

Les bases du langage Python

Gaël LE MIGNOT – Pilot Systems

Décembre 2006

Plan

- 1 Les bases de Python
 - Introduction
 - La syntaxe de Python
 - Les types de Python
- 2 Fonctions et classes

- Les fonctions
 - Les classes
 - Les exceptions
- 3 Modèle d'exécution
 - 4 Modules et packages
 - 5 La bibliothèque standard

Pilot Systems

Une SSL

Pilot Systems est une Société de Services en Logiciels Libres.

Spécialisée en Python, Zope et Plone

- Python : un langage objet dynamique et flexible
- Zope : un serveur d'application orienté Web
- Plone : un CMS modulaire et extensible

Python

Historique

- Créé en 1990 par Guido van Rossum
- En 1995, sortie de Python 1.6.1, compatible GPL
- Actuellement, en version 2.5

Caractéristiques

- Langage impératif, objet et fonctionnel
- Typage dynamique
- Introspection et modification dynamique du code

Python, un interpréteur

Un langage interprété

- Le programme `python` en mode interactif
- Le programme `ipython` : un shell Python évolué
- La fonction `eval`

Un langage byte-compilé

- Byte-compilation
- Les fichiers `.pyc`, générés dynamiquement

Syntaxe simple

Éléments de syntaxe

- Nombres 2, chaînes `'Hello, world'`
- Expressions: `2 * len('Hello, world')`
- Affectations: `a = 2 * len('Hello, world')`
- Commentaires: `a = 2 * b # Double of b`

Règles de syntaxe de Python

- Une instruction par ligne, pas de séparateur
- Affectations multiples autorisées `a = b = 0`
- Les affectations n'ont pas de valeur

Les blocs

Les blocs en Python

- Pas de délimiteurs
- L'indentation délimite les blocs
- Utilisation: conditions, boucles, fonctions

Exemple

```
if nb < 0:  
    print "Warning, nb < 0"  
    nb = 0  
max = min + nb
```

Les conditions

Syntaxe de `if`

- Syntaxe de base: `if condition: instruction`
- Clause else: `else: instruction`
- Contraction du else-if:
`elif condition: instruction`

Les opérateurs booléens

- Les opérateurs s'écrivent en toutes lettres: `and`, `or`, `not`
- Le `or` et le `and` sont *lazy*
- La valeur est la dernière expression évaluée:
`0 or 1 and 2 or 3` vaut `2`

Les boucles

La boucle `while`

- Syntaxe de base: `while condition: instruction`
- Fonctionne comme en C

La boucle `for`

- Syntaxe de base:
`for variable in iterable: instruction`
- Exemple: `for i in range(5): print 2*i`
- Un *iterable*: un objet supportant le protocole itération, par exemple les listes

Types unitaires

Les types simples

- Tout est objet, en Python
- Nombres (entiers, flottants): `42`, `2.5`
- Les booléens: `True` et `False`
- Le type spécial `None`

Les types complexes

- Le type fichier: `file(chemin)`
- Des types divers, comme `code` ou `fonction`

Les listes et les tuples

Généralités

- Séquences d'éléments ordonnés
- Listes *mutable*, tuples *immutable*
- Syntaxe: [2, 3, 5, 7] et (2, 3, 5, 7)

Opérateurs et fonctions

- Accès à un élément: `l[2]`, `l[-2]`
- Accès à une série d'éléments: `l[2:5]`, `l[:-2]`
- Copie d'une liste: `l[:]`
- Longueur: `len`, appartenance: `in`
- Concaténation: `+`, duplication: `*`

Les chaînes

Généralités

- Les chaînes: délimitées par ' et "
- Les chaînes multilignes: ''' et """
- Objets non modifiables (*immutable*)

Méthodes et opérateurs

- Méthodes: `lower`, `split`, `join`, ...
- Se comportent comme des tuples de caractères
- L'opérateur `%`: formatage `printf` simple et avancé

Les dictionnaires

Généralités

- Tableaux associatifs
- En réalité: des tables de hash
- Les clés doivent être *immutable*
- Objets *mutable*

Utilisation

- Création: `{}, { "answer": 42, "question": None }`
- Accès à un élément: `d["answer"]`
- Méthodes: `keys, values, items`

Les fonctions

Définition de fonctions

- **Syntaxe:** `def fonction(parameters): bloc`
- **Valeur de retour:** `return`
- **Exemple:** `def sum(a, b): return a + b`

Le passage de paramètres

- **Valeur par défaut:**
`def mul(a, b = 2): return a*b`
- **Appel classique:** `mul(4)` **ou** `mul(4, 3)`
- **Appel nommé:** `mul(a = 4)` **ou** `mul(b = 3, a = 4)`
- **Appel mixte:** `mul(4, b = 3)`

Les objets fonction

Généralités

- Les fonctions sont des objets comme les autres
- On peut les passer en paramètres, ...
- Utilité: par exemple, la méthode `sort`

Les lambdas

- Fonctions anonymes
- Ne peuvent contenir qu'une expression
- Exemple: `l.sort(lambda s1,s2: cmp(len(s1), len(s2)))`

Les doc strings

Généralités

- Une *doc string* est un commentaire visible par Python
- Elle se met en première ligne d'une fonction, classe ou module
- Elle peut être récupérée par l'attribut `__doc__` et la méthode interactive `help`

Exemple

```
def add(a, b):  
    """This method does an addition"""  
    return a + b  
print add.__doc__
```


Les opérateurs * et **

Principe et utilité

- Répond au besoin de fonction à nombre d'arguments variable
- Principe: on passe les paramètres depuis ou vers des conteneurs
- * converti des arguments non nommés en tuple
- ** converti des arguments nommés en dictionnaire

Exemples

- `def f(*args, **kwargs): print args, kwargs`
- `complex_function(**params)`

Définition de classes

Syntaxe

- `class name (inheritance) : code`
- L'héritage peut se faire sur n'importe quelle classe, ou sur `object`
- Créer un objet se fait en appelant la classe

Sémantique

- Toutes les méthodes sont virtuelles
- L'héritage multiple se fait en profondeur d'abord
- Fonction `isinstance` et attribut `__class__`
- Méthodes privées

Exemple de classe

Exemple

```
class Counter(object):  
    name = "Counter"  
  
    def __init__(self, value = 0):  
        self.value = value  
  
    def get(self):  
        self.value += 1  
        return self.value
```

Méthodes liées et libres

Le paramètre `self`

- Toute méthode prend en premier paramètre l'objet lui-même
- Par convention, on l'appelle `self`

Les méthodes libres et liées

- Lorsqu'une méthode est récupérée sur un objet, elle est dite *bound*
- Lorsqu'une méthode est récupérée sur une classe, elle est dite *unbound*
- L'objet `self` est implicite dans les méthodes liées, mais doit être spécifié dans les méthodes libres

Principe des exceptions

Principe des exceptions

- Une exception correspond à une erreur, ou une nécessité d'arrêter le traitement en cours
- Elle se propage, remontant la pile d'appel, jusqu'à être interceptées
- En Python, une exception est une classe héritant d'`Exception`

Les exceptions standard

Les exceptions standard

- **Génériques:** `RuntimeError`
- **Liées au code:** `SyntaxError`, `NameError`,
`AttributeError`
- **Liées au système:** `SystemExit`, `IOError`,
`KeyboardInterrupt`

Récupérer des exceptions

Le bloc try-except

- Du code pouvant générer (directement ou non) une exception peut être entouré d'un bloc `try` : et d'un bloc `except` :
- Le code présent dans le `try` sera exécuté jusqu'à ce qu'une exception arrive
- Le code se trouvant dans le `except` : ne sera exécuté que si une exception arrive
- Le `except` stoppe la propagation de l'exception

Nettoyer après une erreur

La clause `finally`

- Lorsqu'une erreur survient, on peut avoir besoin de nettoyer (par exemple, effacer un fichier temporaire)
- Le code de la clause `finally` sera exécuté systématiquement, après le reste
- La clause `finally` ne stoppe pas la propagation
- Attention, avant Python 2.5, on ne peut pas avoir `except` et `finally` dans le même `try`

Autres modes de récupération

La clause `except` avec un type

- Un type peut être précisé dans la clause `except`, pour ne récupérer qu'un type d'exceptions
- Toutes les exceptions héritant de ce type seront aussi interceptées
- Il est possible d'avoir plusieurs bloc `except` de suite

Récupérer l'objet exception

- Une syntaxe du type `except IOError, exc` permet de récupérer l'objet exception
- Des méthodes du module `sys` le permettent aussi

Déclencher une exception

La clause raise

- Déclencher une exception se fait avec `raise`
- Exemple:

```
if i < 0:  
    raise RuntimeError, 'i must be positive'
```
- Dans une clause `except`, un `raise` sans paramètre relance l'exception courante

Définir une exception

Définir une exception personnalisée se fait en héritant de la classe `Exception`

Tout est dictionnaire

Des dictionnaires partout

- Les symboles globaux, locaux et builtins sont des dictionnaires
- Une affectation ne crée qu'une référence dans un dictionnaire
- Les builtins: `__builtins__.__dict__`

Dans les classes

- Tout est dans `__dict__`
- Exceptions: les classes de bases

Le garbage collector

Un garbage collector cyclique

- Le compteur de références
- Le détecteur de cycles
- Le mot clé `del`
- Le module `gc`

Les cycles et les destructeurs

- La méthode spéciale `__del__`
- Les cycles et `__del__`

Utilisation de modules et de packages

La clause import

- Syntaxe: `import package`
- Permet d'importer un module externe
- On peut préciser un nom avec `as`

Exemple

```
import os
import os.path as ospath
print os.getenv("PATH")
print ospath.join("home", "kilobug", ".bashrc")
```

Utilisation de modules et de packages (2)

La clause from

- Permet d'importer des symboles directement dans l'espace courant
- Permet de n'importer que ce dont on a besoin
- Exemple: `from os import path`
- Il est possible, mais déconseillé, d'utiliser *

Importation avancée

- Les variables `sys.path` et `PYTHONPATH`
- Le module `imp` permet d'importer un module Python depuis un fichier

Création de modules et de packages

Création de modules

- Un module est fichier Python définissant des symboles
- Le code *top-level* est exécuté à l'import
- Un module peut en importer d'autres, mais pas de manière circulaire

Création de packages

- Un package est répertoire contenant des modules (et éventuellement des packages)
- Il doit contenir un fichier `__init__.py`

Aperçu

Une bibliothèque riche

- Python contient une bibliothèque standard très riche
- Des extensions nombreuses: PIL, Twisted, connecteurs SQL

Des ressources en ligne

- Sur le site de Python
- Depuis l'interpréteur
- La tabcompletion d'ipython

Quelques packages

Les modules génériques

- `sys`, `imp`, `gc`: accès à l'intérieur de Python
- `string`, `re`: manipulation de chaînes et regexp
- `os`, `glob`, `commands`: accès à l'OS
- `math`, `random`: maths et nombres aléatoires

Les modules réseau

- `urllib` **et** `httpplib`
- `smtplib`, `poplib` **et** `mail`
- `xmlrpclib`