Qui suis-je ?

- Marc DEBUREAUX
- Master IAGL de l'université Lille 1 (2009)
- Consultant nouvelles technologies - Alten Nord

PYTHON

A LA DÉCOUVERTE D'UN LANGAGE LIBRE

Qu’est-ce que Python ?

- Langage orienté objet et interprété
- Structures de données haut niveau
- Syntaxe élégante et typage dynamique
- Complètement ouvert et open-source
- Communauté très active et grandissante

Quand utiliser Python ?

- Langage de choix pour :
  - l'apprentissage de la programmation
  - les scripts rapides (administration, bash, etc…)
  - le développement web (Django, Flask)
  - les environnements agiles (TDD, Scrum)
  - le multi-architecture
- … sans oublier que le langage est extensible !

Choisir une version

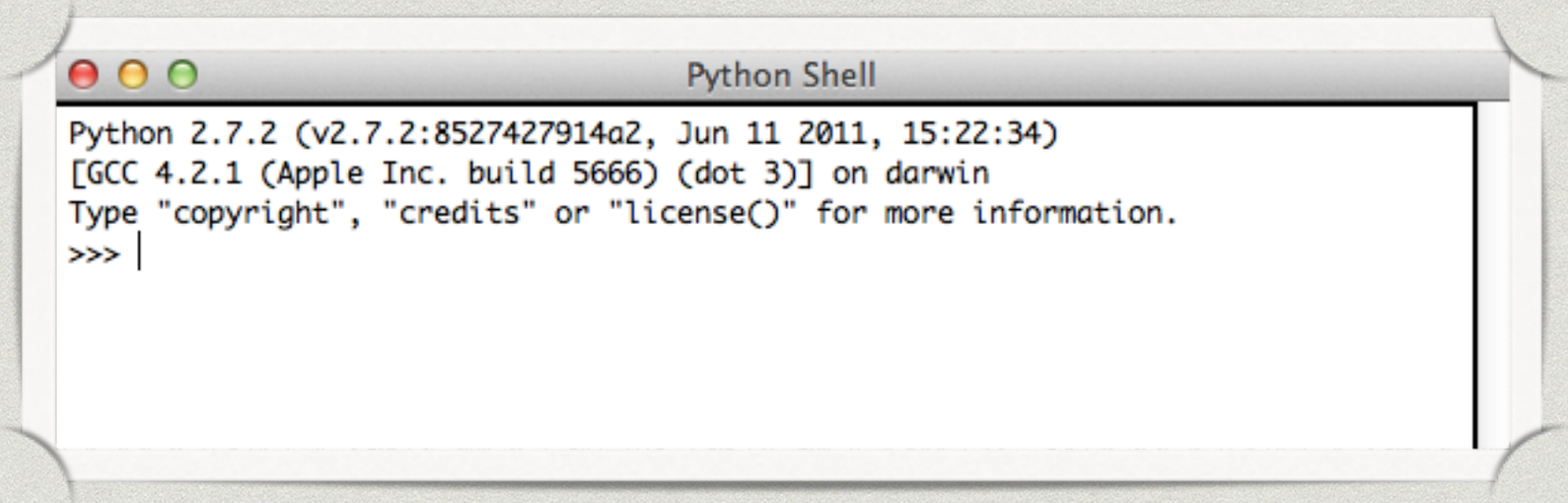
- 2 versions en développement (2.x et 3.x)
- Beaucoup de différences entre les versions
- Plus de support dans la version 2.x
- Nécessité d’un travail de migration vers la 3.x
- Les dernières versions de Django supportent les 2

Installer Python

- Surtout ne pas installer depuis les repositories !
- 2 solutions :
  - Installer depuis les sources
  - Installer les binaires précompilés par distribution
- Installer un gestionnaire de package derrière
  - easy_install ou pip

Premiers pas

- Utiliser le mode interactif de l'interpréteur Python



Banalités

- L'opérateur `=` sert à l'affectation
- L'opérateur `/` provoque une division entière si les deux opérandes sont des entiers
- Il existe des fonctions de conversion de types tels que `float()`, `int()` et `long()`

Banalités

- En mode interactif, la variable _ permet de récupérer le dernier résultat affiché
- Les chaînes de caractères peuvent être déclarées avec des simples ou doubles quotes
- Les triples quotes permettent d'écrire des chaînes de caractères sur plusieurs lignes
- L'opérateur + permet de concaténer les chaînes de caractères, mais il convient de faire autrement

Banalités

- Les chaînes de caractères peuvent être manipulées comme des tableaux de caractères
- Cependant, elles sont non mutables et il n’est pas possible d’affecter directement un caractère
- Il est possible de définir explicitement des chaînes de caractère en unicode ou d’en convertir

Règles d'écriture

- Utilisez les espaces pour indenter, pas les tabs.
- Evitez les lignes de plus de 80 caractères.
- Séparez les classes et les fonctions par des lignes blanches pour plus de lisibilité.
- Mettez un espace de part et d'autre des opérateurs et après les virgules.
- Utilisez les commentaires de type docstrings.

Contrôles de flux

- Si… alors… sinon…

```
# coding=utf-8
nombre = 10
if nombre < 0:
    print nombre, 'est négatif.'
elif nombre > 0:
    print nombre, 'est positif.'
else:
    print nombre, 'est nul.'
```

```
# coding=utf-8
nombre = 10
print nombre, 'est négatif.' if nombre < 0 \
    else 'est positif.' if nombre > 0 \
    else 'est nul.'
```

Boucles

- Pour chaque… dans…

```
# coding=utf-8
tableau = [1, 2, 3, 4, 5, 6, 7, 8, 9]
for element in tableau:
    if element % 2 == 0:
        print element, 'est pair.'
    else:
        print element, 'est impair.'
```

- `range()` peut être bien pratique ici

```
for element in range(1, 10):
    print element
```

Boucles

- Tant que… faire…

```
# coding=utf-8
element = 1
while element < 10:
    print element
    element += 1
```

- `break` : arrête l'itération dans une boucle
- `continue` : passe à l'itération suivante
- `else` : exécuté si le `for` a tout itéré sans `break`

Fonctions

- Le mot clé `def` permet de déclarer une fonction

```
# coding=utf-8
def fibonacci(max_value = 100):
    """Retourne les nombres de la suite de Fibonacci"""
    values = [0]
    current = 1
    while current < max_value:
        values.append(current)
        current = sum(values[-2:])
    return values

print fibonacci(200)
```

- Une fonction sans retour retournera `None`

Fonctions

- Arguments par défaut et arguments nommés

```
# coding=utf-8
def test(a = 1, b = 2, c = 3):
    print '%d %d %d' % (a, b, c)

test() # 1 2 3
test(7, 8, 9) # 7 8 9
test(a = 7, b = 8, c = 9) # 7 8 9
test(7, 8, c = 9) # 7 8 9

args = [7, 8, 9]
test(*args) # 7 8 9

args = {'a': 7, 'b': 8, 'c': 9}
test(**args) # 7 8 9
```

Expressions lambda

- Fonctions anonymes à une seule expression

```
# coding=utf-8
est_pair = lambda x: x % 2 == 0

print est_pair(4) # True
print est_pair(7) # False

valeurs = [
    (1, 'un'),
    (2, 'deux'),
    (3, 'trois'),
    (4, ‘quatre'),
]
valeurs.sort(key=lambda pair: pair[1])
print valeurs
# [(2, 'deux'), (4, 'quatre'), (3, 'trois'), (1, 'un')]
```

Listes

- Méthodes utiles…

`list.append(x)`	`list.extend(L)`
`list.insert(x)`	`list.remove(x)`
`list.pop([i])`	`list.index(x)`
`list.count(x)`	`list.sort()`
`list.reverse()`	`del …`

- Documentation incluse !

```
# coding=utf-8
print list.remove.__doc__
# L.remove(value) -- remove first occurrence of value.
# Raises ValueError if the value is not present.
```

Listes

- … et fonctions utiles !
- `filter(fonction, séquence)`

```
print filter(lambda x: x % 2 == 0, range(1, 20))
# [2, 4, 6, 8, 10, 12, 14, 16, 18]
```

- `map(fonction, séquence)`

```
print map(lambda x: x / 2, range(10, 100, 10))
# [5, 10, 15, 20, 25, 30, 35, 40, 45]
```

- `reduce(fonction, séquence)`

```
print reduce(lambda x, y: x + y, range(1, 10))
# 45
```

Listes

- Simplification de l'écriture des listes (comprehension)

```
carres = []
for nombre in range(10):
    carres.append(nombre ** 2)
print carres
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
carres = map(lambda nombre: nombre ** 2, range(10))
print carres
```

```
carres = [nombre ** 2 for nombre in range(10)]
print carres
```

Tuples

- Un moyen de packager des données

```
# coding=utf-8

t = (3, 'trois', 3 % 2 == 0)
print t
# (3, 'trois', False)

nombre, texte, est_pair = t
print nombre, 'est pair' if est_pair else 'est impair'
# 3 est impair
```

- Attention ! Les tuples ne sont pas mutables.

Sets

- Une liste sans ordre n'acceptant pas les doublons

```
# coding=utf-8
tableau = ['rouge', 'bleu', 'noir', 'rouge', 'jaune', 'noir']
couleurs = set(tableau)
print couleurs
# set(['bleu', 'jaune', 'noir', 'rouge'])

print 'noir' in couleurs # True
print 'vert' in couleurs # False
```

- Attention ! Les sets ne sont pas indexables…

Sets

- Cependant ils ont des propriétés intéressantes !

```
# coding=utf-8
A = set('python')
B = set('django')

print A # set(['h', 'o', 'n', 'p', 't', 'y'])
print B # set(['a', 'd', 'g', 'j', 'o', 'n'])

print A - B # Lettres qui sont dans A mais pas dans B
# set(['y', 'h', 't', 'p'])
print A | B # Ensemble des lettres de A et de B
# set(['a', 'd', 'g', 'h', 'j', 'o', 'n', 'p', 't', 'y'])
print A & B # Lettres qui sont dans A et dans B
# set(['o', 'n'])
print A ^ B # Lettres qui sont dans A ou B mais pas les deux
# set(['a', 'p', 'd', 'g', 'y', 'h', 'j', 't'])
```

Dictionnaires

- Un ensemble de couples clé/valeur

```
# coding=utf-8
ages = {
    'marc': 26,
    'eric': 39,
    'elodie': 23,
    'frank': 32,
}
print ages
# {'frank': 32, 'elodie': 23, 'eric': 39, 'marc': 26}

print ages.keys() # ['frank', 'elodie', 'eric', 'marc']

carres = {x: x ** 2 for x in range(5)}
print carres
# {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

Itérations et énumérations

- Quelques fonctions bien pratiques…

- `enumerate(séquence)`

```
tableau = ['python', 'ruby', 'perl']
for i, v in enumerate(tableau):
    print i, v # 0 python ...
```

- `sorted(séquence)`

  retourne une liste !

```
tableau = [4, 8, 1, 3, 5]
print sorted(tableau)
# [1, 3, 4, 5, 8]
```

- `reversed(séquence)`

  retourne un itérateur !

```
for x in reversed(xrange(10)):
    print x
    # 9 8 7 6 5 4 3 2 1 0
```

- `xrange(…)`

Opérateurs court-circuit

- Evaluation des expressions booléennes de la gauche vers la droite
- Equivalents à `False` :

False	None	0	0L	0.0	0j	‘’	()	[]	{}

- Ne retournent pas explicitement `True` ou `False`

```
# coding=utf-8
b1, b2, b3 = True, False, True
print b1 and b2 and b3 # False
# b3 n'est pas évalué car résultat déterminé à b2

s1, s2, s3 = '', 'python', 'django'
print s1 or s2 or s3 # 'python'
```

Modules

- Permet de cataloguer et hiérarchiser les définitions des fonctions et des classes dans des fichiers
- Le nom de l'import est celui du fichier enregistré

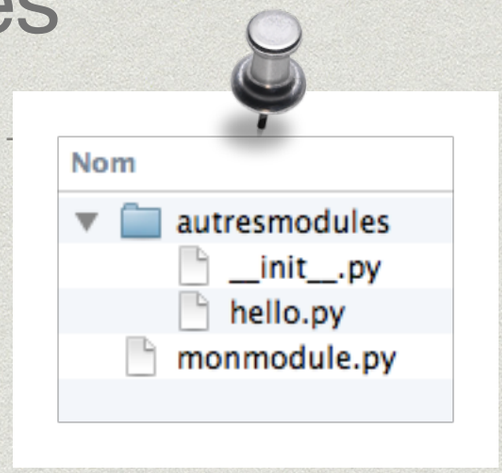
```
# coding=utf-8

import monmodule
print monmodule.fibonacci(50)
# [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

from monmodule import fibonacci
print fibonacci(50)
# [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Modules

- Les modules sont organisés en packages




```
# coding=utf-8
import mon_module
dir(mon_module)
# ['__name__', 'fibonacci']
dir()
# ['__builtins__', '__name__', '__package__', 'monmodule']
print mon_module.__name__
# 'monmodule'
from autresmodules.hello import hello_world
hello_world()
# 'Hello world!'
```

```
# __init__.py
__all__ = ['hello']
```

Formatting

- De nombreuses méthodes de formatage…

```
# coding=utf-8
print '{} est un super {}'.format('python', 'langage')
# python est un super langage
print '{langage} est en version {version}'.format(
        langage='python', version='2.7.6')
# python est en version 2.7.6
import math
print 'PI vaut approximativement {0:.3f}'.format(math.pi)
# PI vaut approximativement 3.142
```

- … et quelques fillers !

- `str.rjust()`, `str.ljust()`, `str.center()`, `str.zfill()`

Manipulation de fichiers

- Fonctionne à la fois comme un flux et un itérateur

```
# coding=utf-8
f = open('fichier1.txt', 'r')
print f.read()
# Contenu de fichier.txt
print f.read()
# ''
```

```
# coding=utf-8
f = open('fichier2.txt', 'r')
print f.readline()
# Première ligne
print f.readline()
# Seconde ligne
```

```
f.close()
f.read()
# ValueError: I/O operation on closed file
```

```
with open('fichier.txt', 'r') as f:
    texte = f.read()
f.closed # True
```

Gestion d'erreur

```
# coding=utf-8
try:
    # Bloc à tester...
    f = open('fichier.txt')
except IOError as e:
    # En cas d'erreur de type I/O...
    print e
except:
    # Erreur non prise en charge...
    print 'Erreur inconnue !'
else:
    # En cas de succès...
    print f.read()
    f.close()
finally:
    # Dans tous les cas...
    print 'Fin'
```

- Lever explicitement une exception via le mot-clé `raise`
- Les exceptions personnalisées doivent hériter de `Error`

Classes

- Déclaration et utilisation d'une classe :

```
# coding=utf-8
class MaClasse():
    valeur = 0
    def __init__(self, valeur):
        self.valeur = valeur
    def afficher(self):
        print self.valeur

a = MaClasse(10)
a.afficher() # 10

a.valeur = 20
a.afficher() # 20
```

```
# coding=utf-8
class Personne:
    pass

marc = Personne()
marc.nom = 'Marc'
marc.age = 27
marc.famille = []

eric = Personne()
eric.nom = 'Eric'
eric.age = 39
marc.famille.append(eric)
```

Héritage

```
a = ClasseMere()
a.super() # 0
a.test() # 1

b = ClasseFille()
b.super() # 0
b.test() # 2
```

```
# coding=utf-8
class ClasseMere():
    def super(self):
        print 0
    def test(self):
        print 1

class ClasseFille(ClasseMere):
    def test(self):
        print 2
```

- Méthodes utiles :

- `isinstance(instance, type)` : permet de déterminer si l'instance donnée est du type fourni ou d'un type dérivé

- `issubclass(type1, type2)` : permet de déterminer si le type 1 dérive directement du type 2

- L'héritage multiple est possible (mais déconseillé)

That’s all folks!

http://docs.python.org/2/tutorial/index.html