Python par l'exemple

Nicolas Duboc
<nduboc@debian.org>

Au programme ...

- Présentation et historique
- Un peu de philosophie
- Principales caractéristiques
- Particularités
- Python pour quoi faire ?
- Quelques défauts
- Conclusion

Présentation et historique

Python est un langage de programmation dynamique, généraliste, orienté objet et bien équipé.

Initialement créé par **Guido Van Rossum** au début des années 90. Développé par une importante communauté autour du Dictateur Bénévole à Vie (BDFL), Guido.

Plusieurs implémentations existent :

- **cPython** : implémentation principale et historique en C (v 2.4.3)
  travail commencé sur la version 3000
- **Jython** : implémentation en Java et intégrée à la plateforme Java
  projet peu actif mais toujours en vie
- **IronPython** : implémentation en C# et intégrée à la plateforme .NET
  projet très actif. Développement par Microsoft
- **PyPy** : implémentation en Python
  travail de recherche

Un peu de philosophie pythonienne

```
~% python
Python 2.3.5 (#2, Aug 30 2005, 15:50:26)
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

Principales caractéristiques : 1. syntaxe simple et claire

```
gurus = ['guido', 'james', 'bjarne']❶

def hello_guru(name):❷
❸   "Return an hello message for 'name'"
    name = name.capitalize()
    return 'Hello, guru %s' % name

for name in gurus :❹
    print hello_guru(name)
```

1 : structures de données intégrées au langage : listes, dictionnaires, ensembles
2 : définition de fonction
3 : indentation prise en compte par la syntaxe
4 : seules deux structures de boucles : for et while

Principales caractérisques : 2. langage objet

Tout est objet :

```
"spameggs".capitalize()

def spam(i):
    return 2*i
type(spam)     =>     <type 'function'>
```

même si la syntaxe laisse penser le contraire parfois :

```
1.__str__()    =>     SyntaxError
a = 1
a.__str__()    =>     '1'
```

Principales caractérisques : 2. langage objet (continued)

```
class Guru (object) :
    def __init__(self, firstname, lastname,
                 domain='Python') :
        self.firstname = firstname
        self.lastname = lastname
        self.domain = domain

    def __str__(self) :
        return '%s %s, %s guru' % (self.firstname,
                                   self.lastname,
                                   self.domain)

    def quack(self):
        return "Coin !"

guido = Guru('Guido', 'Van Rossum')
bjarne = Guru('Stroustrup', 'Bjarne', 'C++')
```

Principales caractérisques : 3. dynamique

```
guido = Guru('Guido', 'Van Rossum', 'Python')

guido.title = 'BDFL'
guido.__dict__
```

```
=> {'firstname': 'Guido',
    'lastname': 'Van Rossum',
    'domain': 'Python',
    'title': 'BDFL'}
```

```
def rename(self, new_name) :
    self.firstname = new_name

Guru.rename = rename

guido.rename('Just')
```

Principales caractérisques : 4. bien équipé

Structures de données de base directement dans le langage :

```
list = ['un', 'deux', 3, 4]
dictionary = {1: 'un', 2: 'deux'}
```

ou pas très loin :

```
set = set([1, 2, 3, 4])
```

Librairie standard très complète (battery included) :

```
import email
import SimpleHTTPServer
import imaplib
import optparse
import threading
import logging
import csv
import xml
import unittest
```

Particularités : 1. list comprehension

Définition d'une liste à partir d'une autre :

```
ints = range(6)          # ints = [0, 1, 2, 3, 4, 5]
power_of_two = [pow(2,i) for i in ints]


def qsort(L):
    if L == []: return []
    pivot = L[0]
    return qsort([x for x in L[1:] if x < pivot])
         + [pivot] + \
          qsort([y for y in L[1:] if y >= pivot])
```

Particularités : 2. Duck typing : la généricité par défaut

"Si ca marche comme un canard et que ca fait coin comme un canard, c'est que ca doit être un canard."

```
def write_to_file(content, file):
    '''Write elements of content to a file-like
     object'''
    for item in content:
        file.write(str(item))
```

Pré-requis sur les arguments de cette fonction :

- `content` est iterable (implemente un des protocoles d'itération)
- les éléments de `content` proposent une représentation en chaîne de caractères
- `file` a une méthode `write` acceptant une chaîne de caractères

Parenthèse : protocoles d'itération

Est itérable tout objet fournissant :

- soit une méthode `__getitem__(index)` qui lève une exception `IndexError` si `index` est supérieur à la taille de l'objet
- soit une méthode `__iter__()` qui renvoie un object ayant une méthode `next()` qui lève une exception `StopIteration` en fin d'itération

```
class GuruList(list) :
    def __iter__(self) :
        return ReverseIterator(self)

class ReverseIterator(object) :
    def __init__(self, container):
        self.container = container
        self.index = len(container)
    def next(self) :
        self.index -= 1
        if self.index < 0 :
            raise StopIteration
        else :
            return self.container[self.index]
```

Particularités : 3. générateurs

Les générateurs sont un moyen d'implémenter des itérateurs générant des valeurs de façon faignante (lazy evaluation).

```
def xrange(max) :
    val = 0
    while val < max :
        yield val
        val += 1

xrange(10)   => <generator object at 0xb7d1f88c>

for i in xrange(10):
    print i
```

Particularités : 3. generateurs (continued)

Nouvelle implémentation de la classe `GuruList` et de son itérateur :

```
class GuruList(list) :

    def __iter__(self):
        index = len(self.container) – 1
        while index >= 0 :
            yield self.container[index]
            index -= 1
```

Le module `itertools` contient quelques fonctions intéressantes pour manipuler les itérateurs et générateurs (chainage, filtrage, cycles, ...).

Particularités : 4. programmation fonctionnelle

Exemple : curryfication

```
def multiply(a, b) :
    return a * b

class Curry (object) :
    def __init__(self, function, *args):
        self.function = function
        self.arguments = args

    def __call__(self, *args):
        arguments = self.arguments + args
        return apply(self.function, arguments)

multiply_by_2 = Curry(multiply, 2)

evens = map(multiply_by_2, [0, 1, 2, 3, 4, 5])
      => [0, 2, 4, 6, 8, 10]
```

Particularités : 5. Et aussi...

Excellent support d'Unicode

```
name = u'Journée Méditerranéenne'.encode('UTF-8')
=> 'Journ\xc3\xa9e M\xc3\xa9diterran\xc3\xa9enne'
```

Garbage collecting

Nombreux IDE disponibles : IDLE, plugin Eclipse, DrPython, eric, ...

Debuggeur dans la distribution standard.

Multi-plateformes : Unices, Windows, Mac OS, Nokia S60, OS/390, ...

Python pour quoi faire ?

Syntaxe claire et simple, pas (trop) de surprises, support de différents paradigmes de programmation (procédural, objet, fonctionnel, ...), introspection :

=> enseignement

Librairie standard très complète :

=> administration système, prototypage

Intégration et extension aisée en C, Java, .NET, nombreux tooltiks d'interface utilisateur (Tk, wxWidgets, GTK, ...)

`=> glue entre composants, interpréteur embarqué`

...

Et surtout :
Syntaxe claire et simple, "seulement une façon de faire" :

`=> code qui devra être relu`

Quelques points négatifs

Quelques incohérences dans le langage (principalement pour raisons historiques) comme :

- présence de deux systèmes d'objets en parallèle

- `', '.join(['guido', 'bjarne'])`
  au lieu de `['guido', 'bjarne'].join(', ')`

Python évolue rapidement. On peut avoir à supporter plusieurs versions. On trouve encore des installations en 2.2, 2.1 voire 1.5. Heureusement les incompatibilités avec les versions précédentes sont minimes.

Conclusion

Python, un langage
- moderne
- simple
- évoluant rapidement
- avec une importante communauté
- et surtout : incite à produire du code compréhensible et maintenable

Utilisé par la NASA, Google, Redhat, ...

Ressources :
- http://www.python.org : nouveautés, téléchargements, doc, ...
- http://planet.python.org : blog de la communauté Python
- http://cheeseshop.python.org/pypi : le CPAN de Python
- news://comp.lang.python
- http://aspn.activestate.com/ASPN/Python/Cookbook/