Programmation pour le traitement automatique des langues : le langage Python

Guy Perrier

1 - Premiers pas avec Python

1. Présentation de Python
2. Expressions et types
3. Variables et instructions d’affectation
4. Scripts Python

1.1 - Présentation de Python

- Python a été créé au début des années 1990 par **Guido van Rossum**.
- Python est un langage de programmation distribué sous **licence libre** et utilisable sur toutes les plateformes courantes (Windows, Linux, Mac).
- La version courante est la version 2.6.1 et une nouvelle version 3 est en disponible mais cette version n'est plus compatible avec les versions précédentes.
- Pour en savoir plus sur Python :
  - Le site officiel en anglais: http://www.python.org/
  - Un manuel d'apprentissage de Python en français :

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

1.1 - Présentation de Python

- **Python** est un **langage interprété** par opposition aux langages compilés.

- Un programme dans un langage interprété est lu pas à pas par un programme spécial dépendant du langage et de la plate-forme sur laquelle il s'exécute : **l'interprète**. L'interprète exécute le programme au fur et à mesure de sa lecture.

- **Un programme source** dans un langage compilé est traduit dans un langage de plus bas niveau par un programme spécial dépendant du langage et de la plateforme sur laquelle il s'exécute : le **compilateur.** C'est ensuite le programme compilé ou **programme objet** qui est exécuté.

1.1 - Présentation de Python

- Python peut être exécuté en **mode interactif** ou en **mode automatique**. En mode interactif, chaque ligne est exécutée dès qu'elle est entrée au clavier. En mode automatique, un programme, appelé aussi **script**, est stocké dans un fichier (reconnu par l'extension .py) et il est ensuite exécuté d'un seul coup.

- **Idle** est une interface graphique qui facilite l'utilisation de Python.

1.2 - Expressions et types

- Une **expression** est un objet qui peut être évalué : on peut calculer sa **valeur**. Une expression possède aussi un **type**.
- Une expression peut être **atomique**. Sa valeur se confond alors en général avec elle.

```
>>> type(2)
<type 'int'>

>>>type(100000000000000000)
<type 'long'>

>>> type(2.34)
<type 'float'>

>>> type(2.4E5)
<type 'float'>
```

```
>>> type('bonjour')
<type 'str'>

>>> type("bonjour")
<type 'str'>

>>> type(False)
<type 'bool'>

>>> type(True)
<type 'bool'>
```

1.2 - Expressions et types

- Une expression peut être **composée** à partir d'autres expressions et d'**opérateurs** portant sur ces expressions. Sa valeur est calculée à partir des valeurs des expressions composantes et du sens des opérateurs.

```
>>> "la " + "table"
'la table'

>>> 4.5*4
18.0

>>> True and False
False

>>> 2>3
False

>>> range(2)
[0,1]
```

Appel de fonction

1.2 - Expressions et types

- Lorsque une expression comporte plusieurs opérateurs, des **règles de priorité** indique l'ordre dans lequel les opérations sont effectuées. Les **parenthèses** permettent de modifier cet ordre.

```
>>> 2.5+3/4
2.5

>>> (2.5+3)/4
1.375

>>> False or True and False or True
True

>>> 'a' == 'b' or False
False

>>> 3 in range(3)
False
```

1.2 - Expressions et types

Opérateur	Description
lambda	Lambda expression
or	Booléen OU
and	Booléen ET
not	Booléen NON
in, not in	Tests d'appartenance
is, is not	Tests d'identité
<, <=, >, >=, <>, !=, ==	Comparaisons
|	OR bit à bit
^	XOR bit à bit
&	AND bit à bit
<<, >>	Décalages de bits
+,-	Addition et soustraction
*, /, %	Multiplication, division, reste
+, -	Positif, opposé
~	NOT bit à bit
**	Puissance
id. attribute	Référence d'un attribut
id [*index*]	Accès à une entrée d'un tableau
id [*index : index*]	Accès à une tranche d'un tableau
id (arguments)	Appel de fonction

1.2 - Expressions et types

- Pour un opérateur donné, les types des opérandes doivent être dans les types possibles attendus par l'opérateur.

```
>>> "table"- "ta"
Traceback (most recent call last):
  File "<pyshell#60>", line 1, in <module>
    "table"- "ta"
TypeError: unsupported operand type(s) for -: 'str' and 'str'

>>> "table" + 2
Traceback (most recent call last):
  File "<pyshell#61>", line 1, in <module>
    "table" + 2
TypeError: cannot concatenate 'str' and 'int' objects
```

- Une utilisation non standard d'un opérateur avec certains types d'opérandes est parfois possible après une **conversion de type** des opérandes.

```
>>> True + 12
13
>>> type(True+12)
<type 'int'>

>>> 12 + 3.0/4
12.75
>>> type(12 + 3.0/4)
<type 'float'>
```

1.3 - Variables et instructions d'affectation

- Une **variable** est une zone mémoire désignée par un **identificateur** permettant de stocker une **valeur**.

- Le stockage d'une valeur se fait par le biais d'une **instruction d'affectation** qui a la forme :

  variable = expression

- Dans une affectation, la variable prend le type de l'expression utilisée jusqu'à une nouvelle affectation la concernant (**typage dynamique**).

- Les variables peuvent être utilisées pour former des expressions.

- Une instruction d'affectation permet de modifier la valeur d'une variable en utilisant l'ancienne valeur pour calculer la nouvelle. On peut alors utiliser une instruction de la forme :

  variable op = expression qui est une abréviation de : *variable = variable op expression*

1.3 - Variables et instructions d’affectation

```
>>> x = 2==3
>>> type(x)
<type 'bool'>
>>> x
False

>>> x=3*x
>>> x
0
>>> type(x)
<type 'int'>

>>> x="la"
>>> type(x)
<type 'str'>
>>> x
‘la’

>>> x=3*x
>>> x
'lalala’

>>> x+= ‘ boule’
>>> x
‘lalala boule’
```

1.4 - Scripts Python

- En mode automatique, un programme Python appelé aussi **script**, doit être stocké dans un fichier (avec l'extension .py). L'intérêt par rapport au mode interactif est que cela permet de conserver le fichier pour le réutiliser en le modifiant éventuellement.

- Un script est exécuté soit par le biais d'une interface graphique comme Idle, soit en ligne de commandes à l'aide de la commande python suivie du nom du fichier (avec le chemin d'accès).

1.4 - Scripts Python

- Un script peut être écrit à l'aide d'un éditeur standard. En entête du script, on peut préciser le codage des caractères utilisé à l'aide d'un commentaire. Par exemple, pour le codage Latin-1, il faut mettre : *# -*- coding:Latin-1 -*-* Un codage de plus en plus utilisé est UTF-8. Il faut alors écrire en entête : *# -*- coding:Utf-8 -*-*

- Un script est en général composé d'une séquence d'instructions. Pour sa compréhension, il est important d'y insérer des **commentaires**. Un commentaire commence par le caractère # et termine par un passage à la ligne. L'interprète de Python ignore les commentaires.

1.4 - Scripts Python

- Exemple de script Python stocké dans un fichier *aire.py*.

```
# Programme de calcul de l'aire d'un rectangle : aire = longueur x largeur



longueur = input("Entrez une longueur en cm :  ") # entrée de la longueur en cm au clavier
largeur = input("Entrez une largeur en cm :  ") #entrée de la largeur en cm au clavier
aire = longueur * largeur #calcul de l'aire en cm2
print "Aire du rectangle : ", aire, " cm2" # affichage de l'aire
```

- Une exécution possible du script *aire.py*.

```
>>>
Entrez une longueur en cm :  8.5
Entrez une largeur en cm :  4
Aire du rectangle :  34.0  cm2
>>>
```

1.5 Exercices

1. Déterminer si chacune des expressions suivante est bien formée. En cas de réponse positive, déterminer son type et sa valeur.

   a) *3 + 4 ** 2 * 2 / 3*

   b) *3 * ‘a’ + ‘b’ + 2 * “c”*

   c) *2==3 + “bonjour”*

   d) *2 >= 2 and not 3 >2*

   e) *“a” < “b” or 2==3 and 0 in range(2)*

2. On suppose que l’on exécute dans l’ordre les instructions d’affectation ci-dessous. Indiquer pour chacune d’elles, la modification du contenu de la variable qu’elle entraîne.

```
x  = 2/3 - 3**2
y  = 2.5 + 5 % 2
y -= x
z = y <= x+9
z *=3
x = ‘bon’ + ‘jour’ > ‘bon’ + (z + 2) * ‘jo’
```