Introduction à Python

Olivier Richard <olivier.richard@imag.fr>,
Université Joseph Fourier

Jeudi 3/05/2013

Outline

Sources utilisées

- Tutoriel officiel : http ://docs.python.org/2/tutorial/index.html
- http ://www.korokithakis.net/tutorials/python/
- http ://hebergement.u-psud.fr/iut-orsay/Pedagogie/MPHY/Python/courspython3.pdf

Python : en quelques points

- Version initiale créé en **1989** par *Guido van Rossum*
- Un langage de script
- **Interprété** (*bytecode compilé*)
- **Typage Dynamique**
- **Indentation significative**
- **Orientation objet**
- Gestion automatique de la mémoire (*garbage collector*)

http ://fr.wikipedia.org/wiki/Python_(langage)

Interpréteur de commande

```
auguste@akira:/tmp$ python
Python 2.7.4rc1 (default, Mar 30 2013, 15:39:28)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more infor
>>>
>>>5*(3+2)
25
```

- Pour sortir de l'interpréteur faire `Crtl-D` ou `exit()`
- Documentation : **help(<object>)**
- Autre usage : *python nom_script*

Programme simple

```
#!/usr/bin/python

# import modules used here -- sys is a very standard one
import sys

# Gather our code in a main() function
def main():
    print 'Hello there', sys.argv[1]
    # Command line args are in sys.argv[1], sys.argv[2] ...
    # sys.argv[0] is the script name itself and can be ignored

# Standard boilerplate to call the main() function to begin
# the program.
if __name__ == '__main__':
    main()
```

Variables et type simple

Type	Description	Exemple
int	32 bits	4, -6
long	64 bits	140L
float	nb flottant	4.5
complex	nb complexe	3.4j
bool	booléen	True, False
str	chaîne carac.	« yop »
list	liste	[1, 2, 3]

```
a = 4
if type(a) == int:
  print "c'est un entier"

b = True
if b:
  print "yop"

x = y = z = 0  #affectation multiple
n, m = 10, 20  #affectation multiple
```

Les chaînes de caractères

- Les chaînes de caractères sont **non modifiable** (immutable)
- Opérations sur les chaînes par *fonction* ou *méthode*

```
a = "yop"
len(a) # 3
a.upper() # YOP (nouvelle chaine)
```

- Tranche (slice) : **[indice1 :indice2]**

```
>>> word = 'Help' + 'A'
>>> word[4] #
'A'
>>> word[0:2]
'He'
>>> word[2:4]
'lp'
```

- testez [:], [:-1], [2 :]

Les entrées / sorties

- la saisie clavier :

```
nb = input("Entrez un nombre : ")
print nb
```

- la sortie :

```
a,b = 2,6
s = a + " * " + b + " = " + a*b
print s
```

- lecture de fichier

```
for line in open("toto.txt"):
    print line
```

- écriture de fichier

```
f = open('titi', 'w')
f.write("hello")
```

Contrôle de flot d'instructions

Contrôle de flot

- **if - elif - else**
- **for**

```
words = ['cat', 'window', 'defenestrate']
for w in words:
   print w, len(w)

cat 3
window 6
defenestrate 12
```

- **while**
- **pass** opération nop
- Note : **range**

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Defintion de fonction

```
def fib(n):    # write Fibonacci series up to n
    """Print a Fibonacci series up to n."""
    a, b = 0, 1
    while a < n:
        print a,
        a, b = b, a+b
```

Notes :

- ligne vide pour terminer la fonction
- *docstring* accessible par *nom_de_fct*.__doc__

Fonction : argument par défaut

```
def ask_ok(prompt, retries=4, complaint='Yes or no, please!'):
    while True:
        ok = raw_input(prompt)
        if ok in ('y', 'ye', 'yes'):
            return True
        if ok in ('n', 'no', 'nop', 'nope'):
            return False
        retries = retries - 1
        if retries < 0:
            raise IOError('refusenik user')
        print complaint
```

Fonction : Nombre d'argument arbitraire

```
def a(*args):
    for val in args:
        print val

>>> a("e",1,5,"t")
```

Les séquences : *Chaînes*, *Listes* et *Tuples*

Listes

- Les listes sont **modifiable** (*mutable*).

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam', 'eggs', 100, 1234]
>>> a[0] # index debute a zero
'spam'
>>> a[3]
1234
>>> a[-2]
100
```

Listes : Tranche (*slice*)

```
>>> a[1:-1]
['eggs', 100]
>>> a[:2] + ['bacon', 2*2]
['spam', 'eggs', 'bacon', 4]
>>> 3*a[:3] + ['Boo!']
['spam', 'eggs', 100, 'spam', 'eggs', 100, 'spam', 'eggs', 100
```

Listes : Fonctions

- *list.nom_fonction* : **append, extend, insert, remove, pop, index, count, sort, reverse**

```
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print a.count(333), a.count(66.25), a.count('x')
2 1 0
>>> a.insert(2, -1)
>>> a.append(333)
>>> a
[66.25, 333, -1, 333, 1, 1234.5, 333]
>>> a.index(333)
1
>>> a.remove(333)
>>> a
[66.25, -1, 333, 1, 1234.5, 333]
>>> a.reverse()
>>> a
[333, 1234.5, 1, 333, -1, 66.25]
>>> a.sort()
>>> a
[-1, 1, 66.25, 333, 333, 1234.5]
```

-autre **del**, utilisation de list comme queue **from collections import deque**

Tuples

Les tuples sont **non-modifiable** (*immutable*).

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
>>> # Tuples are immutable:
... t[0] = 88888
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> # but they can contain mutable objects:
... v = ([1, 2, 3], [3, 2, 1])
>>> v
([1, 2, 3], [3, 2, 1])
```

Autres types de : *sets* et *dictionairies*

Sets

- Les *sets* sont des ensembles non-ordonnés et sans duplicat d'élément.
- Opérations disponibles : test d'appartenance, opérations sur ensemble

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> fruit = set(basket)               # create a set without duplicates
>>> fruit
set(['orange', 'pear', 'apple', 'banana'])
>>> 'orange' in fruit                 # fast membership testing
True

>>> # Demonstrate set operations on unique letters from two words
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                                 # unique letters in a
set(['a', 'r', 'b', 'c', 'd'])
>>> a - b                             # letters in a but not in b
set(['r', 'd', 'b'])
>>> a | b                             # letters in either a or b
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
>>> a & b                             # letters in both a and b
set(['a', 'c'])
>>> a ^ b                             # letters in a or b but not both
set(['r', 'd', 'b', 'm', 'z', 'l'])
```

Dictionaries

- Les dictionnaires sont des tableaux associatifs (*associative arrays*), ils associent une **clé** à une **valeur**
- Opérations : appartenance (**in**), taille (**len**), rétrait de clé/valeur **del**

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}
>>> tel.keys()
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
```

Disctionnaire et fonction

```
def cheeseshop(kind, *arguments, **keywords):
    print "-- Do you have any", kind, "?"
    print "-- I'm sorry, we're all out of", kind
    for arg in arguments:
        print arg
    print "-" * 40
    keys = sorted(keywords.keys())
    for kw in keys:
        print kw, ":", keywords[kw]

cheeseshop("Limburger", "It's very runny, sir.",
           "It's really very, VERY runny, sir.",
           shopkeeper='Michael Palin',
           client="John Cleese",
           sketch="Cheese Shop Sketch")
```

Lambda

Fonction annonyme

```
>>> def make_incrementor(n):
...     return lambda x: x + n
...
>>> f = make_incrementor(42)
>>> f(0)
42
>>> f(1)
43
```

Programmation fonctionnelle : compréhensions de liste

```
l =  [x**2 for x in range(10)]
print l # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

set(l) # pour creer un set
tuple(l) # pour creer un tuple

d = {x: x**2 for x in (2, 4, 6)}
print d # {2: 4, 4: 16, 6: 36}
```

Programmation fonctionnelle : *map(), filter()*

```
def premierCaractere(s):
    return s[0]

>>>map(premierCaractere, ["hello", "world"])
['h', 'w']

>>> [premierCaractere(x) for x in ["hello", "world"]]
['h', 'w']

>>>filter(lambda x: x > 0, [1, -2, 3, -4])
[1, 3]

>>>[x for x in [1, -2, 3, -4] if x > 0]
[1, 3]
```

Programmation Objet

- Classes : une forme de type de données (données + méthodes)
- Objets : un instance particulière d'une classe
- Encapsulation : le fait qu'un objet est composé d'attributs (données) et de méthodes (fonctions)
- Héritage : une classe fille peut hériter des attributs (données + méthodes)
- Surcharge : redéfinition d'une méthode
- Polymorphisme : un même nom de méthode pour des objets de type différent

Une simple classe

```
class Animal:
    pass # Un bloc vide

a = Animal()

>>>print(a)

<__main__.Animal instance at 0x1420950>
```

Méthode et Self

```
class Animal:
    def cri(self):
        print('Grrr')

a = Animal()
a.cri()
```

```
Grrr
```

Méthode spéciale : __init__

- La méthode __init__ est automatiquement invoquée lors de l'instanciation d'un objet

```
class Animal:
    def __init__(self, n):
        self.nb_pattes = n

a = Animal(4)
print(a.nb_pattes)
```

```
4
```

Autres méthodes spéciales

Pour la surcharge d'opérateur

Méthodes spéciales	Opérations
__neg__	-a
__add__	a+b
__mul__	a*b
__sub__	a-b
__div__	a/b
__floordiv__	a//b

Classe : réference et variable

```
class MyClass(object):
    common = 10
    def __init__(self):
        self.myvariable = 3
    def myfunction(self, arg1, arg2):
        return self.myvariable
>>> classinstance = MyClass()
>>> classinstance.myfunction(1, 2)
??
>>> classinstance2 = MyClass()
>>> classinstance.common
??
>>> classinstance2.common
??
>>> MyClass.common = 30
>>> classinstance.common
??
>>> classinstance2.common
??
>>> classinstance.common = 10
>>> classinstance.common
??
>>> classinstance2.common
??
>>> MyClass.common = 50
>>> classinstance.common
??
>>> classinstance2.common
??
```

Le style de codage

- La référence le PEP8

Extraits

- Indentation **4 espaces**, ne pas mélanger tabulations et espaces
- Insérer une ligne de séparation à la fin d'une fonction, d'une méthode et de grand bloc d'instruction
- Ligne de 79 caractères au maximum (sinon \ et retour à la ligne)
- Les imports devraient être faits sur des lignes séparées
- Espace dans les expressions et les déclarations
- Espace autour de opérateurs et de égal : **a = f(1, 2) + g(3, 4)**
- Toujours utiliser **self** comme premier argument d'une méthode
- Pas de parenthèses autour des conditions des **if** et des **while**

Exception

```
def some_function():
    try:
        # Division by zero raises an exception
        10 / 0
    except ZeroDivisionError:
        print "Oops, invalid."
    else:
        # Exception didn't occur, we're good.
        pass
    finally:
        # This is executed after the code block is run
        # and all exceptions have been handled, even
        # if a new exception is raised while handling.
        print "We're done with that."

>>> some_function()
```

Import et *From*

```
>>> import random
>>> random.randint(10, 20)
12
>>> randint(10, 20)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'randint' is not defined

>>> from random import randint
>>> randint(10, 20)
```

Quelques Bibliothèques

La bibliothèque standard contient plus de 200 packages et modules (approche *"batteries included"*)

Expressions rationnelles

Regular expression operations

```
import re
pattern = re.compile("d")
>>> pattern.search("dog")      # Match at index 0
<_sre.SRE_Match object at ...>
>>> pattern.search("dog", 1)   # No match; search doesn't inclu

>>>pattern.search(r"dog")
```

Version 2.7 vs 3.0

Pourquoi s'intéresser aux versions 2.7.x ?

- Certains modules et bibliothèques ne sont pas encore disponibles en 2.7
- Pour débuter être en 2.7 est suffisant
- A moyen terme la version 3.x sera la référence
- What's New In Python 3.0

Liens utiles

- Tutoriel officiel : http ://docs.python.org/2/tutorial/index.html
- Antiséches : http ://www.cheat-sheets.org/saved-copy/PQRC-2.4-A4-latest.pdf
- Documents divers pour enseignant CPGE http ://goo.gl/AVV5t.
- Beginners' Guide
- Cours sur Python 3 de Bob CORDEAU.

Conclusion

- Python un représentant important des langages scripts dynamiques
- **Rapidité de développement**
- **Un très grand nombre de bibliothèques disponibles**