العناصر داخل زوج من الأقواس المربعة [ ] تشبه ما فعلناه في القوائم lists . وهذا ما يسمى عامل الفهرسة operator *indexing*. نستطيع الوصول إلى البند الثالث في new_zoo بتحديد [new_zoo [2 ، ويمكن الوصول إلى البند الثالث في new_zoo tuple بتحديد [new_zoo [2] [2]. هذا واضح جدا ، بمجرد ان تفهم الأسلوب.

Tuple with 0 or 1 items: ال Tuple الفارغة تنشأ عن طريق زوج من الأقواس( ) مثل myempty = ().
ومع ذلك ، tuple بها بند واحد ليس بهذه البساطة. عليك ان تحددها مستخدما فاصلة , بعد البند الاول (والوحيد) البند ؛ لذلك فإن بايثون يمكن ان يفرق بين tuple وبين زوج من بين الأقواس المحيطة لل object داخل التعبير . مثلا عليك ان تحدد (2 ,) = singleton اذا كنت تعني أنك تريد من tuple أن يتضمن البند 2.

ملاحظة لمبرمجي بيرل

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

***Tuples and the print statement :**

واحدة من أكثر الاستعمالات الشائعة هي tuples مع البيان print. وإليك هذا المثال :

Example 9.3. Output using tuples

```
#!/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
```

Output

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

***كيف يعمل البرنامج**

- البيان print يمكن ان يأخذ سلسلة نصية باستخدام مواصفات معينة يتبعها الرمز ٪ يليها tuple من البنود المطابقه للمواصفات. المواصفات تستخدم في صياغه النتائج بطريقة معينة. المواصفات يمكن ان

تكون على غرار s% للسلاسل النصية strings و d% للأعداد الصحيحة. tuple يجب ان تحتوي على بنود مقابلة لهذه المواصفات في نفس النظام .

- لاحظ أن اول استعمال حيث نستخدم s% أولا وهذا مطابقا لاسم المتغير الذي هو البند الاول في tuple ، والوصف الثاني هو d% المقابل لل age الذي هو البند الثاني في tuple.
- ما يعمله بايثون هنا هو أنه يحول كل بند في tuple الى سلسلة نصية وبدائل لقيمة هذه السلسلة داخل مكان المواصفات. لذا s% هو استعاضة عن قيمة المتغير name وهلم جرا.
- هذا الاستخدام للبيان print يجعل من السهل للغاية كتابة الناتج ويتجنب الكثير من التلاعب بال string 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 اعطاء المواصفات الصحيحه ؛يث أن هذا يضيف مستوى واحد من التأكد من صحة برنامجك .
- في البيان الثاني 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 عنوان 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 انه يمكنك استخدام ojects ثابتة فقط (مثل السلاسل النصية) لمفاتيح القاموس ولكن يمكنك استخدام ojects ثابتة أو قابلة للتغيير لقيم القاموس يمكننا أن نترجم ذلك بقولنا أنه ينبغي ان لا تستخدم سوى اشياء بسيطة للمفاتيح.

زوج من المفاتيح والقيم المذكوره في القاموس باستخدام العبارة d = {key1 : value1, key2 : value2} لاحظ أن أزواج المفتاح/القيمة منفصلين عن طريق النقطتين":" والأزواج أنفسهم منفصلان عن طريق فاصلة , و كل هذا داخل في زوج من الاقواس المجعدة { } .

تذكر ان أزواج key/value في القاموس ليست لها أي طريقة ترتيب. اذا اردت ترتيبا معينا ، سيتعين عليك ترتيبها بنفسك قبل استعمالها .

القواميس تستخدم بشكل عام أمثلة/كائنات instances/objects من الطبقة " dict " .

***استخدام القواميس**

Example 9.4. Using dictionaries

```
#!/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']
```

Output

```
$ 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**

قمنا بصنع القاموس ab باستخدام الترقيم الذي سبق مناقشته. ثم شغلنا أزواج key/value من خلال تحديد المفتاح باستخدام عامل الفهرسة indexing operator كما نوقش في الكلام عن lists و tuples. نلاحظ ان التركيب بسيط جدا للقوام يس كذلك .

ويمكننا ان نضيف أزواج جديدة من key/value ببساطة عن طريق استخدام indexing operator للوصول الى مفتاح وا سناد قي مة إل يه ، ك ما فعل نا ل Guido في الحالة المذكورة اعلاه .

يمكننا حذف أزواج المفتاح/القيمة باستخدام صديقنا القديم البيان " del". نحن ببساطة نحديد القاموس indexing operator لإزالة المفتاح وتمرير ذلك إلى البيان "del". ليست هناك حاجة لمعرفة القيمه المقابلة للمفتاح في هذه

بعد ذلك نصل إلى كل زوج من key/value في القاموس باستخدام items method من القاموس التي تعيد قائمة من ال tuples حيث كل tuple يحتوي زوجا من البنود – والمفتاح متبوعا بقيمة. نسحب هذا الزوج ونسنده إلى اسم المتغيرات والعنوان المقابل لكل زوج باستخدام الحلقة for..in ، ثم نطبع هذه القيم في كتلة for-block . يمكننا معرفة ما اذا كان زوج key/value موجود باستخدام المشغل in او حتى طريقة has_key من ال class "dict"" تستطيع ان ترى الوثائق للاطلاع على القائمة الكاملة للطرق من ال "dict" باستخدام help(dict "class")

:*Keyword Arguments and Dictionaries

على صعيد آخر نلاحظ ، ان كنت قد استخدمت keyword arguments في الدوال الخاصة بك ، ولقد سبق ان استخدمت قواميس! فقط فكر في ذلك – زوج key/value محدد من قِبلِك في قائمة بارامترات تعريف الدالة ، وعند تشغيل المتغيرات بداخل الدالة ، وهو مجرد مفتاح الوصول إلى القاموس (وهو ما يسمى *symbol table* في مصطلح تصميم المترجم) .

Sequences: *المتسلسلات

tuples و lists و strings هي أمثلة على الممتسلسلةات Sequences ، ولكن ما هي الممتسلسلة ، وماذا فيها من الخصوصية ؟ اثنان من السمات الرئيسية للممتسلسلة هي عملية الفهرسة التي تتيح لنا ان جلب بند بعينه في الممتسلسلة مباشرة ، وعملية التقطيع الذي يتيح لنا ان تستعيد شريحة من الممتسلسلة أي جزءا من الممتسلسلة.

استخدام المتسلسلات*

Example 9.5. Using Sequences

```
#!/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[:]
```

Output

```
$ 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**

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

يمكن للفهرس ايضا ان يكون عددا سلبيا ، في هذه الحاله ، يحسب من نهاية الممتسلسلةو. لذا ، shoplist [-1] يشير الى البند الأخير في الممتسلسلةة و shoplist [-2] يجلب ثاني آخر بند في الممتسلسلة.

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

الرقم الأول (قبل النقطتين) في عملية التقطيع يشير الى الموضع الذي تبدأ منه الشريحة ، والعدد الثاني (بعد النقطتين) يشير فيها للموضع الذي تتوقف عنده الشريحة. إذا كان اول عدد غير محدد ، فإن بايثون ستبدأ من بداية المتسلسلة. واذا كان الرقم الثاني متروكا فإن بايثون 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 ايضا تقطيع مع المواضع السالبة. وتستخدم الارقام 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 موصفات هذه الشريحة باستخدام مفسر بايثون التفاعلي .أي المحث الفوري بحيث يمكنك ان ترى النتائج فورا. والشيء العظيم في المتسلسلات هو أنك يمكنك تشغيل tuples ، و lists و strings ، الجميع بنفس الطريقة!

References

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

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

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
```

Output

```
$ 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 تحاج الى ان نتذكره انك اذا اردت ان تجعل نسخة من القائمة او من تلك الانواع من المتسلسلات أو الكائنات المعقدة (ليست كائنات بسيطة مثل الأعداد الصحيحة) ، فإن عليك أن لعمل نسخة. اذا قمت فقط بمجرد اسناد اسم المتغير الى اسم آخر ، كلاهما slicing operation تستخدم عملية التقطيع يشير الى الكائن ذاته ، فهذا يمكن ان يؤدي الى جميع انواع المتاعب اذا لم تكن حذرا.

: ملاحظة لمبرمجي بيرل

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

***المزيد عن السلاسل النصية strings**

لقد ناقشنا بالفعل السلاسل النصية بالتفصيل في وقت سابقا . ما المزيد الذي يمكن معرفته عنها؟
ولديها الاساليب لفعل كل شيء من أول فحص جزء objects حسنا ، هل تعرف ان السلاسل النصية تعتبر هي ايضا !من النص حتى تعرية المساحات

بعض من الأساليب المفيدة لهذه class (str). من ال objects التي تستخدمها في البرنامج هي جميع ال sjrings ال الفئة تتجلى في المثال التالي. وللحصول على قائمة كاملة من هذه الاساليب ، انظر `help(str)`.

Example 9.7. String Methods

```
#!/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)
```

Output

```
$ 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*

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

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

الخلاصة

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

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

الفصل العاشر

حل مشكلة ـ كتابه سكربت في بايثون

لقد استكشفت أجزاء مختلفة من لغة بايثون والآن سوف نلقي نظرة على الطريقة التي تناسب جميع هذه الاجزاء معا ، عن طريق تصميم وكتابة البرنامج الذي لا شيء مفيد.

***المشكلة**

المشكلة هى أذ ني أر يد برنام جا يقوم بعـ مل نسـ خة احتياط ية من جمـ يع الملفات المهـ مة لدي . ورغم أن هذا يشكل مشكلة بسيطة ، ليست هناك معلومات كافية بالنسبة لنا لنبدأ عملية الحل. القليل من التحليل هو المطلوب. على سبيل المثال ، كيف يمكننا أن نحدد الملفات التي سيتم نسخها احتياطيا ؟ أين ستوضع النسخة الاحتياط يه المخز نه ؟ كـ يف يـ تم تخزين ها في النسـ خة الاحتياط يه؟

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. اذا قمت بعمل التصميم ، لعلك لا تواجه نفس النوع من المشا كل - - كل شـ خص له طريق ته الخا صة لتسـ يير الأمور ، وهذا أ مر ط يب .

1 . الملفات والأدلة الـ تي نعـ مل لـ ها نسـ خة احتياط ية محددة فى قائـ مة .

2 . النسخة الاحتياطية يجب أن تكون مخزنة في الدليل الرئيس للنسخ الاحتياطي .

3 . الملفات المنسوخة ينبغي أن تكون في ملف مضغوط .

4 . اسم الأرشيف المضغوط يتضمن التاريخ والوقت الحالي .

5. نحن نستخدم الأمر القياسي zip المتاح بشكل افتراضي في أي توزيعة قياسية من لينكس / يونكس. ويمكن لمستخدمي ويندوز استخدام برنامج Info-Zip program – علما انه يمكنك استخدام اي أمر لبرنامج أرشفة تريده طالما انه يملك سطر الاوامر حتى يتسنى لنا تمرير قيم arguments إليه من السكربت الخاص بنا.

الحل:*

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

الإصدار الأول First Version

Example 10.1. Backup Script - The First Version

```
#!/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'
```

Output

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

- نحن الآن في مرحلة الاختبار ؛ حيث إننا نختبر برنامجنا ،هل يعمل بشكل سليم. فإذا لم يتصرف كما هو متوقع ، فسيكون علينا الانتقال الى مرحلة تصحيح برنامجنا ؛ اي ازالة الـ *bugs* (الاخطاء) من البرنامج.

* **كيف يعمل البرنامج**

ستلاحظ كيف قمنا بتحويل ما لدينا من تصميم الى الكود خطوة فخطوه. ونحن نستفيد من الموديلز os و time ولذا قمنا باستيرادها. ثم ، نحدد الملفات والأدله التي سيتم نسخها احتياطيا في قائمة ""source. الدليل " target " يعني مكان تخزين جميع الملفات الاحتياطيه ، وهذا هو المحدد في المتغير "target_dir اسم الأرشيف المضغوط هو سنقوم بإنشائه هو التاريخ الحالي والوقت الذي يجلب لنا باستخدام الدالة time.strftime () . وسوف يكون ايضا. بامتداد .zip وسيخزن في الدليل target_dir

الدالة time.strftime ()تأخذ مواصفات مثل التي استخدمناها في البرنامج المذكور اعلاه. 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 بايثون في التوزيعة الخاصة بك. لاحظ ان هذا هو مماثل (ولكن ليس على النحو نفسه) للمواصفات المستخدمة في البيان print (باستخدام ٪ تليها tuple)

قمنا بعمل اسم الدليل المضغوط target باستخدام المشغل الإضافي الذي يشبك الجمل اي يربط بين اثنين معا ويعيدها إلينا واحدة جديدة. ثم ، ننشئ سلسلة نصية : zip_command ، والتي تتضمن الأمر سنقوم بتنفيذه. يمكنك معرفة ما اذا كان هذا الامر يعمل عن طريق تشغيله على الشل (طرفية لينكس طرفية أو مؤشر دوس)

الأمر zip الذي نستخدمه يحتوي بعض الخيارات والبارامترات – الخيار q يستخدم للإشارة إلى ان الأمر zip ينبغي أن يعمل بهدوء **q**uietly – الخيار r يحدد ان الأمر zip ان تعمل **r**ecursively للأدلة {من أعلى لأسفل} اي ينبغي ان تشمل الادله الفرعية والملفات داخل الادله الفرعية كذلك. وقد تم الجمع بين خيارين لا ثالث لهما والمحدد في اقصر الطريق وهما qr – هذه الخيارات متبوعة باسم الأرشيف المضغوط المراد إنشاؤه متبوعا بقائمة الملفات والادله التي سنقوم بنسخها احتياطيا. نحن نحول قائمة source داخل الجملة باستخدام طريقه join لضم الجمل والتي شاهدنا بالفعل كيفية استخدامها.

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

واعتمادا على نتيجة الأمر ، ونقوم بطباعة رسالة مناسبة بأن النسخة الاحتياطيه فشلت أو نجحت ، وهذا هو كل ما في الموضوع ، لقد قمنا بانشاء سكربت لعمل نسخة احتياطيه من الملفات المهمة!

ملاحظة لمستخدمي ويندوز :

يمكنك ان تحدد القائمة source الدليل **target** لأسم أي ملف أو دليل ، ولكن يجب ان تكون متانياً قليلا في ويندوز. والمشكلة هي ان ويندوز يستخدم (\)**backslash** كدليل منفصل ،ولكن بايثون يستخدم **backslashs** (\) لتمثيل سلاسل الهروب! **escape sequences**

لذلك ، عليك ان تمثل الشرطة نفسها باستخدام ! **escape sequence** او عليك أن تستخدم **raw strings**. على سبيل المثال ، استخدم **'C:\\Documents'** أو **r'C:\Documents'** ولكن لا تستخدم **'C:\Documents'** - انت تستخدم سلسلة هروب **escape sequence مجهولة : \D** !

الآن قمنا بعمل سكربت للنسخ الاحتياطي ، يمكننا استخدامه حينما نريد ان نأخذ نسخة احتياطية للملفات. مستخدمي لينكس / يونكس ينصحون باستخدام طريقة الملف التنفيذي على النحو الذي سبق مناقشته حتى يتمكنوا من تشغيل برنامج النسخ الاحتياطي في اي وقت وفي اي مكان. وهذا ما يسمى مرحلة التشغيل أو مرحلة نشر البرمجيات. يعمل البرنامج اعلاه بشكل صحيح ، ولكن (عادة) البرامج الاولى لا تعمل تماما كما كنت تتوقع. على سبيل المثال ، قد تكون هناك مشاكل ،واذا كنت لم تصمم البرنامج بشكل صحيح ، او اذا كنت قد اخطأت في كتابه الكود ، الخ وبشكل منا سب ، سيتعين عل يك العودة الى مرحلة التصـ ميم او ستضطر لتصـ حيح برنام جك .

*الإ صدارة الثان ية :
النسخه الاولى من السكريبت يعمل لدينا. ومع ذلك ، يمكننا ان إدخال بعض التحسينات عليه حتى يمكنه ان يعمل على نـ حو أفضـ ل على ا ساس يو مى . وهذا ما يسـ مى مرحلة صيانة البرمجيات .

أحد هذه التحسينات التي شعرت بفائدتها هي ميكانيكية أفضل لتسمية الملف -- باستخدام *time* كاسم للملف بداخل الدليل مع تاريخ اليوم current *date* كدليل ضمن دليل النسخة الاحتياطيه. أحد الميزات هي ان نسختك الاحتياطية يتم تخزينها بطريقة هرمية ، ولذا فمن الأسهل إدارتها. وهناك ميزة اخرى وهي ان طول أسماء الملفات أقصر بكثير بهذه الطريقة. ولكن هناك ميزة اخرى هي ان الادله المنفصلة ستساعدك ان تعرف بسهولة اذا ما كنت قد قمت بعمل نسخة احتياطية عن كل يوم منذ إنشاء الدليل فقط اذا كنت قد اتخذت نسخة احتياطية لذلك اليوم.

Example 10.2. Backup Script - The Second Version

```
#!/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'
```

Output

```
$ 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 البرنامج**

الكثير من هذا البرنامج ما زال هو نفسه.و التغيرات هي ان نتحقق اذا كان هناك دليل باسم اليوم الحالي داخل الدليل الرئيس للنسخة الاحتياطيه باستخدام الدالة os.exists . فإذا كان غير موجود ، فنحن نصنعه مستخدمين الدالة os.mkdir .

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

***الإصدارة الثالثة :**

النسخة الثانية تعمل جيدا عندما كنت قمت بعمل الكثير من النسخ الاحتياطية ، ولكن عندما تكون هناك الكثير من النسخ الاحتياطيه ، وجدت صعوبة في التفريق بين غرض كل نسخة احتياطية ،وكانت من أجل ماذا ! على سبيل المثال ، فاني قد جعلت بعض التغييرات الرئيسة للبرنامج أو المثال ، ثم أردت ان اعرب عن أضم هذه التغييرات مع اسم الأرشيف المضغوط . وهذا يمكن تحقيقه بسهولة عن طريق ارفاق التعليق من المستخدم على اسم الأرشيف المضغوط.

Example 10.3. Backup Script - The Third Version (does not work!)

```
#!/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'
```

Output

```
$ 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!. بايثون يقول ان ثمة خطأ لغوي مما يعني ان السكربت لا يرضي التنظيم الذي يتوقع بايثون ان يراه. عندما نلاحظ الخطأ التي قدمه بايثون ، كذلك يخبرنا عن المكان الذي اكتشف الخطأ عنده. حتى نبدأ يضبط برنامجنا من ذلك السطر .

وبالملاحظه الدقيقة، نرى ان السطر المنطقي الوحيد قد انقسم الى سطرين ماديين ، ولكن ليس لدينا وصف محدد بان هاذين الحطين الماديين معا. اساسا ، فقد وجد بايثون أن المشغل (+) بدون حدود حسابية operand في هذا السطر ، وبالتالي من المنطقي أنه لا يعرف كيف يواصل العمل . تذكر اننا يمكن أن نحدد أن السطر المنطقي لا تزال متواصلا في السطر المادي القادم باستخدام الشرطة المائلة الخلفية backslash \ في نهاية السطر المادي. لذلك ، نقوم بعمل هذا التصحيح لبرنامجنا . وهذا ما يسمى إصلاح الخطأ **bug fixing**

***الإصدارة الرابعة**

Example 10.4. Backup Script - The Fourth Version

```
#!/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'
```

Output

```
$ 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
```

*كـ يف يع مل البرنا مج :

هذا البرنامج يعمل الآن! دعونا نمضي من خلال التحسينات الفعلية التي قمنا بعملها في النسخه 3. نستوعب تعليقات المستخدم باستخدام دالة raw_input ثم معرفة ما اذا كان المستخدم بالفعل قد أدخل به شيئا يستوضح طول المدخلات باستخدام الدالة len. اذا قام المستخدم بالضغط على مفتاح enter لسبب ما (لعلها كانت مجرد روتين احتياطي ،أو لم تـ تم أ ية تغييرات) ، بعد ها نشرع في ع مل ما قمـ نا به من ق بل .

وعلى أية حال ؛ اذا كان هناك تعليق ، فهذا التعليق يلحق باسم الأرشيف المضغوط فقط قبل الامتداد .zip ونلاحظ ان استبدال المساحات في التعليق مع شرط سفلية __ underscores فذلك لان ادارة مثل اسماء الملفات هذه أ سهل بكث ير .

*مز يد من التحسـ ينات

الإصدارة الرابعة هي سكربت يعمل بصورة مرضية لمعظم المستخدمين ، ولكن هناك دائما مجال للتحسين. على سبيل المثال ، يمكنك إضافة مستوى من الإسهاب للبرنامج ، حيث يمكنك تحديد الخيار –v لجعل برنامجك تصبح اكـ ثر حوارا .

من التحسينات الأخرى أن تسمح للملفات والأدلة الاضافية للمرور إلى السكربت على سطر الاوامر. وسنتوصل الى ذلك من قائمة sys.argvoghg ونستطيع أن نضيفها إلى قائمة source التي لدينا باستخدام طريقة التوسيع extend method الـ تي توفر ها الطبـ قة . list

من الأمور المهذبة في البرنامج أني سمحت باستخدام الأمر tar بدلا من الأمر zip . وأحد مزايا ذلك الأمر أنه عن استخدامك tarجنبا إلى جنب مع gzip تصبح النسخ الاحتياطي يصبح أكثر سرعة والنسخة الاحتياطية تكون أقل حجما . وإذا أردت استخدام هذا الأرشيف في ويندوز فإن WinZip يتعامل مع ملفات .tar.gz كذلك بسهولة. الأمر tar متاح بشكل افتراضي في معظم أنظمة لينكس/يونيكس . ويمكن لمستخدمي ويندوز تحميله من الإنترنت ثم تثبيته كذلك.

الآن سيكون الأمر بالشكل التالي :

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

والخيارات موضـ حة أدناه .

c- يشـ ير الى creation إنشاء أرشـ يف .

v- يشـ ير إلى verbose أي أن الأ مر يـ جب ان تكون اكـ ثر إيضا حا وثرثرة talkative .

z- يشـ ير إلى أن فلـ تر gzip الذي ينبـ غي ا ستخدامه .

f- يشير الى forceأي القوة في انشاء الأرشيف اي انه ينبغي أن يحل محل ملف آخر إذا كان يحمل نفس الاسم بالفـ عل .

x- يشير الى الملف الذي يتضمن قائمة اسماء الملفات التي يجب استبعادها excluded من النسخة الاحتياطيه. على سبيل المثال ، يمكنك تحديد ~* في هذا الملف لعدم ادراج اي اسماء الملفات المنتهيه ب ~ في النسخة الاحتياطية

*مـ هم *

أكـ ثر الطرق المفضلة لإنشاء مـ ثل هذا النوع من الأرشـ يف سيكون با ستخدام الموديلز zipfile او tarfile على

التوالي. اذ ها تشكل جزءا من مكتبة بايثون المعيارية والمتاح لك استخدامها بالفعل. باستخدام هذه المكتبات ايضا تتجنب استخدام os.system والتي لا ينصح باستخدامهاعلى وجه العموم، لانها من السهل أن تكلفك أخطاء باهظة با ستخدامها.
ومع ذلك ، فقد كنت أستخدم طريقة os.system لعمل نسخ احتياطية لاغراض تعليمية بحتة ، لذا يعتبر ذلك مثال بسيط بشكل كاف ليكون مفهوما من قبل الجميع ، ولكنها في الحقيقة مفيد أيضا بما يكفي.

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

1. ماذا (التحليل) ..... What (Analysis)
2. كيف (التصميم)..... How (Design)
3. فعل ذلك (التنفيذ)..... Do It (Implementation)
4. الاختبار (اختبار وتصحيح الاخطاء) Test (Testing and Debugging)
5. استخدام (او عملية النشر) Use (Operation or
6. الصيانة (التحسين) Maintain (Refinement) .Deployment)

مهم
الطريقة الموصى بها لكتابة البرامج هي الاجراء الذي اتبعناه في سكربت 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/السكربتات الخاصة في بايثون والمراحل المختلفة التي تشارك في كتابة مثل هذه البرامج. وربما تجد انه من المفيد انشاء برامجك بنفسك مثلما فعلنا في هذا الفصل حتى يتسنى لك ان تصبح مرتاحا مع بايثون فضلا عن القدرة على حل المشاكل. وفيما يلي؛سوف نناقش البرمجة الكائنية " object-oriented"

الفصل الحادي عشر

البرمجة الكائنية الموجهة

Object-Oriented Programming

***مقدمة:**

في جميع برامجنا وحتى الآن ، لقد قمنا بتصميم برنامجنا حول دوال أو كتل من البيانات التي تتلاعب بالبيانات. ويسمى هذا طريقة البرمجة الإجرائية الموجهة **procedure-oriented_**. وهناك طريقة أخرى لتنظيم برنامجك الذي هو الجمع بين الوظيفة والبيانات وتغليفها معا فيما يسمى بالكائن object. وهذا ما يسمى نموذج البرمجة الكائنية التوجه. في معظم الوقت يمكنك استخدام البرمجه الاجراءيه ولكن في بعض الاحيان عندما تريد كتابة برامج كبيرة او ان يكون هذا هو الـ حل الأنسـ ب لـ ها ، يمكـ نك ا ستخدام تقنيات البرمـ جة كائنـ ية التو جه .

الطبقات Classes والكائنات objects يعتبران هما الأشكال الرئيسة للبرمجه الكائنية الموجهة. فالطبقة تخلق نوعا جديدا حيث تعتبر الكائنات أمثلة من الطبقة. أحد الأقيسة على ذلك هو انه يمكن ان يكون لديك متغيرات من نوعية العدد الصحيح int والتي تترجم الى قولنا ان المتغيرات التي تخزن الاعداد الصحيحه هي المتغيرات التي تعتبر حالات (أوكائنات objects) من الطبقة int .

ملاحظة لمبرمجي C/Java/C++/#C:

نلاحظ انه حتى الاعداد الصحيحه تعامل على انها كائنات (من الطبقة int). وهذا بخلاف C++ وجافا (قبل الاصدار 1.5) حيث الاعداد الصحيحه هي انواع بدائية الأصل. انظر help(int) لمزيد من التفاصيل حول الطبقة class .

unboxing و *boxing* سيجدون ذلك الأمر مألوفا إليهم حيث أنه يشبه مفهوم Java 1.5 و C# **مبرمجو**

يمكن للكائنات تخزين البيانات باستخدام المتغيرات العاديه التي تنتمي الى هذه الكائنات. والمتغيرات التي تنتمي الى الطبقة او الكائن تسمى حقول fields . يمكن للكائنات أن يكون لها ايضا مهام وظيفيه باستخدام الدوال التي تنتمي الى الطبقة. هذه الدوال تسمى طرق أو اساليب methods لهذ الطبقة. هذه المصطلحات مهمة لانها تساعدنا في التفريق بين الدوال والمتغيرات التي هي مستقلة في حد ذاتها ، وتلك التي تنتمي الى طبقة معينة او كائن ما. وكلها جميعا ، الحقول وال طرق يمكن ان يشار اليها على انها صفات لتلك الطبقة.

الحقول تتكون من نوعين – يمكن لكل منهما ان تنتمي الى حالة / كائن instance/object من الطبقة ، أو يمكنها ان تنتمي الى الطبقة نفسها. فهي تسمي متغيرات الحالة ومتغيرات الطبقة على التوالي.

الطبقة يتم إنشاؤها باستخدام كلمات المفتاحية(المحجوزة) للطبقة class keyword. الحقول وطرق الطبقة مدرجة في منظومة الكتلة.

*الذات *The self*

_يوجد لأساليب الطبقة فارق واحد محدد يخالف الدوال العادية -- وذلك انها يجب أن تكون لها اسم أول إضافي يضاف الى بداية باراميترالقائمة ، ولكنك لا تعطي قيمة لهذا الباراميتر عندما تستدعي ال method، وسوف يقدمها لـ نا بايثون . هذا المتغير المحدد يشير الى الكائن ذاته ، وحسب الاتفاق ، فإنها تحظى باسم 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 المتخصصة ل IDE (بيئة التطوير المتكامله) يمكن ان تساعدك اذا كنت تستخدم self .

*** ملاحظة لمبرمجي C++/Java/C#***

عبارة self في بايثون تعادل المؤشر self في لغة C++ و الإشارة this في جافا و C#

عليك ان مندهشا ؛ كيف أن بايثون يعطي قيمة ل self ؟ ولماذا أنت لست بحاجة الى اعطاء قيمة لها ؟.أحد الأمثلة سيجعل هذا الأمرواضحا. لنقل مثلا أن لديك class تدعى myclass ومثال هذه الطبقة يسمى myobject. عندما تستدعي ال method لهذا الكائن كما يلي : (MyObject.method(arg1, arg2 ، فإنه يتم تحويلها تلقائيا عن طريق بايثون إلى MyClass.method(MyObject, arg1, arg2 - وهذا كل ما يخص self

وهذا يعني ايضا انه اذا كان لديك طريقة لا تأخذ أي argument ، فإنك لا تزال بحاجة الى تحديد طريقة للحصول على self argument .

*Object Methods:

إضافي . self يمكنها أن تحتوي طرقا مثل الدوال إلا إذا كان لدينا متغير classes/objects لقد ناقشنا بالفعل أن ال . والآن سوف نرى مثالا على ذلك

*Object Methods: استخدام

Example 11.2. Using Object Methods

```
#!/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()
```

Output

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

***كيف يعمل البرنامج :**

ولكن ما تزال parameters لا تأخذ أي معاملات sayHi المسماة method تعمل . لاحظ أن ال self هنا ؛ نرى . بداخل الدالة self تحتوي على

***The __init__ method:**

__init__ يوجد العديد من أسماء الطرق التي لها اهمية خاصة في طبقات بايثون. وسنرى ما المغزى من طريقه الآن.

تعمل بمجرد عمل الكائن المنتمي للطبقة . هذه الطريقة مفيدة لفعل اي تهيئة تريد القيام بها مع __init__ طريقة كلاهما في بداية الاسم وفي نهايته (underscore__) الكائن الخاص بك. لاحظ الشرطة السفلية المزدوجه.

***استخدام __init__ method:**

Example 11.3. Using the __init__ method

```
#!/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()
```

Output

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

***كيف يعمل البرنامج :**

المعتاده). وهنا ، نقوم بمجرد self طريقة لتأخذ اسم الباراميتر (جنبا إلى جنب مع __init__ هنا ، قمنا بتحديد طريقة dotted .لاحظ أن هناك متغيرين مختلفين رغم انها تحمل نفس الاسم . name إنشاء حقل جديد يسمى ايضا notation يسمح لنا ان نفرق بينهما.

بداخل القوسين arguments ولكن نقوم بتمرير __init__ والأهم من ذلك ، لاحظ اننا لا نستدعي صراحة طريقة وهذا هو المغزى الخاص من هذه .class جديدة من هذه ال instance عندما ننشئ خلق حالة class بعد اسم ال الطريقة.

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

ملاحظة لمبرمجي* C++/Java/C#

طريقة __init__ مماثلة لـ ***constructor*** في #C++/Java/C

: *Class and Object Variables

لقد سبق أن ناقشنا بالفعل الجزء النتعلق بوظيفة الطبقات والكائنات ، والان سنرى جزء البيانات الخاص بها. في الواقع ، انها ليست سوى متغيرات عادية مرتبطة بفراغات أسماء الطبقات والكائنات . هذه الأسماء صالحة ضمن سياق هذه الطبقات والكائنات فقط.

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

متغيرات الطبقة تشترك في معنى انها تعمل منة خلال جميع الكائنات (الحالات) لهذه الطبقة. لا يوجد سوى نسخة من متغير الطبقة وعندما يقوم الكائن بعمل على متغير الطبقة ، ينعكس هذا التغيير في جميع الحالات الاخرى ايضا. متغيرات الكائن يملكها كل فرد من الكائن / المثال object/instance" " في الطبقة. وفي هذه الحاله ، كل كائن له نسخة خاصة به من الحقل أي أنها ليست مشتركة ولا ترتبط باي شكل من الاشكال مع الحقل الذي بنفس الاسم في instance مختلفة من نفس الطبقة. وهذا المثال سيجعل من السهل فهمها.

***استخدام متغيرات الكائن والطبقة Using Class and Object Variables**

Example 11.4. Using Class and Object Variables

```
#!/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()
```

Output

```
$ 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 البرنامج :**

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

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

لاحظ بأن __init__ طريقة تُستَعمل لعمل initialize للحالة Person مَع name . وفي هذه الطريقةِ، نزيد عدد population بمقدار 1 حيث أن لدينا واحد Person يصبح مضافا . كذلك نلاحظ أن قيم self.name تحدد لكل كائن يشير إلى طبيعة متغيرات الكائن .

تذكر أنه يجب أن تشير غلى المتغيرات والطرق الخاصة بنفس الكائن باستخدام المتغير self فقط . وذلك يدعى إشارة خاصة . في هذا البرنامج نرى أيضا استخدام docstrings للطبقات وكذلك الطرق . يمكننا الوصول إلى class docstring في وقت التشغيل runtime باستخدام Person.__doc__ والطريقة docstring كـ Person.sayHi.__doc__ . مثل __init__ method ،ويوجد طريقة خاصة أخرى __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 ننقوم ببساطة بإنقاص حساب ال Person.population بمقدار 1 .

طريقة __del__ تعمل عندما يكون الكائن غير مستخدم ، وليس هناك ضمان أن هذه الطريقة ستعمل .وإذا أردت عمل ذلك بوضوح عليك فقط أن تستخدم البيان del الذي استعملناه في الأمثلة السابقة .

ملاحظة لمبرمجي C++/Java/C :

تعتبر تخيلية في methods وكل ال *public* إضافة إلى عناصر البيانات) تعتبر عمومية) class كل عناصر ال بايثون .وهناك استثناء واحد ؛ غذا طنت تستخدم عناصر البيانات مع الأسماء باستخدام شرطة سفلية مزدوجة __ يستخدم بايثون صقل الاسم بفاعلية ليجعل لها قيمة خاصة . __privatevar خاصة مثل private variable .

_ هكذا فإن الاتفاقية المتبعة هي أن كل متغير يستعمل فقط داخل الطبقة أو الكائن يجب أن يصبح بشرطة سفلية underscore ويمكن أن تستخدم من قبل أي *public* وجميع الأسماء الأخرى عامة classes/objects .
تذكر أن هذه اتفاقية فحسب وليست إجبارية من بايثون (ما عدا بادئة الشرطة السفلية المزدوجة __ double underscore prefix)

التوارث Inheritance

أحد المنافع الرئيسة للبرمجة الكائنية الموجهة هو إعادة استعمال الكود ، وأخذ وسائل ذلك يتم عمله من خلال آلية التوارث Inheritance mechanism .

التوارث يمكن تخيله بشكل افضل على أنه تطبيق علاقة بين نوع رئيس ونوع فرعي بين الطبقات . 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 الطبقات لكل نوع وتعالجهما ، ولكن بإضافة خاصية مشتركة جديدة ، معناها إضافتها إلى كل طبقة مستقلة . وسريعا يصبح هذا الأمر ثقيلا جدا . والطريقة الأفضل يمكن ان تكون بإنشاء طبقة مشتركة تسمى SchoolMember ، وبعدها تجعل طبقة teacher وطبقة student ترث من هذه الطبقة الأولى (SchoolMember) . وبمعنى آخر سيصبحان أنواع فرعية sub-types لهذه الطبقة . وبعد ذلك يمكننا أن نحدد خصائص هذه الأنواع الفرعي sub-types .

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

الميزة الأخرى انه يمكنك أن تشير إلى كائنات المعلمين أو الطلبة باعتبارها كائن SchoolMember الذي يمكن أن مفيدا في بعض الحالات مثل حساب عدد أعضاء المدرسة . وذلك يسمى تعدد الأوجه polymorphism ؛ حيث ان النوع الفرعي sub-type يمكن أن يستبدل في أي حالة عندما يكون النوع الأصل متوقعا . فمثلا الكائن يمكن تكراره بصفته حالة من الطبقة الأصلية . ونلاحظ كذلك أننا نعيد استخدام كود الطبقة الأصل ،ولسنا بحاجة إلى تكراره في طبقات مختلفة ، كما كان واجبا في حالة ما استخدمنا طبقات مستقلة .

الطبقة المسماة SchoolMember في هذه الحالة تعرف بأنها الطبقة الأساسية أو superclass ، طبقة Teacher وطبقة Student تسمى طبقات مشتقة derived classes أو طبقات فرعية subclasses . وسنرى الآن هذا المثال التالي في صورة برنامج .

استخدام التوارث Using Inheritance

Example 11.5. Using Inheritance

```
#!/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 Students
```

Output

```
$ 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 البرنامج :**

لاستعمال التوارث ، نقوم بتحديد اسم الطبقة الأساسية base class في tuple متبوعا باسم الطبقة في تعريف الطبقة . بعد ذلك نلاحظ أن وسيلة __init__ الخاصة بالطبقة الأساسية تستدعى بشكل واضح باستخدام المتغير self ، من أجل ذلك يمكننا إعداد جزء الطبقة الأساسية في الكائن . من الأمور المهمة التي عليك أن تتذكرها ،إن بايثون لا يستدعي الدالة المشيِّدة "constructor"" للطبقة الأساسية بطريقة آلية ، وعليك أن تقوم باستدعائها بشكل واضح بنفسك.

كذلك نلاحظ أنه يمكننا أن نستدعي وسائل الطبقة الأساسية عن طريق تقديم اسم الطبقة لنداء ال method ، وبعد ذلك نمر إلى المتغير self مع أي arguments . لاحظ أنه يمكننا أن نعالج حالات Teacher أو Student كمجرد حالات لـطبقة SchoolMember .

ونلاحظ كذلك أن الوسيلة tell الخاصة بالنوع الفرعي يتم استدعاؤها وليست الخاصة بالطبقة SchoolMember.

أحد الطرق لفهم ذلك هو أن بايثون دائما يبدأ في البحث عن الوسائل methods في النوع ، وهوفي هذه الحالة يفعل ذلك. وغذا لم يستطع إيجاد ال method فإنه يبدا في البحث عن ال methods المنتمية إلى الطبقات الأساسية واحدة تلو الخرى من أجل أنها محددة في tuple في تعريف الطبقة .

ملاحظة خاصة بالمصطلح –إذا كان هناك أكثر من فئة مندرجة في قائمة tuple التوريث ،عندئذ تسمى التوريث المتعدد .

لقد قمنا الآن باستكشاف الجوانب المختلفة للطبقات والكائنات فضلا عن مختلف المصطلحات المرتبطه بها. وقد شهدنا ايضا فوائد ومطبات البرمجة الكائنية الموجهة. بايثون تعتبر لغة برمجة عالية المستوى في مجال البرمجة الكائنية الموجهة وفهم هذه المفاهيم بعناية سيساعدك كثيرا في المدى البعيد.

وفيما يلي ، سوف نتعلم كيفية التعامل مع المدخلات والمخرجات وكيفية الوصول الى الملفات في بايثون.

الفصل الثاني عشر

Input/Output

سيكون هناك الكثير من الأوقات عندما ترغب في إعطاء قدرة لبرنامجك على التفاعل مع المستخدم (ويمكن أن تكون أنت نفسك هذا المستخدم). انك تريد إن تأخذ مدخلات من المستخدم ، ثم تطبع بعض النتائج إلى الوراء. ولا يمكننا أن نحقق ذلك باستخدام raw_input والبيان print على التوالي. بالنسبة لل output، يمكننا أيضا استخدام مختلف أساليب class (string) str. على سبيل المثال ، يمكنك استخدام طريقة rjust لتحصل على سلسلة نصية string ، والذي هو حق مبرر right justified لعرض محدد . اذ ظر (help(str المز يد من التفا صيل .

يوجد نوع شائع آخر من المدخلات والمخرجات input/output هي التعامل مع الملفات. القدرة على إنشاء ، وقراءة وكتا بة الملفات أ مر أ ساسي لكث ير من البرا مج ، و سنبحث في هذا الجا نب في هذا الفص ل .

*الملفات:
يمكنك فتح واستخدام الملفات للقراءة أو الكتابة عن طريق إنشاء كائن للطبقة file واستخدام أساليب read, readline أو write بشكل مناسب للقراءة من الملف أو الكتابة إلى الملف. القدرة على القراءة أو الكتابة إلى ملف يتوقف على الأسلوب الذي قمت بتحديده لفتح الملف. ثم أخيرا ، وعندما تنتهي من الملف ، يمكنك استدعاء أسلوب close لتبلغ بايثون بأننا انتهينا من استخدام الملف .

Example 12.1. Using files

```
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
```

Output

```
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 حالة/ instance من الطبقة file عن طريق تحديد اسم الملف والنمط / mode التي نريد فتح الملف بها. النمط يمكن أن يكون للقراءة واسطة ('r') ، أو نمط للكتابة ('w') أو نمط مشترك ('a') ، وهناك في الواقع العديد من الأنماط المتاحة ، help(file) سوف تعطيك المزيد من التفاصيل عنها.

- أولا نفتح ملف في نمط الكتابة واستخدام أسلوب write للطبقة file للكتابة إلى الملف ثم أخيرا 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 أسلوب readline ، في حلقة/ loop. هذه الطريقة إلينا سطرا كاملا بالإضافة إلى سطر جديد نهاية الخط. ، ولذا عندما يرجع إلينا سطر فارغ ، فهو يشير إلى أن نهاية الملف قد تم الوصول إليها وتتوقف الحلقة/ loop.
- ولاحظ أننا نستخدم فاصلة مع بيان print لمنع حدوث سطر جديد تلقائيا ،والذي يضيفه البيان print لأن السطر الذي يقرأ من الملف بالفعل ينتهي مع إشارة سطر جديد. ثم ، أخيرا close هذا الملف.
- الآن ، اطلع على محتويات الملف poem.txt للتأكد من أن البرنامج يعمل بشكل صحيح.

Pickle

بايثون توفر لنا وحدة معيارية/standard module تدعى pickle ، تستخدم في إمكان تخزين أي كائن/Object في بايثون في ملف واحد ، ثم تحصل عليها لاحقا دون مساس. وهذا ما يسمى تخزين الكائن على الدوام.

وهناك وحدة/module أخرى تسمى cpickle والتي تعمل بالضبط نفس ما يقوم به الموديل pickle ؛ إلا انه مكتوب بلغة C وهو أسرع بمقدار(١٠٠٠ مرة أو أكثر). يمكنك استخدام أي من هذه الوحدات/modules ، على الرغم من أننا سوف تستخدم الوحدة cpickle هنا. تذكر ، نحن نشير إلى أن كل من هذه الوحدات ببساطة مجرد الموديل pickle

Pickling and Unpickling

Example 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
```

Output

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

***كيف يعمل البرنامج**

أولا ، نلاحظ أن نستخدم التركيب اللغوي import..as. و هو سهل المنال حيث يمكننا استخدام اسم اقصر من اجل الموديل. وحتى في هذه الحالة ، فإنه يتيح لنا الانتقال إلى موديل مختلف (cPickle or pickle) من خلال تغيير بسيط لسطر واحد! في بقية البرنامج ، ونحن ببساطة نشير إلى هذه الموديل ، كـ p
لتخزين كائن في الملف ، أولا نقوم بفتح الكائن file في نمط الكتابة وتخزين الكائن في الملف المفتوح عن طريق استدعاء الدالة dump من الموديل pickle . هذه العملية تسمى pickling .

بعد ذلك ، ونسحب الكائن باستخدام الدالة load للموديل pickle الذي يعيد الكائن. هذه العملية تسمى unpickling

***الخلاصة**

لقد ناقشنا مختلف أنواع المدخلات والمخرجات وأيضا معالجة الملفات واستخدام الموديل pickle.
وفيما يلي سنبحث في مفهوم الاستثناءات exceptions .

الفصل الثالث عشر

الاستثناءات Exceptions

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 يتولاها بايثون والذي يرفع يديه {منبها لك} ويخبرك أن هناك خطأ .

*الأخطاء Errors

نظرة بسيطة إلى print statement. ماذا لو أخطأنا إملائيا في كتابة print وكتبناها كـ Print؟ لاحظ الحرف الكابيتال والحرف السمولوفي هذه الحالة ، بايثون يرفع إلينا أن ثمّة خطأ لغوي syntax error.

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

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

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

الاستثناء Try

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

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

بايثون يرفع إلينا خطأ يدعى EOFError والذي يعني أساسا أنه تم العثور على نهاية الملف عندما لم نكن نتوقعه (الذي يتمثل من خلال الضغط Ctrl-d)

وفيما يلي ، سنرى كيفية التعامل مع مثل هذه الأخطاء.

معالجة الاستثناءات .. Handling Exceptions

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

Example 13.1. Handling Exceptions

```
#!/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'
```

Output

```
$ 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 الـ statements التي قد تثير خطأ في كتلة try block ومن ثم محاولة معالجة جميع الأخطاء والاستثناءات في except . البند except / الكتلة except يمكنه معالجة خطأ أو استثناء واحد محدد ، أو قائمة الجمل المعترضة (بين قوسين) للأخطاء / الاستثناءات. إذا لم يكن هناك أسماء من الأخطاء أو الاستثناءات المعطاة ، ستعالج جميع الأخطاء والاستثناءات. ويجب أن يكون هناك بند except واحد على الأقل مرتبط مع كل بند من try .

إذا كان أي خطأ أو استثناء لم يعالج فإن المعالج الافتراضي لبايثون يستدعي و يوقف تنفيذ البرنامج ويطبع رسالة. وقد رأينا بالفعل في هذا العمل.
ويمكننا أيضا أن يكون لديك البند else مرتبط بكتلة try..catch . البند else يتم تنفيذه عند عدم وجود أي استثناءات.
يمكننا كذلك الحصول على exception object لذا يمكننا استرجاع معلومات إضافية حول الاستثناء الذي حدث. ويتجلى هذا في المثال التالي.

رفع الاستثناءات Raising Exceptions

يمكنك رفع الاستثناءات باستخدام raise statement. يجب عليك أيضا أن تحدد اسم الخطأ /الاستثناء ، والـ exception object يكون موضوعا جنبا إلى جنب مع الاستثناء/ exception. الخطأ أو الاستثناء الذي يمكنك رفعه ينبغي أن يكون class والتي تعتبر بشكل مباشر أو غير مباشر طبقة مشتقة عن الطبقة Error أو الطبقة Exception على التوالي.

How To Raise Exceptions

Example 13.2. How to Raise Exceptions

```
#!/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.'
```

Output

```
$ 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 يمكن أن تستخدم أي استثناء/ خطأ محدد سلفا لأغراض إيضاحية. وهذا النوع من الاستثناء الجديد هو الكلاس ShortInputException. وهي تحتوي على حقلين -- length وهو طول المدخلات ، و atleast والذي هو أقصر طول كان يتوقعه البرنامج.

في البند except ، نذكرك بالكلاس error ، فضلا عن المتغير الذي يقوم بإجراء المقارنة مع الكائن الخطأ / الاستثناء. والتي تعتبر مماثلة لل parameters وال arguments في استدعاء الدالة. وفي داخل الكلاس الخاص except نستخدم object الحقلين: length و atleast لطباعة رسالة مناسبة للمستخدم.

Try..Finally

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

Using Finally

Example 13.3. Using 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'
```

Output

```
$ 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 بطاقم العمل file-reading كالمعتاد ، ولكني كنت أدخلت طريقة اعتباطية من sleeping لمدة 2 ثانية قبل طباعة كل سطر باستخدام طريقة time.sleep. والسبب الوحيد لذلك هو أن البرنامج يعمل ببطء (بايثون سريع جدا بطبيعته) . وعندما يظل البرنامج يعمل ، اضغط Ctrl-c لمقاطعة/إلغاء البرنامج.

نلاحظ أن الاستثناء KeyboardInterrupt يُلقى و البرنامج في طريقه للخروج ، ولكنه قبل انتهاء البرنامج .يتم تنفيذ البند finally ويتم إغلاق الملف.

ملخص:

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 سنبحث مكتبة بايثون القياسية.

الفصل الرابع عشر
مكتبة بايثون القياسية
The Python Standard Library

قائمة المحتويات ..... **Table of Contents**

مقدمة : مكتبه بايثون القياسية متاحة مع كل تركيب لبايثون. وهي تحتوي على عدد هائل من الوحدات/ modules المفيدة جدا. ومن الأهمية بمكان أن تعتادوا على مكتبة بايثون القياسية ؛ حيث إن معظم المشاكل يمكن حلها بسهولة وبسرعة إذا كنت تعرف هذه المكتبة من الوحدات البرمجية/modules.

سنبحث بعض من الوحدات modules/ المستخدمة في هذه المكتبة. يمكنك العثور على التفاصيل الكاملة لجميع الوحدات modules في مكتبة بايثون القياسية في قسم "مرجع المكتبة/Library Reference " في الوثائق التي تأتي مع تركيب بايثون الخاص بك.

*The sys module :
يحتوي هذا الموديل " sys" على وظيفة محددة من النظام system-specific functionality .
وقد رأينا بالفعل قائمة sys.argv الذي يحتوي على command-line arguments .

Command Line Arguments

Example 14.1. Using 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)
```

Output

```
$ 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 مخرج/ output .

عندما يعمل برنامج لبايثون ليس بطريقة تفاعلية ، هناك دائما عنصر واحد على الاقل في قائمة sys.argv الذي هو اسم البرنامج الحالي يصبح عاملا ويكون متاحا كـ sys.argv[0] حيث أن بايثون يبدأ العد من الصفر . command line arguments أخرى تلي 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 اذا كان أي من الخيارات محددة لبرنامجنا. إذا كان الخيار --version مستخدما ، يتم طباعة رقم إصدارة البرنامج. وبالمثل ، عندما نحدد الخيار help -- ، نعطي قليلا من الشرح حول البرنامج. نحن نستفيد من استعمال دالة sys.exit للخروج من البرنامج. وكما هو الحال 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 بايثون التي قمت بتثبيتها . و

التيوبل / tuple المسماة sys.version_info تعطيك طريقة أسهل لإتاحة أجزاء محددة من إصدارة بايثون 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 المحنكين : العناصر الأخرى ذات الأهمية في الوحدة (الموديل) sys تتضمن sys.stdout ، sys.stdin و sys.stderr تتطابق مع standard input، و standard output و standard error في مجريات برنامجك على التوالي .

الموديل "os" The os module

هذه الوحدة البرمجية تمثل وظيفة عامة لنظام التشغيل operating system. هذه الوحدة لها أهمية خاصة إذا كنت تريد عمل منصات مستقلة لبرامجك - اي أنه 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 أو إعداد متغيرات البيئة على التوالي .
- الدالة os.listdir() تعيد أسماء كل الملفات والمجلدات في الدليل الحالي .
- الدالة os.listdir() تستخدم لحذف أحد الملفات .
- الدالة os.system() تستخدم لتشغيل أمر للشل .
- 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 موجود بالفعل .

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

ملخص

قد رأينا بعضا من وظائف الموديلز sys في مكتبة بايثون القياسية . ينبغي عليك أن تبحث في وسائق بايثون القياسية لتحصل على المزيد حولها والمزيد من الموديلز كذلك .

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

الفصل الخامس عشر

المزيد من بايثون

More Python

جدول المحتويات

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

Special Methods :

هناك بعض الأساليب الخاصة التي لها أهمية خاصة في الطبقات/classes مثل أساليب __init__ و__del__ والتي لها أهمية قد شهدناها بالفعل.

عموما ، الأساليب الخاصة تستخدم لتقليد سلوك معين. فعلى سبيل المثال ، إذا أردت ان تستعمل x[key] لعملية الفهرسة الخاصة بك من أجل class لديك (مثل التي تستخدمها في القوائم وtuples) ثم مجرد تنفيذ أسلوب __getitem__() ويتم عملك . إذا كنت تفكر في ذلك ، فهذا ما يقوم بايثون بعمله مع طبقة list نفسها!

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

Table 15.1. Some Special Methods

الاسم	الشرح
__init__(self, ...)	وهذه الطريقة تستدعى فقط عندما يعود الكائن المنشأ حديثا للاستعمال
__del__(self)	تستدعى فقط قبل أن يتم تدمير الكائن
__str__(self)	تستدعى عندما print مع الكائن أو عندما نستخدم str() نستخدم البيان
__lt__(self, other)	وبالمثل يوجد أساليب خاصة لجميع العوامل "+،>،إلخ " < "Less than تستدعى عند استخدام العامل
__getitem__(self, key)	تستدعى عندما تستخدم عملية الفهرسة x[key]
__len__(self)	تستدعى عند استعمال len()المتسلسلة/ لكائن sequence الدالة المدمجة

كتل التصاريح المفردة..Single Statement Blocks:

والآن ، ينبغي أن يكون لديك فهم راسخ أن كل كتلة من البيانات هي جزء من بقية أخواتها ذات نفس مستوى التثليم/indentation {راجع معنى indentation في الفصل الرابع}. حسنا ، هذا صحيح بالنسبة لمعظم الأجزاء ، ولكنها ليست دقيقة 100 ٪. إذا كانت كتلة البيانات لا تتضمن سوى بيان واحد ، حينئذ يمكنك أن تحدده على نفس السطر ، لنقل مثلا ، conditional statement أو looping statement.
والمثال التالي يشرح ذلك بوضوح :

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

كما يمكننا أن نرى ، فإن التصريح الواحد يستخدم في داخل ذات المكان - وليس كبند مستقل من الكتلة. على الرغم من ذلك ، يمكنك استخدام هذا لجعل برنامجك أصغر ، وإنني أوصي بشدة ألا تستخدم طريقة الـ short-cut هذه باستثناء حالة التحقق من الأخطاء ، الخ.أحد الأسباب الرئيسية أنه سيكون من الأسهل بكثير إضافة تصريح/ statement إضافي إذا كنت تستخدم التثليم/ indentation السليم .

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

List Comprehension تضمين القائمة

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.

مثاليه لمثل هذه الحالات List comprehensions.

Using List Comprehensions

Example 15.1. Using List Comprehensions

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

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

Output

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

***كيف يعمل**

هنا ، نشتق قائمة جديدة من خلال تحديد التلاعب الذي ينبغي القيام به (i*2) عندما تقع 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 قائمة ، ونفس الشيء يمكن أن يتحقق باستخدام list comprehensions وهي طريقة أكثر دقة ، وإحكاما ، ووضوحا .

Receiving Tuples and Lists in Functions استقبال التيوبل والقوائم في الدوال

وهناك طريقة خاصة ، لاستقبال معاملات/ parameters الدالة بوصفها tuple أو قاموس باستخدام بادئة * أو ** على التوالي. وهذا أمر مفيد عندما نأخذ عدد متغير من arguments 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
```

وبسبب البادئة * على المتغير args ، وجميع arguments الإضافية التي تمرر إلى الدالة يتم تخزينها في args بوصفها tuple.وإذا كانت البادئة ** قد استخدمت بدلا من * ، يجب أن ينظر إلى المعاملات/ parameters لتكون أزواج من مفتاح/قيمة key/value للقاموس.

Lambda Forms:

يستخدم التصريح lambda لإنشاء كائنات دالة جديدة وبعدها إرجاعها أثناء وقت التشغيل/ runtime

Using Lambda Forms

Example 15.2. Using Lambda Forms

```
#!/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)
```

Output

```
$ python lambda.py
```

```
wordword
10
```

Using Lambda Forms

Example 15.2. Using Lambda Forms

```
#!/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)
```

Output

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

كيف يعمل :
هنا ؛ استخدمنا الدالة make_repeater لإنشاء كائنات دالة جديدة في وقت التشغيل/ runtime وإرجاعها .
التصريح lambda يستخدم لإنشاء كائن للدالة . في الأساس ، lambda تأخذ معاملا/parameter متبوعا بتعبير/ expression واحد فقط والذي يصبح الجسم لهذه الدالة ، وقيمة هذا التعبير يتم إرجاعها من خلال الدالة الجديدة . لاحظ أنه حتى التصريح print لا يمكن أن يستخدم داخل lambda form . ولكن مع التعبيرات فقط

The exec and eval statements

يستخدم لتنفيذ بيانات بايثون التي يتم تخزينها في سلسلة نصية أو ملف. على سبيل المثال ، يمكننا توليد سلسلة نصية تختوي كود لبايثون عند وقت التشغيل/ runtime ومن ثم تنفيذ هذه البيانات باستخدام البيان **exec** . وهذا مثال بسيط تراه بأسفل :

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

البيان eval
يستخدم لحساب/تقييم تعبيرات بايثون الصالحة التي تخزن في السلسلة النصية كما ترى في المثال بأسفل:

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

تصريح "assert" The assert statement"
يستخدم تصريح "assert" لتأكيد أن شيئا ما true . فعلى سبيل المثال ، إذا كنت متأكدا بأن لديك واحدا على الأقل من العناصر في قائمة ، وأنت تريد أن تستخدمه للتحقق من ذلك ، وترفع خطأ إذا لم يكن true ، حينئذ يعتبر تصريح "assert" مثاليا في هذه الحالة.
وعندما يفشل التصريح "assert" يرتفع لنا AssertionError .

الدالة "repr" .. The repr function"

الدالة repr تستخدم للحصول على تمثيل قانوني لسلسلة نصية لكائن .الـ Backticks (وتسمى أيضا تحويل أو عكس الاقتباسات) تفعل الشيء نفسه. لاحظ أنه سيكون لديك eval(repr(object)) == object معظم الوقت.

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

أساسا ، الدالة repr او backticks تستخدم للحصول على تمثيل للكائن قابلة للطبع.
يمكنك التحكم فيما تعيده الكائنات الخاصة بك للدالة repr من خلال تحديد طريقة __repr__ في ال class الخاصة بك.

خلاصة* :
لقد تناولنا المزيد من مميزات بايثون في هذا الفصل، و يمكنك التأكد من أننا لم نغطَّ بعد كل ملامح بايثون. ولكن، في هذه المرحلة، نكون قد غطينا معظم ما سوف تستخدمه في

تطبيقاتك . وهذا الأمر فيه الكفاية لك لتبدأ أيا من البرامج أنت في سبيلك لإنشائها.

في الفصل المقبل، سوف نناقش كيفية استكشاف المزيد من بايثون.

الفصل السادس عشر

What Next?- وماذا بعد؟

قائمة المحتويات

إذا كنت قد قرأت هذا الكتاب بعناية حتى الآن، وطبقته من خلال كتابة العديد من البرامج، فلابد أنك أصبحت متآلفا ومستريحا مع بايثون الآن. وربما تكون أنشأت بعض البرامج لاستكشاف المادة العلمية لديك وتطبيق المهارات التي اكتسبتها في بايثون. إذا لم تقم بذلك بالفعل، فينبغي عليك إن تفعل. والسؤال الآن هو 'ماذا بعد؟'.

وأود أن اقترح عليك معالجة هذه المشكلة: اصنع لنفسك برنامج address-book بسطر الأوامر - باستخدام ما يمكنك إضافة أو تعديل أو حذف، أو البحث عن جهات الاتصال الخاصة بك؛ مثل الأصدقاء والأسرة والزملاء، ومعلوماتهم مثل عنوان البريد الإلكتروني و / أو رقم الهاتف. التفاصيل يجب تخزينها واسترجاعها في وقت لاحق.

هذا أمر سهل إلى حد ما ، إذا فكرت في جميع العناصر المختلفة التي نعتقد أننا مررنا عبرها حتى الآن. وإذا كنت تريد توجيهات بشأن كيفية المضي قدما، فإليك ذلك التلميح.

تلميح

(ينبغي ألا تكون قرأت ذلك).

* أنشئ class لتمثيل معلومات الشخص.
*استخدم القاموس لتخزين كائنات الشخص مع اسمه باعتباره مفتاح/key.
*استخدم الموديل cPickle لتخزين الكائنات باستمرار على القرص الصلب.
*استخدم الأساليب المدمجة للقاموس لإضافة وحذف وتعديل الأشخاص.

عندما تكون قادرا على أن تفعل كل ذلك، يمكنك أن تدعي أنك مبرمج بايثون. والآن، وعلى الفور أرسل لي بريد شكر لهذا الكتاب العظيم ☺. هذه الخطوة اختيارية ولكني أوصيك بها.

إليك بعض الطرق لمواصلة رحلتك مع بايثون :

البرمجيات الرسومية Graphical Software

مكتبات GUI {واجهة المستخدم الرسومية-Graphical User Interface} باستخدام بايثون -- أنت بحاجة إليها لعمل برامجك الرسومية باستخدام بايثون. يمكنك إنشاء irfanview او kuickshow الخاصة بك أو أي شيء مثل ذلك باستعمال مكتبات GUI في بايثون مع الأغلفة الخاصة بها. الأغلفة هي التي تسمح لك الكتابة في برامج بايثون واستخدام المكتبات التي تمت كتابتها في حد ذاتها بلغة C أو C++ أو غيرها من اللغات.

هناك الكثير من الخيارات لـ GUI باستخدام بايثون :

- **PyQt**. هذه هي تغليف بايثون لصندوق أدوات **Qt** الذي هو الأساس الذي بنيت عليه **Qt** KDE. 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 برامج طبقا لرخصة (GPL) على لينكس/ يونكس وتدفع المقابل إذا أردت إنشاء برمجيات ذات ملكية. من المصادر الجيدة لـ **PyQt** هو 'Programming with Python: Qt Edition' انظر الصفحة الرسمية official homepage لمزيد من التفاصيل.
- **PyGTK** . هذه هي تغليف بايثون لصندوق أدوات GTK+و الذي هو الأساس الذي بنيت عليه GNOME.

GTK+ لديها الكثير من المراوغات في الاستعمال ، وذلك بمجرد أن تصبح مرتاحا معها ، ويمكنك إنشاء تطبيقات GUI سريعا. المرور بمصمم الواجهة الرسومية أمر لا غنى عنه. الوثائق الخاصة بتحسين GTK+ تعمل جيدا على لينكس ولكن ميناءها إلى ويندوز لم يكتمل بعد. يمكنك أن تصنع البرمجيات الحرة وكذلك البرمجيات المملوكة على GTK+ . انظر الصفحة الرسمية official homepage للمزيد من التفاصيل.

- **wxPython** . هذه تغليف بايثون لصندوق أدوات wxWidgets.

wxPython لها منحنى تعليمي خاص بها ، ورغم ذلك فهي محمولة/ portable جدا وتعمل على لينكس , ويندوز , ماك , وحتى على منصات العمل المدمجة embedded platforms . يوجد العديد من IDEs { بيئات التطوير المتكاملة} متاحة لـ **wxPython** بالإضافة إلى مصممات GUI مثل (Stani's Python Editor (SPE و مصمم الواجهات wxGlade . يمكنك أن تصنع البرمجيات الحرة وكذلك البرمجيات المملوكة على **wxPython** . انظر الصفحة الرسمية official homepage للمزيد من التفاصيل .

- **TkInter** هذه واحدة من أقدم صناديق الأدوات في الوجود . إذا سبق لك واستخدمت IDLE فلابد أنك رأيت **TkInter** أثناء العمل . الوثائق الخاصة بـ **TkInter** **على موقع** PythonWare.org شاملة . **TkInter** محمولة/ portable وتعمل على كل من لينكس/يونيكس والوندوز على حد سواء . والأهم أن **TkInter** **جزء من توزيعة بايثون القياسية .**
- للمزيد من الخيارات ، انظر : GuiProgramming wiki page at Python.org

ملخص عن أدوات GUI

لسوء الحظ أنه لا يوجد أداة قياسية واحدة ل GUI على بايثون . إقترح عليك أن تختار واحدة من تلك الأدوات المذكورة أعلاه ،وذلك يعتمد على موقفك أنت . العامل الأول في تحديد اختيارك هو : هل يمكنك الدفع لشراء أي من أدوات GUI ؟ . العامل الثاني : أي المنصات تريد لبرنامجك أن يعمل عليها ؟ لينكس أم وندوز أو كلاهما ؟ .
العامل الثالث : أي الواجهات الرسومية تستخدمها على لينكس ؛ KDE أم GNOME؟ .

الفصول المستقبلية:

لقد فكرت في كتابة فصل أو فصلين لهذا الكتاب عن برمجة الواجهات الرسومية. ومن المحتمل أن يكون اختياري لـ wxPython كخيار لصندوق الأدوات. إذا أردت إن تقدم وجهات نظر حول هذا الموضوع يمكنك الانضمام إلى القائمة البريدية : byte-of-python mailing list حيث أتبادل النقاش مع القراء حول التحسينات التي يمكن عملها لها الكتاب .

استكشاف المزيد :

المكتبة القياسية لبايثون مكتبة شاملة. وفي معظم الوقت ، ستجد في هذه المكتبة ما تبحث عنه. وهذا يشار إليه بوصفه فلسفة 'البطاريات الإضافية' في بايثون. أنا أوصى بشدة بأن تتجول خلال الوثائق القياسية لبايثون قبل المتابعة في بدء كتابة برامج كبيرة بلغة بايثون.

- Python.org – هذه هي الصفحة الرئيسية الرسمية لبايثون. ستجد أحدث الإصدارات من لغة بايثون ومفسر للغة. وهناك أيضا مختلف القوائم البريدية حيث تجري المناقشات النشطة حول مختلف جوانب بايثون.
- Comp.lang.python هي مجموعة الأخبار على الشبكة ، حيث يجري النقاش حول هذه اللغة. يمكنك إرسال رسائلك واستفساراتك إلى مجموعة الأخبار تلك. يمكنك الوصول إلى هذه المجموعات على الانترنت باستخدام Google Groups أو الانضمام إلى القائمة البريدية mailing list التي هي مجرد انعكاس لمجموعة الأخبار.
- Python Cookbook : كتب قيم للغاية حيث جمع مجموعة من الوصفات أو النصائح حول كيفية حل بعض أنواع المشاكل باستخدام بايثون. هذا الكتاب لا بد من قراءته لكل مستخدم لبايثون.
- Charming Python : سلسلة مقالات قيمة للغاية عن بايثون. كتبها David Mertz. .
- Dive Into Python : وهو كتاب جيد جدا لذوي الخبرة من مبرمجي بايثون. إذا قرأت تماما هذا الكتاب الحالي فأنت الآن تقرأ، ولكن سأوصيك غاية الوصية أن تقرأ Dive Into Python 'الغوص في بايثون' بعد ذلك. وهو يشمل مجموعة من المواضيع بما في ذلك لغة الترميز القابلة للامتداد والتجهيز ،XML Processing Unit Testing و Functional Programming.
- Jython : هو أحد تطبيقات مفسر بايثون في لغة جافا. وهذا يعني انه يمكنك كتابة برامج في بايثون واستخدام مكتبات جافا كذلك! Jython برنامج مستقر وناضج. إذا كنت مبرمج جافا كذلك، وأنا أنصحك بشدة بأن تعطي jython محاولة منك.
- IronPython : هو تطبيق لمفسر بايثون في اللغة C#. ويمكن تشغيله على منصة .NET / Mono / DotGNU. وهذا يعني انه يمكنك كتابة برامج في بايثون واستخدام. مكتبات NET والمكتبات الأخرى التي توفرها هذه المنصات الثلاث كذلك . Iron python ما تزال برمجية بمرحلة pre-alpha ويصلح فقط للتجريب حتى الآن.

Jim Hugunin ، والذي كتب ironPython قد انضم إلى شركة مايكروسوفت ، وسيتم العمل من اجل التوصل إلى الإصدارة الكاملة من ironPython في المستقبل .

- Lython: هو واجهة Lisp frontend للغة بايثون. وهي مشابهة لـ Lisp وتترجم مباشرة إلى bytecode بايثون ، الأمر الذي يعني أنها سوف تعمل داخليا مع كود بايثون المعتاد.

- وهناك العديد والعديد من الموارد في بايثون. ومن الأمور المهمة موقع Daily Python-URL! الذي يجعلك على اطلاع دائم ومحدث على آخر أحداث بايثون ، وكذلك هذه Vaults of Parnassus, ONLamp.com Python DevCenter, dirtSimple.org, Python Notes والكثير والكثير .

الخلاصة:
لقد وصلنا الآن إلى نهاية هذا الكتاب ولكن، كما يقولون "هذا هي بداية النهاية! ". أنت الآن وأكثر من أي وقت تعتبر مستخدما نهما لبايثون، وأنت بلا شك مستعد لحل العديد من المشاكل باستخدام بايثون. يمكنك البدء في ميكنة {أتمتة} جهازك للقيام بكل الأنواع التي كانت في الماضي أمورا لا يمكن تخيلها، أو اكتب ما تريد من ألعاب خاصة بك والكثير الكثير. ولذا، إشرع في البدء ! .

☺☺*☺*

الحمد لله الذي بنعمته تتم الصالحات... والصلاة والسلام على محمد وعلى آله وصحبه
بأتم التسليمات
تم بعون الله وفضله الانتهاء من ترجمة هذا الكتاب الرائع في ليلة الخميس
11من ربيع الآخر 1429هـ - الموافق 17 من أبريل 2008م - الساعة 2.45 صباحا
أسأل الله السميع البصير أن يكون في ميزان حسناتنا وأن ينفع به إخواني القراء...
أخص منهم أعضاء مجتمع لينكس العربي http://www.linuxac.org
برجاء من إخواني قارئي هذا الكتاب.. لا تحرمونا من دعوة صادقة من القلب بظاهر الغيب

كتبه:أشرف علي خلف [kaspersky0]
الإسكندرية - مصر