تعلم بايثون بكل سهولة

بقلم:

فهد بن عامر السعيدي

OMLX

2005

مقدمة

لغة بايثون تتميز بالسهولة من حيث الكتابة و القراءة ومن حيث التركيب النحوي لها، و تستطيع أن تبرمج بها في كل المنصات المعروفة الآن، فهي تضارع لغة Java العريقة، و تصلح لكل الأغراض تقريبا.

ولقد شجعني على كتابة هذه الورقات القليلة، التي أسأل الله أن يمن علي بإتمامها قلة المصادر العربية التي تتكلم عن هذه اللغة، وكأن هذه اللغة ليست موجودة في عالم التقنية، فتجشمت عناء كتابة هذه الورقات.

ملاحظات حول الكتاب:

كتب هذا الكتاب على منصة أرابيان 0.6 باستخدام برنامج OpenOffice.org 2.0 لذا إذا حاولت فتحه على منصة الويندوز فلربما يذهب التنسيق، فحاول أن تعدل عليه في المنصة التي إنشأ فيها لتجنب إعادة الجهد من جديد.

هذا الكتاب ناقص، ويسعد المؤلف أن يجد أناس عندهم الحماس لإكماله، حتى يتم العمل بصورة جيدة.

معلومات عن الكتاب:

تأليف: فهد بن عامر السعيدي
بريد الكتروني fahad.alsaidi@gmail.com
ص.ب 370
الخابورة- سلطنة عمان
الرمز البريدي 326

الوحدة الأولى : أساسيات Python

الفصل الأول : البداية

الفصل الثاني: الأنواع و العمليات

الفصل الثالث: التعابير الأساسية

الفصل الثالث: الوظائف

الفصل الرابع: الوحدات

الفصل الخامس: الصفوف

الفصل السادس:الاستثناءات

الفصل الأول : البداية

النقاط المهمة:

لماذا لغة Python ؟

كيف تشغل برنامج مكتوب بلغة Python؟

معرفة بيئات التطوير الخاصة بـ Python

في هذا الفصل سنمضي سويا في رحاب لغة بايثون، لنعرف كيف نشأة لغة بايثون؟ و لما ذا نتعلم لغة بايثون؟ و ما مجالات التي تصلح لها لغة بايثون؟ كل هذه الأسئلة ضرورية لمن يريد أن يبدأ في تعلم أي لغة ! ، فلننتقل سريعا ...

لماذا لغة Python ؟

في عام ١٩٩٠م قام Guido van Rossum باختراع لغة بايثون، وقد استقى هذه اللغة من عدة لغات سابقة من أمثال: C و C++ و Modula-3 و ABC و Icon.

وتعتبر بايثون من اللغات النصية التي لا تحتاج إلى بناء لتشغيل البرنامج المكتوب بها، وهي من اللغات السهلة و المنظمة بشكل صارم مما أهلها أن تكون الخيار الأول في صنف اللغات الأكاديمية التي تعتمد في الجامعات، و سنلخص مميزات لغة بايثون في جدول و نبين فائدة كل مميزة ..

المميزات	***الفوائد***
لا تحتاج إلى بناء أو ربط مثل لغة السي	تسريع دورة التطوير فيها بشكل ملحوظ
لا تحتوي على أنواع المتغيرات	البرامج تصبح أسهل و أبسط و أكثر مرونة
إدارة آلية للذاكرة	مجمع النفايات يجنبك الجهد في تنقيح الكود
برمجة غرضية التوجه	يمنحك التكامل مع C++ و Java و COM
إمكانية التضمين و التمدد مع لغة السي	تحسين الأداء و إمكانية التحاور مع النظام
البساطة و الوضوح في قواعد الكتابة و التصميم	درجة عالية من المقروئية و إمكانية الصيانة و سهولة التعليم
محمولية عالية	تعمل على عدة منصات: الويندوز و اللينكس و الماكنتوش و اليونكس بدون تغيير الكود
مفتوحة المصدر	تعطيك الحرية في توزيعها و التعديل فيها و ضامن لبقائها

المميزات	الفوائد
دعم أنواع البيانات و العمليات عالية المستوى	تطوير سريع باستخدام أنواع الكائنات المدمجة
تحميل ألي لوحدات السي	امتدادات مبسطة و تصغير الملفات التنفيذية
دعم بروتوكولات الإنترنت القياسية	تسهيل الوصول إلى البريد الالكتروني و HTTP ..
كثرة المكتبات المضمنة و من أطراف ثالثة	تسريع و تسهيل تطوير البرامج

و أهم مميزاتها أنها سهلة التعلم ، وهذا ما ستلحظه أثناء تعلمك لها مع قوتها في آن واحد مما جعلها الخيار الأمثل لكثير من الشركات، و سنذكر مجموعة منها:

- *Yahoo Maps*
- *Yahoo Groups*
- *Google*
- *Ultraseek*
- *Jasc Software, Paint Shop Pro*
- *National Weather Service*
- *NASA*
- *Red Hat*
- *SGI, Inc*
- *IBM*
- *Real Networks*

ما مجالات لغة بايثون ؟

باختصار شديد، بايثون خاضت جميع المجالات التي تتطلب سرعة التطوير و السهولة في المجال الأول، و تأخرت قليلا في المجالات التي تحتاج إلى سرعة التطبيق، فمن المجالات التي تتميز فيها لغة بايثون:

- مجال الأدوات التي تتعامل مع النظام مباشرة
- مجال برمجة الإنترنت
- مجال برمجة واجهات المستخدم الرسومية
- مجال برمجة قواعد البيانات
- مجال البرمجة الموزعة

وغيرها الكثير من المجالات، مما سهّل لها وجود أدوات كثيرة تسهل عمل المبرمج بشكل ملاحظ، فمن أشهر هذه الأدوات:

المجال	الأدوات
برمجة النظام	Sockets و threads و signals و pipes و RPC و calls و POSIX bindings
واجهات المستخدم الرسومية	Tk و PMW و MFC و X11 و wxPython و KDE و Gnome
واجهات قواعد البيانات	Oracle و sybase و PostGres و mSQL و dbm و persistence
أدوات Microsoft Windows	MFC و COM و ActiveX و ASP و ODBC و .NET
أدوات الانترنت	Jpython و CGI tools و HTML/XML parsers و Zope و email tools
الكائنات الموزعة	DCOM و CORBA و ILU و Fnorb
أدوات أخرى مشهورة	SWIG و PIL و regular expressions و numPy و cryptography

كيف تشغل برنامج مكتوب بلغة Python؟

بما أن لغة بايثون نصية ، فإنه يتوجب عليك أن يكون لديك مفسر اللغة فقط لتشغيل البرنامج، و للحصول على المفسر اذهب إلى موقع لغة بايثون ، و نزل المفسر حسب النظام الذي تعمل عليه :

http://www.python.org

لغة بايثون في نظام اللينكس من اللغات الأساسية، فهي افتراضيا مثبة على النظام لذلك يفضل استخدام هذا النظام ، وخاصة أن هذه الدروس ستكون مبنية على توزيعة أرابيان و لكن لا يمنع هذا من العمل على منصة الويندوز

بطبيعة الحال لكتابة برنامج بايثون ستحتاج إلى محرر نصوص ثم حفظ الملف بلاحقة .py

بعد ذلك ادخل على سطر الأوامر و اكتب:

```
>> python program.py
```

بحيث program اسم الملف، و للتعامل مبا شرة مع محث لغة بايثون ، اكتب في سطر الأوامر :

```
>> python
```

و لكتابة أول برنامج لك ، اكتب :

```
>>> print ' My name is Fahad Al- Saidi'
```

بعد كتابتك لأول برنامج لك بوا سطة بايثون، ستعرف بنفسك مدى سهولة بايثون و قوتها.

بيئات التطوير الخاصة بـ Python

سترغب مع الوقت في امتلاك بيئة تطوير تسهل عليك أداء الكثير من العمليات الروتنية، وهناك الكثير من بيئات التطوير الخا صة بلغة بايثون ، ما عليك إلا أن تختار حسب رغباتك ، فقط تابع هذين الرابطين :

http://wiki.python.org/moin/PythonEditors
http://wiki.python.org/moin/IntegratedDevelopmentEnvironments

و لكن يكفيك أن تملك مفسر لغة بايثون مع محرر نصوص عا دي لتكمل مشوارك معنا في تعلم لغة بايثون

الفصل الثاني: الأنواع و العمليات

النقاط المهمة:

بنية برنامج بايثون

لماذا نستخدم الأنواع المدمجة؟

الأعداد

السلاسل النصية

القوائم

القواميس

المجموعات

الملفات

الخصائص العامة للكائنات

تلميحات مهمة

الملخص

بنية برنامج بايثون

قبل البدء في خوض غمار تفا صيل لغة بايثون، من المهم أن نعرج على بنية البرنامج المكتوب بلغة بايثون، ليتكون لديك تصور واضح بشكل عام حول البرمجة بلغة بايثون، وباختصار نقول: برنامج بايثون يمكن أن يحلل إلى ثلاثة أجزاء: وحدات، و عبارات، وكائنات، على نحو التالي:

١- البرنامج يتكون من وحدات

٢- و الوحدات تحتوي على عبارات

٣- و العبارات تنشأ الكائنات و تعالجها.

لماذا نستخدم الأنواع المدمجة؟

إذا كنت قد برمجت سابقا با ستخدام السي أو السي بلس بلس أو الجافا ستدرك كم من الوقت

يستغرق ترتيب الكائنات و تنسيقها في الذاكرة بما يسمى بنى المعطيات،وكم هو مرهق للأعصاب ترتيب تلك الكائنات في الذاكرة و التعامل مع الذاكرة و كيفية الو صول إليها و ترتيبها و البحث من خلالها..

في برامج بايثون المثالية ،معظم ذلك الجهد يذهب عنك بعيدا ، لأن بايثون تزو دك بكائنات من صلب اللغة تجعل برمجة تلك الأ شياء بمنتهى السهولة، فقط فكر في حل المشكلة ثم اكتب الحل، لا داعي من الآن فصاعدا أن ترهق نفسك في ترتيب الكائنات. في الحقيقة، كل ما تريده ستجد تلك الأنواع تزو دك به ما لم تكن بحاجة إلى أنواع خا صة.

ستجد في غالب الأحيان أنك تفضل تلك الكائنات ،لعدة أ سباب منها:

كائنات المدمجة تجعل البرامج البسيطة سهلة الكتابة

للمهمات البسيطة ستجد أن أنواع الكائنات المضمنة تلبي جميع احتياجاتك بعيدا عن مشاكل بنى المعطيات.لأن ستجد الأ شياء مثل المجموعات (القوائم) و الجداول (القواميس) في متناول يدك، و ستجد أن كثيرا من العمل أنجز فقط با ستخدام كائنات بايثون المدمجة.

بايثون تزو دك بالكائنات و تدعم التو سعات

بايثون تستعير في بعض الطرق من اللغات التي تدعم الأ دوات المضمنة مثل Lisp و اللغات التي تعتمد على المبرمج في تزويدها بالأ دوات المطلوبة أو إطار العمل الذي يحتاجه مثل C++

و بالرغم من أنك ستطيع أن تنشيء أنواع كائنات فريدة في بايثون إلا أنك في الغالب لا تحتاج إلى ذلك.

الكائنات المدمجة أكثر كفاءة من بنى المعطيات المخصصة

الأنواع المدمجة في بايثون تستخدم خوارزميات بنى معطيات محسنة و معمولة بالسي لزيا دة السرعة، وبالرغم أنك تستطيع أن تكتب أنواع كائنات مشابهة إلا أنك ستبذل جهدا مضاعفا للحصول على أ داء أنواع الكائنات في بايثون.

الجدول التالي يبين أنواع الكائنات المدمجة التي سنأخذها في هذا الفصل، إ ذا كنت قد ا ستعملت لغة قبل ذلك ستجد أن بعض هذه الكائنات متشابهة مثل (الأعدا د و السلا سل

النصية و الملفات) و ستجد أيضا أنواع قوية وعامة مثل (القوائم و القواميس) تزودك بها بايثون بدون تعب خلافا لكثير من اللغات مثل C و C++ و Java.و سنقوم بشرحها واحدا واحدا

نوع الكائن	مثال عليه
الأعداد Numbers	3.1415, 1234, 999L, 3+4j
سلاسل نصية Strings	'spam', "guido's"
قوائم Lists	[1, [2, '4'], three']
قواميس Dictionaries	{'food':'spam', 'taste':'yum'}
المجموعات Tuples	('spam', 4, 'U',1)
الملفات Files	text = open('eggs', 'r').read()

الأعداد

تدعم لغة بايثون تشكيلة واسعة من أنواع الأعداد : الأعداد الصحيحة و الأعداد ذات النقطة العائمة، و هذا مألوف لمن تعامل مع لغات سابقة، و أيضا تدعم أنواع مركبة من الأعداد مثل الأعداد المركبة و الأعداد ذات دقة الفاصلة العائمة غير محدود و تشكلية واسعة من الأدوات

و فيما يلي سنشرح النوعين:

أنواع الأعداد القياسية:

بايثون تدعم الأنواع القياسية في بقية اللغات و أتت بأنواع جديدة و في ما يلي جدول يبين تلك الأنواع مع أمثلة لكل نوع:

النوع	***تفسيره***
1234, -24, 0	Normal integers (C longs)
999999999999L	Long integers (unlimited size)
1.23, 3.14e-10, 4E210, 4.0e+210	Floating-point (C doubles)
0177, 0x9ff	Octal and hex constants
3+4j, 3.0+4.0j, 3J	Complex number constants

-الأعداد الصحيحة و الأعداد ذات الفاصلة العائمة

الأعداد الصحيحة هي مجموعة من الأعداد العشرية مثل (10) ، و الأعداد ذات الفاصلة العائمة هي التي تحتوي على فاصلة مثل (10,10).

-الدقة الرقمية

الأعداد الصحيحة تعامل معاملة longs في لغة السي مما يعني أنها غير محدودة بدقة رقمية، و الأعداد ذات الفاصلة العائمة تعامل معاملة doubles في لغة السي مما يعني أنك يمكنك أن تكتب أي رقم يخطر على بالك، مع ملاحظة أنك إذا ألحقت حرف L أو l فإنك تخبر مفسر لغة بايثون أن هذا العدد عدد صحيح طويل على مقاييس لغة باثيون.

-الأعداد الثمانية و الستعشرية

الأعداد الثمانية هي التي تبدأ بالصفر 0 و الأعداد الستعشرية هي التي تبدأ بـ 0x أو 0X.

-الأعداد المركبة

لغة بايثون تزودك بهذا النوع و هو يكتب كالتالي (الجزء الحقيقي + الجزء التخيلي) (real-part + imaginary-part) و ينتهي باللاحقة j أو J .

تعابير المعاملات في بايثون:

ربما الأداة الأساسية في معالجة الأعداد هي التعابير وهي عبارة عن مجموعة من الأعداد (أو كائنات أخرى) و معاملات تنتج قيمة عند تنفيذها في بايثون، و المثال على ذلك عندما تريد أن تجمع عددين مثل x و y فإنك تقول x + y فالمعامل هنا + .

تزودنا بايثون بقائمة طويلة من هذه المعاملات و الجدول التالي يبينها و يشرحها و يبين أسبقيتها عند التنفيذ تنازليا:

المعامل	الوصف
x or y	معامل المنطقي "أو"
x and y	معامل المنطقي "و"

المعامل	الوصف
not x	معامل المنطقي "عكس"
in, not in	اختبار العضوية الكائن
is, is not	اختبار هوية الكائن
x | y	معامل "أو" على مستوى البت
x ^ y	معامل "عكس" على مستوى البت
x & y	معامل "و" على مستوى البت
x << y, x >> y	إزاحة x يمينا أو شمالا بمقدار y من البتات
x + y, x – y	معامل الطرح و الجمع
x * y, x / y, x %y	معامل الضرب و القسمة و باقي القسمة

العمل على الأعداد

أحسن طريقة لفهم الأشياء النظرية هي تجربتها عمليا و واقعيا، فدعنا نشغل سطر الأوامر لنطبع عليه بعض الأسطر التي ستشرح ما قلناه سابقا عمليا.

العمليات الأساسية:

قبل كل شيء نحتاج إلى إنشاء كائن من فئة الأعداد مثل x و b ، لكي نطبق عليه معاملات ، في اللغات الأخرى ستحتاج إلى ذكر نوع الكائن ثم تسميته ثم إسناد قيمة إليه لكي نتعامل معه، ولكن في بايثون فقط سم الكائن ثم أسند إليه قيمه و تتولى الباقي بايثون للتعرف على نوعه ، وهذا بشكل عام في كائنات بايثون يكفي فقط اسناد القيمة إلى الكائن لتعريف بنوعه.

لتطبيق ذلك عمليا ،اكتب التالي في سطر الأوامر :

```
%python
>>> a = 3          # name created
>>> b = 4
```

وبهذا نكون قد أنشأنا كائنين من فئة الأعداد وأسندنا إليهما قيمتين ، ولعلك لاحظت كيفية إضافة التعليقات في بايثون فكل ما بعد # فهو تعليق ، وللتعليق أهمية كبرى في توضيح الكود و سهولة قراءته و سهولة تطويره من قبل مطورين آخرين.

بعد إنشاء الكائنات ستحتاج إلى تطبيق بعض المعاملات ، وكلما كانت المعاملات بين الأقواس كلما كانت الأمور أوضح ، ولكن هذا لا يعني أن المعاملات لا تعمل بدون الأقواس، ولكن انتبه إلى أسبقية المعاملات على حسب ما ذكر سابقا ، و إليك بعض الأمثلة :

```
>>> b / 2 + a      # same as ((4 / 2) + 3)
5
>>> b / (2.0 + a)  # same as (4 / (2.0 + 3))
0.8
```

فكما هو واضح في المثال الأول أن بايثون تولت ترتيب المعاملات في الأقواس، وبما أن المعامل القسمة أسفل من معامل الجمع فهو مقدم ، ولكن في المثال أضفنا إلى التعبير قوسين فأجبرنا بايثون على تنفيذ المعاملات التي نريد تقديمها أولا ، وفي النهاية يجب مراعاة مثل هذه الأمور عند كتابة برامجك في لغة بايثون.

ولعلك تسأل ما فائدة إضافة النقطة العائمة في المثال الثاني ، سأقول لك جرب تنفيذ المثال بدون إضافة النقطة العائمة و ستجد أن النتيجة هي 0 ، ما ذا حصل؟ لقد تعاملت بايثون على أنه عدد صحيح، ولكن عندما تكتب النقطة العائمة فإنك تخبر بايثون أني أريد النتيجة كما هي بدون تقريب وكذلك إذا أضفت صفرين ستلاحظ النتيجة بنفسك.

معاملات على مستوى البت:

ستحتاج إلى التعامل على المستوى البت في بعض الأحيان فيجب عليك أن تعرف العد الثنائي أولا و ستتضح لك الأمثلة التالية :

```
>>> x = 1       # 0001
>>> x << 2      # shift left 2 bits: 0100
4
>>> x | 2       # bitwise OR: 0011
3
>>> x & 1       # bitwise AND: 0001
1
```

إذا لم تفهم ولم تدرس الأعداد الثنائية ، فلا عليك ، استمر ، و ستجد أنه هذه الجزئية لا

يحتاجها إلا من يطلبها !!

الأعداد المركبة :

الأعدا د المركبة من تخصصات الهندسة و الكهربا ء ، كشخص لا تخصص عندك في مثل هذه الأمور ،مر على هذه الجزئية مرور الكرام ، لأنك لكي تفهمها ستحتاج إلى أكثر من فصل درا سي ، ونحن نريد أن نو صلك إلى فهم البرمجة با ستخدام الباثون في وقت قيا سي ، أما إ ذا كان هذه الأعدا د من لب تخصصه فإنك ستجد بايثون توفر لك الكثير ، فانظر إلى هذه الأمثلة:

```
>>> 1j * 1J
(-1+0j)
>>> 2 + 1j * 3
(2+3j)
>>> (2+1j)*3
(6+3j)
```

للا ستزا ده في هذا المجال راجع وثائق بايثون للمزيد من التفصيل.

المزيد من الأدوات :

توفر لك بايثون عدة أ دوات تتعامل مع الرياضيات ، وتقدم وحدة math التي تتعامل مع جميع قوانين الرياضيات ، ولكن يجب عليك أن تستور د هذه الوحدة ، و إليك المثال:

```
>>> import math
>>> math.pi
3.14159265359
>>>
>>> abs(-42), 2**4, pow(2, 4)
(42, 16, 16)
```

و سنتكلم عن وحدات بشيء من التفصيل في الفصول القا دمة بإ ذن الله تعالى .

السلاسل النصية

النوع الثاني من الكائنات المدمجة الرئيسية في بايثون هو السلاسل النصية،والسلاسل النصية عبارة عن تجمع حروف لتخزين اسم أو بيانات في وحدة واحدة ،وبمعنى آخر يمكنك استخدام السلاسل النصية في كل شيء يقبل تمثيله كنص مثل الروابط و الأسماء و الكلمات وما شابه ذلك.

إذا كنت استخدمت لغات أخرى قبل بايثون ستجد أنها تحتوي على الكائن سلاسل النصية و أيضا كائن حرف char في بايثون الحروف تعامل كسلاسل نصية و ذلك تسهيلا للتعلم وتسريعا للبرمجة.

بايثون تزودنا بالكثير من الدوال التي تتعامل مع السلاسل النصية مثل الفهرسة و التقطيع و معرفة طول السلاسلة و تجميع السلاسل ،وهناك وحدات مستقلة لمعالجة السلاسل النصية في بايثون مثل string و regex و re.

و الجدول التالي يعرض بشكل سريع كيفية إنشاء السلاسل النصية وبعض دوالها :

العملية	شرحها
S1 = ' '	سلاسلة فارغة
s2 = "spam's"	علامات اقتباس مزدوجة
block = """...""""	ثلاث علامات تنصيص
s1 + s2, s2 * 3	الجمع التكرار
s2[i], s2[i:j], len(s2)	الفهرسة التقطيع معرفة الطول
"a %s parrot" %'dead'	تهيئة السلاسل النصية
for x in s2, 'm' in s2	الحلقة تكرارية العضوية

لاحظ أنه لا فرق بين علامة الاقتباس المفردة و المزدوجة ، فكلهن يؤدين العمل نفسه، ولك حرية الأختيار.

العمل على السلاسل النصية

كما مر عليك أن إنشاء كائن من السلاسل النصية يكفي له أن تذكر اسمه وتسند له قيمة من نوع السلاسل النصية ، وقد مر عليك ثلاثة أمثلة في الجدول السابق، فدعنا نستكشف الجوانب الأخرى..

العمليات الأساسية

لقد مر عليك المعامل الجمع + و معامل الضرب * وعرفت كيف التعامل معها في كائنات الأعداد ، أما كائنات السلاسل النصية فهي تعامل الجمع كإضافة و الضرب كتكرار للنص ، ولكن يشترط في معامل الجمع، أن يكون كلا الطرفين سلاسل نصية.

وتزودنا بايثون بدالة تحسب لنا طول السلسلة النصية وهي الدالة len و هي مدمجة مع اللغة لا تحتاج إلى استيراد ، وهذه بعض الأمثلة:

```
%python
>>> len('abc')        # length: number items
3
>>> 'abc' + 'def'     # concatenation: a new string
'abcdef'
>>> 'Ni!' * 4         # like "Ni!" + "Ni!" + ...
'Ni!Ni!Ni!Ni!'
```

سنأتي الآن إلى عمل حلقة تكرارية في السلسلة النصية و ذلك باستخدام for و اختبار العضوية باستخدام in :

```
>>> myjob = "hacker"
>>> for c in myjob: print c,      # step though items
...
h a c k e r
>>> "k" in myjob                  # 1 means true
1
```

الفهرسة و التقطيع السلسلة النصية

بما أن السلا سل النصية تعرف في بايثون كأنها مجموعة من الحروف ، فإن هذا التركيب يعطينا مميزات المجموعة من إمكانية الو صول إلى أي من أعضا ءه بما يسمى المفهرس ، وكذلك توفر لنا بايثون إمكانية تقطيع تلك السلسلة با ستخدام المفهرس، ولكن لاحظ أن بايثون تبدأ العد من الصفر في المفهرس وليس الواحد مثل كل لغة مشتقة من السي، والآن إليك هذه الأمثلة :

```
>>> S = 'spam'
>>> S[0], S[-2]           # indexing from front or end
('s', 'a')
>>> S[1:3], S[1:], S[:-1]    # slicing: extract section

('pa', 'pam', 'spa')
```

قمنا أولا بتعريف متغير s بأنه سلسلة نصية وأ سندنا إليه قيمة معينة ، ثم قمنا بعملية الفهر سة من البداية ومن النهاية ، فـS[0] تعني أظهر الحرف الذي فهر سته 0 و الناتج كان s ، وS[-2] تعني أظهر الحرف الذي فهر سته 2 من الأخير.

بعد ذلك قمنتا بعملية التقطيع با ستخدام المفهرس فـS[1:3] تعني اجلب من الحرف الأول فما أعلى ولكن لا تجلب الحرف الثالث وما فوقه ، أماS[:1] فتعني اجلب من الحرف الأول فما فوقه إلى النهاية ، أما S[:-1] فتعني اجلب كل السلسلة ماعدا الحرف الأخير

تهيئة السلاسل النصية

إذا كان عندك سلسلة طويلة وأر دت أن تضيف إليها كائنات متغيرة فهناك عدة طرق، ولكن بايثون توفر لك طريقة تستعملها لغة السي بشكل كبير ،ولغة C# حديثا ، انظر المثال التالي:

```
>>> S = 'spam'
>>> S[0] = "x"
 Raises an error!

>>> S = S + 'Spam!'      # to change a string, make a new one
>>> S
'spamSpam!'
>>> S = S[:4] + 'Burger' + S[-1]
>>> S
'spamBurger!'
>>> 'That is %d %s bird!' %(1, 'dead')    # like C sprintf
That is 1 dead bird!
```

لاحظ أن بايثون أعادة تعريف المعامل % ليعمل مع السلاسل النصية ، وعند الأعداد كباقي القسمة، كما قلنا سابقا أن السطر الأخير استخدم هيئة السي في ترتيب النص وخاصة الدالة sprintf و أخذ كل قواعدها، وهي بسيطة تعني ما كل على اليسار يساوي ما على اليمين على الترتيب ، وإليك أمثلة أكثر على هذه التهيئة:

```
>>> exclamation = "Ni"
>>> "The knights who say %s!" %exclamation
'The knights who say Ni!'
>>> "%d %s %d you" %(1, 'spam', 4)
'1 spam 4 you'
>>> "%s -- %s -- %s" %(42, 3.14159, [1, 2, 3])
'42 -- 3.14159 -- [1, 2, 3]'
```

ولكن هل الحروف بعد % اعتباطية ؟ نقول لا ولكنها كل واحدة تدل على شيء ، أما %s فهي عامة لكل كائن سواء أكان عددا أو سلسلة نصية أو غيره ، أما البقية فالجدول التالي يشرحها:

%	سلسلة نصية String	%X	عدد ستعشري Hex integer
%c	حروف Character	%e	الفاصلة العائمة الهيئة 1 Floating-point
%d	عدد عشري Decimal	%E	الفاصلة العائمة الهيئة 2 Floating-point
%i	عدد صحيح Integer	%f	الفاصلة العائمة الهيئة 3 Floating-point
%u	Unsigned (int)	%g	الفاصلة العائمة الهيئة 4 Floating-point
%o	عدد ثماني Octal integer	%G	الفاصلة العائمة الهيئة 5 Floating-point

%x	Hex integer عدد ستعشري	%%	حرف %

أدوات العامة لسلاسل النصية

كما قلنا سابقا بايثون تزودنا بوحدات خاصة للتعامل مع السلاسل النصية، ولعل أشهر واحدة وأقواها هي string . فهي تزودنا بالعديد من الدوال فمنها القدرة على تحويل الحروف من الكبيرة إلى الصغيرة والعكس ،وكذلك البحث في السلاسل المعرفة ، وكذلك تحويل السلسلة النصية إلى عدد ، وغيرها الكثير ، راجع وثائق بايثون للمعرفة جميع الأدوات، وهذا مثال على قدرة وحدة string:

```
>>> import string          # standard utilities module
>>> S = "spammify"
>>> string.upper(S)        # convert to uppercase
'SPAMMIFY'
>>> string.find(S, "mm")     # return index of substring
3
>>> string.atoi("42")  # convert from/to string
42
>>> string.join(string.split(S, "mm"), "XX")
'spaXXify'
```

لعل المثال الأخير وهو الأكثر تعقيدا،فهو سهل الفهم فكل ما في القضية أن الدالة split قامت بتجزئة السلسلة إلى قسمين عند الحرف mm ، ثم قامت الدالة join بدمج XX بين القسمين السابقين ، يمكنك تجربة كل دالة واحدة ومعرفة كيفية عملها.

لاحظ أن الدالة atoi تقوم بتحويل السلسلة النصية إلى عدد فقط ، ولكن توجد دالة أخرى مدمجة تسمى eval تقوم بتحويل السلسلة النصية إلى أي نوع ولكنها أبطأ بطبيعة الحال من الدالة الأولى.

الاختلافات في كتابة السلسلة النصية

في نهاية كلمنا عن السلاسل النصية ، سنتكلم عن أحرف الهروب التي بها تستطيع أن تنسق النص بشكل جيد ، مثل حرف بداية السطر وغيرها والجدول التالي يبين لك هذه الأحرف في

بايثون:

\newline	الاستمرار	\n	سطر جديد
\\	إظهار \	\v	Tab عمودية
\'	إظهار علامة اقتباس واحدة	\t	Tab رأسية
\"	إظهار علامة اقتباس مزدوجة	\r	العودة إلى بداية السطر Carriage return
\a	جرس	\f	صفحة جديدة Formfeed
\b	مفتاح الحذف الخلفي Backspace	\0XX	Octal value XX
\e	زر الهروب Escape	\xXX	Hex value XX
\000	Null عدم إنهاء السلسلة	\other	أي حرف آخر

القوائم

نكمل مشوارنا في غمار لغة بايثون ونصل إلى كائن يعتبر أكثر الكائنات المدمجة مرونة وترتيب ألا وهو القائمة list ، تمتاز القوائم عن السلاسل النصية أنها تستطيع أن تجمع في ضمنها عدة كائنات وليست النصوص فقط ، فالقائمة يمكن أن تكون من سلسلة نصية و أعداد و كائنات أخرى حتى قوائم أخرى. وتقوم القائمة مقام بنى المعطيات في اللغات الأخرى مثل لغة السي و الجافا،وتتميز القوائم في بايثون بعدة مميزات منها:

مجموعة مرتبة من كائنات غير متجانسة

من الناحية الوظيفية، القائمة مكان للتجميع الكائنات لذا يمكنك أن تنظمهم كمجموعة و القائمة أيضا تقوم بترتيبهم من اليسار إلى اليمين .

الو صول با ستخدام المفهرس

مثل السلاسل النصية تستطيع أن تصل إلى أعضاء القائمة با ستخدام المفهرس، وإجراء عملية التقطيع والسلسلة.

مرونة عالية

تتمتع القوائم بمرونة عالية أكبر من مرونة السلاسل النصية، بحيث يمكن أن تكبر وتصغر حسب متطلبات برنامجك، ويمكنك وضع قوائم في قوائم بحيث تصير متشابكة.

مصفوفة من الكائنات المرجعية

من الناحية التقنية تعتبر القوائم نوع خاص من المصفوفات في لغة السي،وهي من هذه الناحية عبارة عن كائنات مرجعية، ونقصد هنا بقولنا مرجعية ، أي عندما نتعامل معها ب can't read superblockعد تعريفها فإننا نتعامل مع مؤشر يؤشر إلى الكائن وليس نسخة عنه، وهذا يعطينا سرعة أكبر و يسهل علينا البرمجة.

الجدول التالي يوضح أهم عمليات القوائم:

العملية	***الوصف***
L1 = []	قائمة فارغة
L2 = [0, 1, 2, 3]	أربعة عناصر و الأدلة من 0 إلى 3
L3 = ['abc', ['def', 'ghi']]	قوائم متداخلة
L2[i], L3[i][j] L2[i:j], len(L2)	الفهرسة التقطيع الطول
L1 + L2, L2 * 3	سلسلة إعادة
for x in L2, 3 in L2	تكرار العضوية
 L2.append(4), L2.sort(), L2.index(1), L2.reverse()	الدوال: توسيع ترتيب البحث عكس
del L2[k], L2[i:j] = []	الانكماش
L2[i] = 1, L2[i:j] = [4,5,6]	إسناد الفهرس اسناد المقطع
range(4), xrange(0, 4)	إنشاء قوائم أو مجموعات من الأعداد

ستلاحظ أنك قد رأيت معظم العمليات التي ذكرت في الجدول السابق قد مرت عليك في السلاسل النصية ، إلا بعض العمليات التي تدعمها القوائم ولا تدعمها السلاسل النصية مثل إسناد الفهرس و إسناد المقاطع و التوسع والانكماش.

العمل على القوائم

أفضل طريقة لفهم القوائم هي العمل عليها،ومرة أخرى سنأخذ العمليات التي ذكرت في الجدول السابق ونحاول أن نشرحها بشيء من الأمثلة العملية.

العمليات الأساسية

القوائم تدعم معظم العمليات التي تدعمها السلاسل النصية و إليك المثال التالي :

```
%python
>>> len([1, 2, 3])                # length
3
>>> [1, 2, 3] + [4, 5, 6]          # concatenation
[1, 2, 3, 4, 5, 6]
>>> ['Ni!'] * 4                    # repetition
['Ni!', 'Ni!', 'Ni!', 'Ni!']
>>> for x in [1, 2, 3]: print x,     # iteration
...
1 2 3
```

الفهرسة والتقطيع

بما أن القوائم عبارة عن سلاسل ،فإن عمليتي الفهرسة والتقطيع تعمل عليها بشكل جيد مثل السلاسل النصية مع ملاحظة الموقع الحقيقي لكل كائن ، وإليك هذا المثال:

```
>>> L = ['spam', 'Spam', 'SPAM!']
>>> L[2]                           # offsets start at zero
'SPAM!'
>>> L[-2]                          # negative: count from the right
'Spam'
>>> L[1:]                          # slicing fetches sections
['Spam', 'SPAM!']
```

الإسناد في القوائم

الأشياء التي أتت بها القوائم وتعتبر متميزة عن السلاسل النصية هي المقدرة على إسناد قيم جديدة إلى الفهارس والمقاطع في القوائم و ذلك بخلاف السلاسل النصية التي تحتاج لفعل ذلك إلى إنشاء نسخة جديدة من الكائن.

عند ا ستخدام القوائم تستطيع تغيير المحتوى با ستخدام الفهرس أو المقطع كما يبين ذلك المثال التالي :

```
>>> L = ['spam', 'Spam', 'SPAM!']
>>> L[1] = 'eggs'                # index assignment
>>> L
['spam', 'eggs', 'SPAM!']
>>> L[0:2] = ['eat', 'more']      # slice assignment: delete+insert
>>> L                         # replaces items 0,1
['eat', 'more', 'SPAM!']
```

مع ملاحظة أن بايثون حين تقوم بالإ سنا د إلى القوائم عن طريق المقاطع ، فإنها أولا تقوم بحذف المقطع المختار ثم تضع القيمة الجديدة مكانه ولو كانت القيمة أكثر من كائن وا حد، على سبيل المثال عندنا قائمة L تساوي [1, 2, 3] فعند تطبيق عملية الإ سنا د با ستخدام المقطع التالي [L[1:2] = [4, 5] فإن النتيجة ستكون [1, 4, 5, 3].

القوائم تدعم بعض الدوال منها تو سيع و الترتيب و البحث و العكس و إليك المثال التالي:

```
>>> L.append('please')
>>> L
['eat', 'more', 'SPAM!', 'please']
>>> L.sort()
>>> L
['SPAM!', 'eat', 'more', 'please']
>>> L.reverse()
>>> L
['please', 'more', 'eat', 'SPAM!']
>>> L.index('eat')
2
```

وأخيرا بما أن القوائم عبارة عن سلسلة مرنة ، فإنها تدعم الحذف با ستخدام الفهرس أو المقطع، و ذلك عن طريق الا سنا د إلى قائمة فارغة أو الحذف عن طريق الفهرس أو المقطع :

```
>>> L.sort()
>>> L
['SPAM!', 'eat', 'more', 'please']
>>> del L[0]                # delete one item
>>> L
['eat', 'more', 'please']
>>> del L[1:]               # delete an entire section
>>> L                     # same as L[1:] = []
['eat']
```

القواميس

بالأضافة إلى القوائم تعتبر القواميس من أهم الكائنات المدمجة المرنة في لغة بايثون،وإذا كنا اعتبرنا أن القوائم عبارة عن مجموعة من الكائنات المرتبة ، فإن القواميس بعكس ذلك فهي مجموعة من الكائنات غير المرتبة، ويعتبر الشيء الرئيسي في القوائم الذي يقوم عليه ترتيب و إحضار عنا صر القاموس هو المفتاح وليس موقع العنصر.

وكما تلاحظ فإن القواميس ا ستطاعت ا ستبدال الكثير من خوارزميات و بنى المعطيات التي ستضطر إلى ا ستخدامها يدويا في بعض اللغات منخفضة المستوى، وأيضا تستخدم القواميس في بعض الأحيان لأداء عمل الجداول في بعض اللغات منخفضة المستوى.

وتتميز القواميس بعدة خصائص منها:

الو صول با ستخدام المفتاح وليس الموقع

القواميس في بعض الأحيان يطلق عليها المصفوفات المترابطة،هذا الترابط يضع القيم با ستخدام المفاتيح، وبا ستطاعتك إحضار أي عنصر في القاموس با ستخدام المفتاح الذي خزن به، ستستخدم نفس عمليات الفهرس ولكن با ستخدام المفتاح وليس با ستخدام الموقع.

مجموعة غير منظمة من كائنات غير متجانسة

بخلاف القوائم، العنا صر في القواميس لا تبقى على ترتيب معين،في الحقيقة بايثون تقدم ترتيب عشوائي يضمن تقديم مشاهدة سريعة، المفاتيح تقدم راوبط (غير فيزيائية) إلى أماكن العنا صر في القواميس.

خصائص مرنة

مثل القوائم القواميس تزودك بميزة التوسيع والتقلص بدون إنشاء نسخة جديدة، وكذلك يمكنها تحتوي على عناصر من كل نوع،وكذلك ميزة التداخل بحيث يمكنك أن تنشأ قواميس في قواميس وكذلك يمكنك قوائم في قواميس،وأيضا يمكنك أن تسند قيم جديدة بالاعتماد على المفاتيح

جداول من كائنات المرجعية

إذا كنا قلنا أن القوائم عبارة عن مصفوفة من الكائنات المرجعية ، فإن القواميس عبارة عن جداول غير منظمة من الكائنات المرجعية. داخليا القواميس تستخدم جداول من بنى المعطيات تدعم ميزة الاسترجاع السريع وهي تبدأ صغيرة و تكبر حسب الطلب، وعلاوة على ذلك بايثون توظف خوارزميات محسنة لإيجاد المفاتيح مما يعطي الاسترجاع سرعة كبيرة. وعند التعمق نجد أن القواميس تخزن مراجع الكائنات وليس نسخ منها مثل القوائم بالضبط.

الجدول التالي يوضح أهم العمليات الشائعة على القواميس، لاحظ أنها تشابه القوائم. تكتب القواميس على شكل التالي: key:value

العملية	الوصف
D1 = { }	قاموس فارغ
d2 = {'spam': 2, 'eggs': 3}	عنصرين في القاموس
d3 = {'food': {'ham': 1, 'egg': 2}}	التداخل
d2['eggs'], d3['food']['ham']	الفهرسة باستخدام المفتاح
d2.has_key('eggs'), d2.keys(), d2.values()	الدوال : دالة العضوية قائمة المفاتيح قائمة القيم
len(d1)	الطول (عدد الإدخالات المخزنة)
d2[key] = new, del d2[key]	الإضافة والتعديل الحذف

العمل على القواميس

دعنا نرجع إلى المفسر لنأخذ حريتنا في تطبيق بعض العمليات التي ذكرت في الجدول السابق:

العمليات الأساسية:

بشكل عام ، يمكنك إنشاء قاموس والوصول إلى عناصره باستخدام المفتاح key ، و الدالة

len المدمجة تعمل أيضا مع القواميس، وهي ترجع عدد العناصر المخزنة في القاموس أو بمعنى آخر ترجع طول قائمة المفاتيح. وعند حديثنا عن المفاتيح فإن الدالة keys ترجع كل المفاتيح في القاموس مجموعة في قائمة، هذا يعتبر أداة قوية لمعالجة القواميس بشكل متسلسل، ولكن لا تعتمد عليه في ترتيب قائمة المفتايح ،(تذكر أن القواميس عشوائية).

```
%python
>>> d2 = {'spam': 2, 'ham': 1, 'eggs': 3}
>>> d2['spam']              # fetch value for key
2
>>> len(d2)                 # number of entries in dictionary
3
>>> d2.has_key('ham')       # key membership test (1 means true)
1
>>> d2.keys()               # list of my keys
['eggs', 'spam', 'ham']
```

التغيير في القواميس

كما قلنا أن القواميس غير مرتبة، فلذا تستطيع أن تكبر و تصغر و اسناد قيم جديدة أيضا ،بدون الحاجة إلى إنشاء قواميس جديدة مثل القوائم بالضبط، فقط عليك إسناد قيمة أو تغييرها لإنشاء مدخلة جديدة في القاموس. والدالة del تعمل أيضا على القواميس مثلما تعمل على القوائم بالضبط ، إليك المثال التالي:

```
>>> d2['ham'] = ['grill', 'bake', 'fry']     # change entry
>>> d2
{'eggs': 3, 'spam': 2, 'ham': ['grill', 'bake', 'fry']}

>>> del d2['eggs']                    # delete entry
>>> d2
{'spam': 2, 'ham': ['grill', 'bake', 'fry']}

>>> d2['brunch'] = 'Bacon'                # add new entry
>>> d2
{'brunch': 'Bacon', 'spam': 2, 'ham': ['grill', 'bake', 'fry']}
```

لاحظ هنا الطريقة في إنشاء مدخلة جديدة في القاموس المثال الأخير ، فهي تختلف عن طريقة

إضافة مدخلة الجديدة في القوائم ، و ذلك أن القواميس لا تعتمد على موقع بل على المفتاح فهي عشوائية و لا يهمها المكان، بعكس القواميس فهي تحتاج إلى الدالة append لإضافة مدخلة جديدة في القائمة.

مثال واقعي

سنأخذ هنا مثال أكثر واقعية وهو إنشاء جدول يحوي أسماء اللغات -ثلاث هنا- في العمود الأول -المفتاح- والعمود الثاني يحوي أسماء مؤلفيها – القيمة-،ركز في هذا المثال وحاول تطبيقه:

```
>>> table = {'Python': 'Guido van Rossum',
...          'Perl':   'Larry Wall',
...          'Tcl':    'John Ousterhout' }
...
>>> language = 'Python'
>>> creator  = table[language]
>>> creator
'Guido van Rossum'
>>> for lang in table.keys(): print lang, '\t', table[lang]
...
Tcl     John Ousterhout
Python  Guido van Rossum
Perl    Larry Wall
```

لاحظ الأمر الأخير،لأن القواميس ليست متسلسلة فلا يمكن أن تكررها بشكل مباشر باستخدام for مثل السلاسل النصية أو القوائم. ولكن إذا أردت أن تعرض جميع عناصر القاموس، فقم أولا باستخدام الدالة keys لجلب قائمة المفاتيح في القاموس ثم كررها واعرض قيمتها باستخدام for ، إذا لم تكن تعرف for فلا تتضايق فسوف ندرسها بالتفصيل في الدروس القادمة.

وهنا بعض الملاحظات يجب أن تتذكرها حول القواميس قبل أن ننتقل إلى الكائنات الأخرى:

-العمليات التي تعتمد على التسلسل لا تعمل مع القواميس مثلما رأينا في المثال الأخير

-عند إسناد قيمة جديدة فإنك تضيف مدخلة جديدة في القاموس

-ليس شرطا أن يكون المفتاح سلسلة نصية ، بل تعمل كل الكائنات ماعدا القوائم

المجموعات

آخر نوع في ا سكشافنا لبايثون لأنواع المجمعات هي المجموعات، المجموعات تتركب ببساطة من مجموعة من الكائنات، وهي تعمل بالضبط مثل القوائم با ستثنا ء أن القوائم لا يمكن أن تتغير من مكانها فهي ثابتة وتكتب عا دة كعنا صر متسلسلة محصورة بين قو سين وليس بين قو سين معكوفين مثل القوائم. والمجموعات تستمد معظم خصائصها من القوائم وهي:

مجموعة منظمة من الكائنات

مثل السلا سل النصية والقوائم المجموعات عبارة عن مجموعة من الكائنات المنظمة في نسق معين ومثل القوائم يمكن أن تحتوي على كل أنوا ع الكائنات

الو صول با ستخدام الموقع

مثل السلا سل النصية والقوائم يمكنك الو صول إلى أي عنصر في المجموعة با ستخدام موقعه وليس مفتا حه، و المجموعات تدعم كل العلميات التي تستخدم الموقع والتي سبق وأن أخذناها مثل الو صول با ستخدام الفهرس والتقطيع.

سلسلة ثابتة من الكائنات

مثل السلا سل النصية المجموعات ثابتة ويعني ذلك أنها لا تدعم أي عملية تغيير في المكان (الا سنا د) مثل التي رأيتها في القوائم، وهي أيضا لا تدعم التو سع والتقلص بل يجب لفعل ذلك إنشا ء نسخة جديدة من المجموعة المرا د تكبيرها أو تصغيرها.

مصفوفة من الكائنات المرجعية

مثل القوائم بالضبط، المجموعة عبارة عن مصفوفة من الكائنات المرجعية.

الجدول التالي يوضح أهم العمليات على المجموعات، مع ملاحظة لكي تنشي ء مجموعة فارغة فقط يكفي وضع قو سين فقط.

العملية	الوصف
()	مجموعة فارغة
t1 = (0,)	عنصر واحد في المجموعة
t2 = (0, 1, 2, 3)	أربعة عناصر
t2 = 0, 1, 2, 3	أربعة عناصر أيضا

العملية	الوصف
t3 = ('abc', ('def', 'ghi')) t1[i], t3[i][j] t1[i:j], len(t1)	التداخل الفهرسة التقطيع الطول
t1 + t2 t2 * 3	الجمع الإعادة
for x in t2, 3 in t2	التكرار العضوية

أربعة الصفوف الأولى في الجدول تستحق إيضاحا أكثر، بسبب أن الأقواس تستخدم في إغلاق المعاملات (راجع الأعداد) فإنك تحتاج إلى شيء مميز لإخبار بايثون أن كائنا واحدا بين القوسين هو من فئة المجموعات وليس عبارة عن تعبير بسيط،بكل بساطة ضع فاصلة سفلية قبل إقفال القوسين، وبذلك تخبر بايثون أن ما بين القوسين هو عبارة عن مجموعة.

وكحالة خاصة فإن بايثون تتيح لك الحرية في وضع الأقواس أو لا في إنشاء المجموعات كما في الصف الرابع، ولكن يفضل دائما إذا سمحت لك الفرص بأن تستخدم الأقواس لأنها تضمن لك عدم التشويش.

في العمليات الأخيرة في الجدول السابق فهي مشابهة تماما لمثيلاتها على السلاسل النصية والقوائم فلذا لا يوجد داعي لشرحها مرة أخرى ، فقط يكفي أن تتطبقها أنت على مفسر بايثون للتأكد من فهمك لها.

لماذا نستخدم المجموعات؟

أول سؤال يتبادر للمبتدئ لماذا نستخدم المجموعات إذا كان عندنا القوائم؟ قد يكون هذا تاريخيا ولكن أفضل إجابة أن ثبات المجموعات يوفر العديد من مميزات، مثلا يمكن باستخدام المجموعات التأكد أن الكائنات لا تتغير باستخدام مراجع أخرى في مكان آخر في البرنامج.

بعض العمليات المدمجة تحتاج إلى المجموعات وليس القوائم، وبشكل عام استخدم القوائم في المجموعات المنظمة التي يطرأ عليها التغيير أما في بقية الحالات فاستخدم المجموعات.

الملفات

على أمل أن معظم القراء عندهم خلفية عن فكرة أسماء الملفات التي تخزن المعلومات في الكمبيوتر و التي يديرها نظام التشغيل،يكون آخر كائن مدمج في بايثون يزودنا بطريقة الوصول إلى تلك الملفات ضمن برامج بايثون.إن الدالة مدمجة open تنشأ كائن الملف في بايثون وهي تقدم لنا خدمة الربط إلى الملف المستقر على جهازنا،بعد مناداة الدالة open ،يمكننا القراءة والكتابة من الملف المربوط،بمناداة دوال الكائن ملف.

عند مقارنة كائن الملفات بالكائنات الأخرى سنجده غريبا قليلا، لأنه ليس عددا و كائنات متسلسلة أو خرائطية، إنما هو استخدام دوال لمعالجة العمليات الشائعة في الملفات، وهذا الكائن ليس موجودا في اللغات الأخرى ، وإنما يدرس في مجال معالجة الخرج والدخل،وهو ليس مستقلا و إنما يستخدم دوال أخرى لمعالجة الملفات.

الجدول التالي يوضح ملخص لأهم العلميات على الملفات، لفتح الملف يجب مناداة الدالة open و تزويدها بمعاملين الأول اسم الملف مع مساره ،و الثاني طريقة معاملة الملف للقراءة r للكتابة w للكتابة في آخر الملف a ، مع ملاحظة أن كلا المعاملين يجب أن يكونا سلاسل نصية:

العملية	الوصف
output = open('/tmp/spam', 'w')	إنشاء ملف output في نمط الكتابة
input = open('data', 'r')	إنشاء ملف output في نمط القراءة
S = input.read()	اسناد خرج الملف بالكامل إلى سلاسل نصية
S = input.read(N)	قراءة N من البايتات (واحد أو أكثر)
S = input.readline()	قراءة السطر التالي
L = input.readlines()	قراءة خرج الملف ووضعه في قائمة بحيث كل سطر في الملف يساوي عنصر في القائمة
output.write(S)	كتابة S داخل الملف output
output.writelines(L)	كتابة جميع أسطر السلاسلة النصية في قائمة L داخل الملف output
output.close()	إغلاق الملف، بعد إغلاق الملف لا يمكن القراءة منه أو الكتابة عليه ويعطي خطأ عند محاولة ذلك

عند إنشائك للملف يمكنك الكتابة والقراءة منه، وفي كل الحالات بايثون تتعامل مع محتويات الملف كسلاسل نصية ولو كانت أعداد، وكذلك عند الكتابة إلى الملفات فهي تعاملها كسلاسل النصية، الجدول السابق يحوي أهم العمليات و يمكنك مراجعة وثائق بايثون للحصول على كل عمليات الملفات.

عملية إغلاق الملف close، تعمل على إغلاق الاتصال بين البرنامج والملف الخارجي وهي مهم لتحرير مساحة من الذاكرة، ولكن كما تعلم أن بايثون تملك مجمع نفايات يقوم بغلق الاتصال عندما لا تكون في حاجة إليه تلقائي، وعملية إغلاق الملف لاتضر في البرامج الصغيرة مثل سكربتات، ولكن عند العمل على أنظمة كبيرة يجب ألا تتهاون عن غلق الملفات بنفسك وعدم الاعتماد على مجمع النفايات لتضمن أداء جيدا.

العمل على الملفات

هنا مثال بسيط يوضح كيفية العمل على الملفات، أول شيء قمنا بفتح ملف في نمط الكتابة،فيقوم المفسر بالبحث عن الاسم المعطى فإن لم يجده يقوم بإنشاء ملف جديد و يعطيه الاسم الذي أعطيناه إياه، ثم قمنا بالكتابة في الملف المنشئ سطر واحد مع ملاحظة إعطاءه علامة سطر جديد، ثم قمنا بإغلاق الملف، ثم قمنا بفتحه في نمط القراءة وقمنا بقراءة السطر الأول منه،ثم حاولنا قراءة السطر الثاني فأعطنا الناتج فراغ لأنه السطر الثاني فارغ:

```
>>> myfile = open('myfile', 'w')          # open for output (creates)
>>> myfile.write('hello text file\n')      # write a line of text
>>> myfile.close()

>>> myfile = open('myfile', 'r')          # open for input
>>> myfile.readline()                     # read the line back
'hello text file\012'
>>> myfile.readline()                     # empty string: end of file
''
```

وهناك بعض الملاحظات حول الملفات:

-من الإصدار 2.2 لبايثون استبدلت الدالة open بالدالة file فيمكنك استخدام الدالة

file مكان الدالة السابقة، وكذلك يمكنك ا ستخدام الدالة السابقة لأنها تعمل كقناع للدالة الجديدة في الإ صدارات الجديدة

-لاحظ أن قراءة الملف تتم مرة واحدة و الملف عند عرضه مرة ثانية با ستخدام الدالة read لا يتم عرضه ويعطي فراغ، فيجب إعا دة قراءة الملف مرة ثانية(هذا ما لاحظته على الإ صدار 2.3 من بايثون)

الخصائص العامة للكائنات

الآن وبعد أن أنهينا جميع الكائنات المدمجة في بايثون، دعنا نأخذ جولة سريعة عن الخصائص العامة للكائنات المدمجة في باثيون التي تتشارك فيها.

تصنيف الكائنات

الجدول التالي يصنف جميع الأنواع التي رأيناها سابقا:

نوع الكائن	صنفه	قابل للتوسع؟
الأعداد Numbers	عددي	لا
السلاسل النصية Strings	متسلسل	لا
القوائم Lists	متسلسل	نعم
القواميس Dictionaries	تخطيطي	نعم
المجموعات Tuples	متسلسل	لا
الملفات Files	امتدادي	N/A

كما نرى من الجدول فإن السلا سل النصية و القوائم والمجموعات تشترك في أنها متسلسلة،و أن القوائم و القواميس فقط تدعم قابلية التو سع و الإنكماش فقط أما غيرها فلا. الملفات تستخدم دوال للتو سع،فهي ليست قابلة للتو سع بالضبط، صحيح تتو سع حين يتم الكتابة، ولكن ليس بالقيو د التي تفرضها بايثون على الأنواع.

العمومية

لقد رأينا العديد من الكائنات المركبة، وبشكل عام نستطيع أن نقول:

-القوائم والقواميس والمجموعات يمكنها أن تخزن أي نوع من الكائنات

-القوائم والقواميس والمجموعات تدعم التداخل المركب

-القوائم والقواميس تستطيع أن تكبر وتصغر ديناميكيا

بسبب أن هذه الكائنات تدعم التداخل المركب فهي مناسبة جدا للتمثيل المعلومات المركبة في التطبيق، انظر إلى المثال التالي:

```
>>>  L = ['abc', [(1, 2), ([3], 4)], 5]
>>>  L[1]
[(1, 2), ([3], 4)]
>>>  L[1][1]
([3], 4)
>>>  L[1][1][0]
[3]
>>>  L[1][1][0][0]
3
```

قمنا بإنشاء قائمة تحوي على قوائم ومجموعات متداخلة، ثم قم قمنا بالوصول إلى الأعضاء عن طريق المفهرس، لاحظ أن بايثون تبدأ من اليسار إلى اليمين في الوصول إلى موقع الكائن باستخدام المفهرس، ولاحظ كيف تعمقنا في الوصول إلى الكائن المراد في كل مرة حتى وصلنا إلى الكائن المراد بالضبط، ومن هنا تعلم أهمية التداخل في بنى المعطيات وما تقدمه بايثون من أدوات سهلة لتأدية الأغراض.

المراجع المشتركة

لقد قلنا سابقا إننا نخزن مراجع إلى الكائنات وليس نسخة عنها،وعمليا هذا ما تريده في أغلب الأحيان،ولكن أحيانا ينبغى عليك التركيز في هذه النقطة وخاصة إذا كانت هناك مراجع مشتركة فإن أي تغيير في المرجع الأصلي يغيير كل النتائج، على سبيل المثال إذا أنشأنا قائمة X ثم قمنا بإنشاء قائمة أخرى L وضمناها مرجعا إلى القائمة X ثم قمنا إنشاء قاموس D وكذلك ضمنا أحد قيم مدخلاته مرجع إلى القائمة X سيكون المثال على الشكل التالي:

```
>>>  X = [1, 2, 3]
>>>  L = ['a', X, 'b']
>>>  D = {'x':X, 'y':2}
```

في هذه الحالة هناك مرجعين إلى القائمة X، وبما أن القوائم تقبل إسناد قيم جديدة إليها، فانظر ماذا يحدث عند إسناد قيمة جديدة:

```
>>> X[1] = 'surprise'        # changes all three references!
>>> L
['a', [1, 'surprise', 3], 'b']
>>> D
{'x': [1, 'surprise', 3], 'y': 2}
```

فكنا منتبها لهذه النقطة.

المقارنة ،والمساواة و التحقق

جميع كائنات في بايثون تستجيب لعمليات المقارنة وعمليات التحقق فيما بينها، وكأنها أعدا د

وهذا بخلاف كثير من اللغات التي لا تسمح بمثل هذه المقارنات، انظر إلى المثال التالي:

```
>>> L1 = [1, ('a', 3)]        # same value, unique objects
>>> L2 = [1, ('a', 3)]
>>> L1 == L2, L1 is L2        # equivalent?, same object?
(True, False)
```

وهنا اختبرنا علاقة المساواة و علاقة التحقق، وانظر المثال التالي:

```
>>> L1 = [1, ('a', 3)]
>>> L2 = [1, ('a', 2)]
>>> L1 < L2, L1 == L2, L1 > L2    # less, equal, greater: a tuple of results?
(False, False, True)
```

وهنا عدة ملاحظات في المقارنة بين الكائنات المختلفة في بايثون:

-الأعدا د تقارن بمقا ديرها التقريبية

- السلا سل النصية تقارن معجميا، أي حرفا بحرف أي"ac" < "abc" لاحظ أن c أكبر من b في المعجم

- القوائم والمجموعات تقارن كل عنصر مع ما يقابله من اليسار إلى اليمين

-القواميس تقارن أيضا با ستخدام القائمة المخزنة من المفتاح والقيمة

تلميحات مهمة

في هذا القسم من كل فصل سنأخذ تلميحات وحيل تساعدك على فهم بايثون بشكل أعمق مع حل مشاكل قد تواجهك ولا تعرف لها إجابة و أنت مبتدئ في اللغة:

إسناد المراجع المشتركة

قد تكلمنا عن هذا النقطة مسبقا، ونعود نكرر شرح هذه النقطة؛لأن عدم فهمها يؤدي إلى غموض في فهم ما يجري في المراجع المشتركة ضمن برنامجك، على سبيل المثال سنقوم بإنشاء قائمة L ثم نقوم بإنشاء قائمة M نضمنها القائمة L ثم نقوم بإسناد قيمة جديدة في القائمة L ، انظر ماذا يحدث:

```
>>> L = [1, 2, 3]
>>> M = ['X', L, 'Y']      # embed a reference to L
>>> M
['X', [1, 2, 3], 'Y']

>>> L[1] = 0               # changes M too
>>> M
['X', [1, 0, 3], 'Y']
```

تلميح

إن تأثير هذه الجزئية يكون مهما فقط في البرامج الضخمة، وعادة المراجع المشتركة تقوم بما تريده بالضبط، ولكن إذا أردت أن تسند نسخة وليس مرجع فماذا تفعل؟ بكل بساطة أضف نقطتين على الشكل التالي:

```
>>> L = [1, 2, 3]
>>> M = ['X', L[:], 'Y']     # embed a copy of L
>>> L[1] = 0                 # only changes L, not M
>>> L
[1, 0, 3]
>>> M
['X', [1, 2, 3], 'Y']
```

التكرار على مستوى واحد

عندما تكلمنا سابقا عن تكرار السلسلة قلنا أنه عبارة عن إعادة السلسلة عدد من المرات،هذا الأمر صحيح ولكن عندما تكون السلسلة متداخلة تكون النتيجة تختلف عما تريده بالضبط، انظر إلى المثال التالي:

```
>>> L = [4, 5, 6]
>>> X = L * 4          # like [4, 5, 6] + [4, 5, 6] + ...
>>> Y = [L] * 4        # [L] + [L] + ... = [L, L,...]
>>> X
[4, 5, 6, 4, 5, 6, 4, 5, 6, 4, 5, 6]
>>> Y
[[4, 5, 6], [4, 5, 6], [4, 5, 6], [4, 5, 6]]
```

لاحظ الفرق عندما عندما وضعنا القوسين، وهذا الفرق أيضا يتجلى عندما نقوم بإسناد قيمة جديدة إلى القائمة L انظر المثال التالي:

```
>>> L[1] = 0          # impacts Y but not X
>>> X
[4, 5, 6, 4, 5, 6, 4, 5, 6, 4, 5, 6]
>>> Y
[[4, 0, 6], [4, 0, 6], [4, 0, 6], [4, 0, 6]]
```

تلميح

هذه حالة ثانية من متاهات المراجع المشتركة، ولكي تحل هذه المشكلة فقط طبق التلميح السابق،وهذا ينطبق أيضا الجمع و التقطيع فكن منتبها.

الأنواع الثابتة لا يمكن أن تتغير في مكانها

كما مر علينا سابقا أن الأنواع الثابتة لا يمكن تغيير مثل السلسل النصية والمجموعات، ولكن إذا أردت أن تغيرها فتضطر إلى إنشاء نسخة جديدة انظر المثال التالي:

```
T = (1, 2, 3)
T[2] = 4          # error!
T = T[:2] + (4,)    # okay: (1, 2, 4)
```

تلميح

لإنشاء نسخة جديدة ،نقوم بإنشاء كائن جديدة ثم نسند إليه الكائن السابق باستخدام ميزة التقطيع ثم نضيف إليه ما نريد إضافته مثل المثال السابق

الملخص

في هذا الفصل تناولنا العديد من المواضيع التي تتعلق بأنواع الكائنات في بايثون،بدأنا ببنة البرامج في بايثون ثم الأعداد و السلاسل النصية ثم القوائم و القواميس والمجموعات ثم أخير الملفات ثم أخذنا أهم الخصائص العامة للكائنات المدمجة في بايثون ثم قمنا باستعراض أهم المشكلة التي تتعلق بالكائنات في بايثون.

الأمثلة في هذا الفصل تميزت بأنها خصصت لتبيين الأشياء الأساسية، في الفصول القادمة ستكون الأمثلة أكثر واقعية.

الفصل الثالث: التعابير الأساسية

النقاط المهمة:

الاسناد

Print

جملة الأختبار if

الحلقة التكرارية while

الحلقة التكرارية for

تلميحات

المخلص

الآن وبعد أن رأينا الأنواع الأ سا سية للكائنات المدمجة في بايثون في الفصل السابق، سنتحرك في هذا الفصل لشرح أنواع التعابير الأ سا سية. وبكل بساطة التعابير هي عبارة عن أ شيا ء نكتبها تخبر بايثون ما على البرنامج أن يفعله بالضبط.

ولفهم التعابير في بايثون نسترجع ما قلناه في الفصل الثاني لما تكلمنا عن بنية البرنامج في بايثون، وقلنا أن هرمية البرنامج تكون كالتالي:

١- البرنامج يتكون من وحدات

٢- و الوحدات تحتوي على عبارات

٣- و العبارات تنشأ الكائنات و تعالجها.

إذا التعابير هي التي تعالج الكائنات -التي مرت علينا في الفصل السابق-،علاوة على ذلك التعابير هي التي تنشأ الكائنات بوا سطة إ سنا د قيم إليها، وكذلك أيضا تنشأ أنواع جديدة من الكائنات مثل الفصول و الدوال والوحدات.

الجدول التالي يلخص تعابير بايثون،لقد مرت علينا .بعضا منها في الفصل الثاني مثل الإ سنا د و الحذف del ،في هذا الفصل سنأخذ معظم ما ورد في هذا الجدول إلا التعابير التي تحتاج إلى متطلبات أكثر و سنأخذها في الفصول التالية:

التعبير	الدور	مثال
الاسناد	المراجع	curly, moe, larry = 'good', 'bad', 'ugly'
مناداة	الدوال	stdout.write("spam, ham, toast\n")
Print	طباعة الكائنات	print 'The Killer', joke
If/elif/else	عمليات الاختيار	if "python" in text: print text
For/else	التكرار	for x in mylist: print x
While/else	الحلقات العامة	while 1: print 'hello'
Pass	المسؤلية	while 1: pass
Continue	قفز في الحلقات	while 1: if not line: break
Try/except/finally	الاستثناءات	try: action() except: print 'action error'
Raise	الاستثناء	raise endSearch, location
Import, From	الوصول إلى الوحدات	import sys; from sys import stdin
Def, Return	الوسائل	def f(a, b, c=1, *d): return a+b+c+d[0]
Class	إنشاء الكائنات	class subclass: staticData = []
Global	اسم الفضاء	def function(): global x, y; x = 'new'
Del	حذف الأشياء	del data[k]; del data[i:j]; del obj.attr
Exec	تشغيل نصوص الأكواد	exec "import " + modName in gdict, ldict
Assert	تأكيد التنقيحات	assert X > Y

الاسناد

لقد رأينا الا سنا د في التعابير سابقا،وبكل بساطة نقول أنك ستكتب الهدف الذي تريد أن تسند إليه على نا حية الشمال ،والمسند إليه على نا حية اليمين بينهما علامة يساوي =، و الهدف من نا حية الشمال يمكن أن يكون ا سم أو كائن ، أما المسند إليه يمكن أن يكون أي نوع من الكائنات التي مرت علينا.

في معظم الأحيان الإ سنا د عملية بسيطة ، ولكن هناك بعض الخائص يجب أن تضعها في

ذهنك:

الإسناد ينشئ كائنات مرجعية

كما رأيت سابقا،بايثون تخزن المراجع إلى الكائنات في أسماء و بنى معطيات، و دائما تنشئ مراجع إلى الكائنات، بدلا من نسخ المراجع.بسبب ذلك تبدو متغيرات بايثون أقرب ماتكون إلى المؤشرات في لغة السي، أكثر من مخزن معطيات.

الأسماء تنشئ عند أول إسناد

وكما رأينا أيضا أسماء المتغيرات تنشئ في بايثون عند أول عملية إسناد إليها، ولست بحاجة إلى أن تعلن عن الأسماء المتغيرات أولا ثم تسند إليها القيمة،وبعض بنى المعطيات وليس كلها تنشء إدخال جديد فيها بواسطة الإسناد مثل القواميس (راجع جزئية القواميس في الفصل الثاني).

يجب أن تسند الأسماء قبل استخدامها

بالمقابل تظهر بايثون خطأ إذا استخدمت الاسم ولم تسند إليه قيمة بعد و ستظهر المزيد من الاستثناءات إذا حاولت أن تفعل ذلك

الإسناد الضمني:import و from و del و class إلخ..

في هذه الجزئية نحن قد تعودنا على أن الإسناد يتم بالمعامل = ،ولكن الإسناد يحدث في العديد من سياقات في بايثون، على سبيل المثال لقد رأينا جلب الوحدات و الدوال و كذلك الفصول و معاملات الدوال و تعابير الحلقات التكرارية ..إلخ وهذه كلها تعتبر إسناد ضمني، وبما أن الإسناد يعمل نفس العمل أينما ظهر، جميع هذه السياقات ببساطة تسند الاسماء إلى مراجع الكائنات في زمن التنفيذ.

الجدول التالي يبين نكهات التعابير الإسنادية في لغة بايثون:

العلمية	***تفسيرها***
spam = 'Spam'	الصيغة الأساسية والاعتيادية
spam, ham = 'yum', 'YUM'	الاسناد المجموعي (اختياري)
[spam, ham] = ['yum', 'YUM']	الاسناد عن طريق القوائم (اختياري)
spam = ham = 'lunch'	تعدد الأهداف

السطر الأول يظهر الصيغة مشهور وهي إسناد اسم إلى قيمة أو بنى معطيات ، أما الصيغ الباقية فهي هيئات خاصة و سنأخذها بشئ من التفصيل:

الاسناد المجموعي والقوائمي:

السطر الثاني والثالث بينهما علاقة،عندما تستخدم مجموعة أو قائمة في يسار علامة يساوي =تقوم بايثون بعملية مزاوجة من جهة اليمين بحيث تسند كل عنصر من اليسار مع ما يقابله من ناحية اليمين بالترتيب من اليسار إلى اليمين، على سبيل المثال في السطر الثاني العنصر spam أسندت إلية القيمة 'yum' .

الإسناد متعدد الأهداف

في السطر الأخير كان هناك أكثر من هدف،قامت بايثون بإسناد مرجع إلى نفس الكائن إلى جميع الأهداف من ناحية اليسار، في السطر الأخير من الجدول الكائن spam و ham أسندا إليهما نفس القيمة وهي 'lunch'، وهذه النتيجة تعادل ولو أننا أسندنا قيمة في كل مرة إلى هدف واحد.

المثال التالي يوضح أكثر عملية الإسناد المتعدد:

```
>>> nudge = 1
>>> wink = 2
>>> A, B = nudge, wink          # tuples
>>> A, B
(1, 2)
>>> [C, D] = [nudge, wink]      # lists
>>> C, D
(1, 2)
>>> nudge, wink = wink, nudge   # tuples: swaps values
>>> nudge, wink                 # same as T=nudge; nudge=wink;
wink=T
(2, 1)
```

قواعد تسمية المتغيرات:

الآن وبعد أن تكلمنا عن قضية الإسناد في بايثون، نريد أن نتوسع في قضية تسمية المتغيرات التي سنسند إليها القيم ونفهم قواعدها. في بايثون أسماء المتغيرات تنشأ عندما نسند إليها القيمة،ولكن هناك قواعد تحكم اختيار الاسم أيضا، وهي مشابهة لقواعد لغة السي وهي:

-اسم المتغير يجب أن يبدأ بحرف أو شرطة سفلية

اسم المتغير يجب أن يبدأ بحرف أو شرطة سفلية فقط ويمكن بعد ذلك أن يتبعه أي عدد من

الأحرف أو الأعداد أو شرطة سفلية ،على سبيل المثال:

أسماء صحيحة: spam _spam Spam

أسماء غير صحيحة: 1spam

ويجب أن لا يحتوي الاسماء على هذه الأحرف : $@#!

-بايثون حساسة لحالة الأحرف

بايثون حساسة لحالة الحروف مثل السي بالضبط فالمتغير omlx يختلف عن المتغير Omlx فكن منتبها لهذه النقطة

-الكلمات المحجوزة

هناك كلمات معينة في لغة بايثون محجوزة ولا يمكن أن تسمى متغيرك بها، وإذا سميت بها متغيرك تعطيك اللغة تحذيرا ،وإذا اضطررت إلى أن تسمي بها فغير حالة الحروف فقط أو حرف معين مثلا class لا يمكنك التسمية بها ولكن Class أو klass يمكنك،

والكلمات المحجوزة موضحة في الجدول التالي:

and	assert	break	class	continue
def	del	elif	else	except
exec	finally	for	from	global
if	import	in	is	lambda
not	or	pass	print	raise
return	try	while		

قبل أن نتحرك عن هذه النقطة، نريد أن ننوه بالفرق بين الأسماء و الكائنات في بايثون، كما رأينا سابقا في الفصل الثاني أنواع الكائنات في بايثون ورأينا أن هناك كائنات ثابتة و أخرى غير ثابتة.

أما الأسماء من ناحية أخرى فهي فقط مجرد مراجع تؤشر إلى الكائنات ولا ترتبط بنوع الكائنات بل تستطيع أن تؤشر إلى أي نوع من الكائنات وبنفس الاسم فهي غير ثابتة ،انظر المثال التالي:

```
>>> x = 0          # x bound to an integer object
>>> x = "Hello"    # now it's a string
>>> x = [1, 2, 3]  # and now it's a list
```

وكما نرى فإن المثال الأخير يوضح لنا مميزات الأسماء في بايثون بشكل عام.

Print

التعبير print بكل بساطة هو التعبير الذي يطبع الكائنات،من الناحية التقنية يقوم هذا التعبير بكتابة التمثيل النصي للكائن ويرسله إلى الخرج القياسي للبرنامج. والخرج القياسي غالبا ما يكون النافذة التي بدأ تنفيذ برنامج بايثون بها، إلا إذا تم إرسال النتائج الخرج إلى ملف باستخدام أوامر الشل.

في الفصل الثاني رأينا دوال الكائن الملف التي تكتب إلى الملف write ، التعبير print مشابهة إليها ولكن بتركيز أكثر: التعبير print يقوم بكتابة الكائنات إلى الخرج القياسي stdout، أما الدالة write تقوم بكتابة السلاسل النصية إلى الملف،ومنذ أن توفر الخرج القياسي في بايثون ككائن stdout في وحدة sys يمكنك أن تحاكي التعبير print باستخدام دالة كتابة الملفات write (انظر الأمثلة التالية) ولكن استخدام print أسهل بكثير.

الجدول التالي يوضح صيغ التعبير print:

العملية	***تفسيرها***
print spam, ham	طباعة الكائنات إلى sys.stdout وإضافة بينهما فراغ
print spam, ham,	نفس السابق ولكن بدون إضافة سطر جديد في النهاية

بشكل افتراضي يقوم التعبير print بإضافة فراغ بين الكائنات التي تفصل بينهما فاصلة مع إضافة علامة نهاية السطر في نهاية السطر من الخرج.لتجاوز علامة نهاية السطر (وبالتالي يمكنك إضافة نصوص أخرى في نفس السطر لاحقا) أنه تعبير print بإضافة فاصلة ،مثلما يظهر في السطر الثاني من الجدول السابق