

# Le langage Python

Arnaud Fontaine

11/03/2006



# Plan

- 1 Présentation de Python
- 2 Syntaxe
- 3 Types natifs
- 4 Contrôle de flux
- 5 Fonctions
- 6 Modules



# Plan

- 1 **Présentation de Python**
- 2 Syntaxe
- 3 Types natifs
- 4 Contrôle de flux
- 5 Fonctions
- 6 Modules



## Caractéristiques de Python

- Langage interprété de haut niveau
- Multi-plateforme
- Orienté objet
- Typage dynamique
- Extensibilité (C, C++...)
- Rapidité de développement



## Domaines d'application

- Scripts d'administration
- GUI (Gtk, wxWindows...)
- Sites internet (Cherryypy, Zope, Plone...)
- Jeux
- Multimédia
- ...



## Script

```
print "Hello World !"  
$ python helloworld.py  
Hello World !
```

## L'interpréteur

```
$ python  
Python 2.3.5 (#2, Mar 6 2006, 10:12:24)  
[GCC 4.0.3 20060304 (prerelease) (Debian 4.0.2-10)] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>> print "Hello World"  
Hello World  
>>>
```

Généralement, on utilisera la ligne de commande à des fins de tests et les scripts pour automatiser un ensemble de tâches



# Plan

- 1 Présentation de Python
- 2 Syntaxe**
- 3 Types natifs
- 4 Contrôle de flux
- 5 Fonctions
- 6 Modules



Python dispose d'une syntaxe élégante :

- blocs se définissant par l'indentation
- commentaires commençant par un “#”
- opérateurs intuitifs (and, or, not, in...)
- les scripts commencent par une directive particulière

```
#!/usr/bin/python
# commentaire1
msg = "toto"
if level == 'admin':
    print "Hello master !" # commentaire2
else:
    print "Hello"
print msg
```



# Plan

- 1 Présentation de Python
- 2 Syntaxe
- 3 Types natifs**
- 4 Contrôle de flux
- 5 Fonctions
- 6 Modules



## Nombres : non modifiables

### Opérateurs

`+`, `*`, `/`, `round ()`, `abs ()`, `pow ()`, ...

```
>>> foo = 2
>>> bar = 3
>>> foo * bar
6
>>> foobar = _ + 2
8
>>> foobar / 3
2
>>> foobar / 3.0
2.6666666666666665
>>>
>>> float (foobar) / 3
2.6666666666666665
>>>
```



## Chaînes de caractères : tableaux indexés non modifiables

### Opérateurs

+ (concaténation), \* (répétition), len (), min (), max (), ...

```
>>> foo = 'mooh'
>>> foo + 'bar'
'moohbar'
>>> foo * 3
'moohmoohmooh'
>>> "foo's bar"
"foo's bar"
>>> 'foo\'s bar'
"foo's bar"
>>> print """
... ceci est un exemple
... de chaine sur plusieurs lignes
... """
ceci est un exemple
de chaine sur plusieurs lignes
>>> mooh = "python et les chaines"
>>> mooh[1]
'y'
>>> mooh[2:]
'thon et les chaines'
>>> mooh[2:5]
'tho'
>>> 'm' in mooh
False
>>> print "x vaut %d\n" % 10 # chaine formatée
x vaut 10
```



## Listes : tableaux indexés modifiables

### Opérateurs

`liste.append ()`, `liste.join ()`, `liste.split ()`, `liste.remove ()`, `del`,  
`liste.sort ()`...

```
>>> liste = ['foo',10,10.0]
>>> print liste
['foo', 10, 10.0]
>>> liste.append ('bar') # ajout d'un élément
>>> print liste
['foo', 10, 10.0, 'bar']
>>> liste.remove ('foo') # suppression du premier élément trouvé
>>> print liste
[10, 10.0, 'bar']
>>> del liste[1] # suppression indexée
>>> print liste
[10, 'bar']
>>> l = ['bonjour','foo','bar']
>>> s = ' '.join (l)
>>> print s.capitalize ()
Bonjour foo bar
>>> del liste # suppression de la liste
```



## Tuples : tableaux indexés non modifiables

### Opérateurs

### Opérateurs équivalents à ceux des chaînes

```
>>> ('foo',2,'bar',4)
>>> 'foo',2,'bar',4 # parenthèse optionnelle
('foo', 2, 'bar', 4)
>>> mooh = 'foo',2,('bar',4) # tuples de tuples
('foo', 2, ('bar', 4))
>>> print mooh[2]
('bar', 4)
>>> print mooh[2][0]
bar
>>> print mooh[1:]
(2, ('bar', 4))
```



## Dictionnaires : tableaux indexé par **clé** modifiables

### Opérateurs

del, len (), dico.has\_key (), dico.keys (), dico.clear (), dict (), ...

```
>>> dict (foo='ex',bar='ex2')
{'foo': 'ex', 'bar': 'ex2'}
>>> {foo: 'ex',bar='ex2',1:'ex3'} # équivalent à la syntaxe précédente
{1: 'ex3', 'foo': 'ex', 'bar': 'ex2'}
>>> test = {foo: 'ex',bar='ex2',1:'ex3'}
>>> test.keys ()
[1, 'foo', 'bar']
>>> test.values ()
['ex3', 'ex', 'ex2']
>>> test.has_key (1)
True
```



## Opérateurs

`open ()`, `fichier.read (N)`, `fichier.readline ()`, `fichier.write ()`,  
`fichier.close ()`, ...

```
# fichier 'fich.txt'
Introduction à Python lors
de la réunion des associations
Tarentux et Albertville Wireless
le 11 mars 2006 à Albertville
>>> fich = open ('/tmp/fich.txt','r')
>>> fich.readline ()
'Introduction a Python lors\n'
>>> fich.read (10)
'de la reun'
>>> fich_out = open ('fich_out.txt','w')
>>> fich_out = fich_out.write (fich.readline ())
>>> fich.close ()
```



# Plan

- 1 Présentation de Python
- 2 Syntaxe
- 3 Types natifs
- 4 Contrôle de flux**
- 5 Fonctions
- 6 Modules



## If, elif et else

```
#!/usr/bin/python
x = int (raw_input ("Entier: ")) # entier demandé à l'utilisateur
if x == 0:
    print "Nombre nul"
elif x < 0:
    print "Nombre négatif"
elif x > 0:
    print "Nombre positif"
else:
    print "Valeur invalide"
```

## Caractéristiques de la structure

- elif présent 0 ou plusieurs fois
- comparable au *switch* d'autres langages

## Les blocs

Notez le caractère “ : ” qui permet de commencer un bloc, il doit être placé à la fin de chaque instruction de contrôle de flux. L'instruction elif peut être présente 0 ou plusieurs fois.



## while

```
>>> a, b = 0, 1
>>> while b < 10:
...     print b
...     a, b = b, a + b
1
1
2
3
4
5
6
7
8
9
10
```

La syntaxe du while est très proche de celle des autres langages tels que le C. Les opérateurs de comparaison disponibles sont : <, <=, ==, >, >=, !=



## for et range

```
>>> range(1, 10)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> a = 0
>>> for i in range(1, 10):
...     a = a + i
>>> print a
45
>>> dico = {'Tarentux': 'GULL', 'AW': 'Wifi'}
>>> for i in dico.keys():
...     print dico[i]
...
GULL
Wifi
>>> l = ["lapin", "gnou", "pingouin"]
>>> l = [word for word in l if not 'g' in word]
>>> l
['lapin']
>>> f = open('/etc/resolv.conf')
>>> for line in f:
...     print len(line), line
11 search lan
23 nameserver 192.168.2.1
```

## Caractéristiques

- L'instruction for s'incrémente par rapport au contenu d'une séquence (différent du C)
- range () permet des itérations sur une séquence de nombres (comparable au C)



## break, continue et pass

```
>>> while True:
...     pass # Ne fait rien
...
>>> c = 100
>>> while True:
...     if c == 0:
...         break # permet de sortir de la boucle actuelle
...     c = c / 2
...     print c,
...
50 25 12 6 3 1 0
# l'instruction continue procède à l'itération suivante
```

Afin d'augmenter la lisibilité, il est conseillé de recourir aux `break` et `continue` avec parcimonie. L'instruction `pass` est utilisée dans les cas où le corps de la boucle est vide.



# Plan

- 1 Présentation de Python
- 2 Syntaxe
- 3 Types natifs
- 4 Contrôle de flux
- 5 Fonctions**
- 6 Modules



## def

```
>>> import random
>>> from string import ascii_lowercase, digits
>>> def rand_passwd ():
...     """Generate a random password for Radius accounts"""
...     return ''.join ([random.choice (PASSWD_RAND_CHAR) for i in range (PASSWD_LEN)])
>>> rand_passwd ()
'kqql6nxh'
>>> rand_passwd
<function rand_passwd at 0xb7dc4b54>
>>> def foo (nb=None,str='lapin'):
...     print nb,str,
...
>>> foo ()
None lapin
>>> foo (6,lapins)
6 lapins
>>> foo (str='gnous')
None gnous
```

Une fonction est un objet. On peut définir des arguments ayant une valeur par défaut. None est retourné dans le cas d'une procédure.



# Plan

- 1 Présentation de Python
- 2 Syntaxe
- 3 Types natifs
- 4 Contrôle de flux
- 5 Fonctions
- 6 Modules**



## Qu'est ce qu'un module ?

Il est généralement plus pratique de diviser son code en fichiers distinctes, un module correspond au script python appelé depuis le programme principal.

## import, from, dir ()

```
>>> dir () # namespace courant
['__builtin__', '__doc__', '__name__']
>>> a = 10
>>> dir ()
['__builtin__', '__doc__', '__name__', 'a']
>>> import re
>>> dir ()
['__builtin__', '__doc__', '__name__', 'a', 're']
>>> re.search
<function search at 0xb7d9e7d4>
>>> from re import search
>>> dir ()
['__builtin__', '__doc__', '__name__', 'a', 're', 'search']
```

## Chemins ...

Notez qu'un import sur un module que vous avez créé ne cherche que dans le répertoire courant, vous pouvez faire ainsi pour ajouter un répertoire :

```
import sys
sys.path.insert (0, 'dir')
```

