PPOOGL

Florent de Dinechin

Python pour les nuls

Python en 30 minutes

Introduction

Grands principes

Python en diagonale

Le reste est dans le manuel

Introduction

La première fois

- Fichier source `Toto.py`, fichier objet `Toto.pyc` et `Toto.pyo` *mais on s'en fiche*
- `python Toto.py` exécute Toto.py, et parfois crée `Toto.pyc` (mais encore une fois on s'en fiche)
- Voici le code source :

```
print "Python est plus compact que Java"
```

Grands principes

Les trucs reposants

- *Dans tes fichiers tu mets ce que tu veux*
  - Une classe, deux classes, pas de classe du tout.
- *Langage interprété et édition de lien dynamique*
  - Lorsque Python a besoin d'un fonction définie autrepart, il la charge (et la compile si nécessaire).
  - ⊖ Performance encore plus erratique que Java : des accès disques et même parfois des compilations cachées au milieu de votre programme
- *Indentation signifiante* : au lieu d'user la touche AltGr, on use la touche tab.
- Tout est objet (même les fonctions etc)

Les trucs reposants pour vous, pas pour moi

- Langage fortement typé, mais typé *dynamiquement*
  - Rien besoin de déclarer
  - Si vraiment on veut déclarer un objet on fait `a=None` (non typé)
- Donc, pas de notion de *portée* lexicale

(à suivre)

Python en diagonale

Introduction

Grands principes

Python en diagonale

Le reste est dans le manuel

Classes : syntaxe

- Déclaration d'une classe :

```
class Case:
    "Une case de notre super jeu"
    meteo=None
    def AfficheToi(self):
        "Methode d'affichage universelle"
        raise NotImplementedError()
```

- Instanciation d'un objet de cette classe :

```
a=Case()
```

Héritage : syntaxe

- Définition d'une sous-classe :

```
class CaseMer(Case):
    "Case maritime, avec un courant et tout"
    courant=None
    # TODO: surcharger AfficheToi()
```

Héritage multiple

```
class A(B,C,D):
(...)

a=A()
a.toto()
```

- Pb en cas de surcharge : s'il y en a plusieurs, c'est quel `toto()` qu'on appelle ?
- Réponse : le premier dans une recherche *en profondeur d'abord, de gauche à droite*.
  - D'abord on cherche `toto()` dans `A`
  - puis dans `B`
  - puis dans toutes les surclasses de `B`
  - puis dans `C`, puis dans toutes les surclasses de `C`
  - etc

Autres subtilités de l'OO

```
class A:

   def f(self):
      self.g()

   def g(self):
     print "Bonjour"

class B(A):
   def g(self):
     print "Au revoir"

a=A()
b=B()
a.f()
b.f()
```

... donc il faut se méfier.

Exercice : traduire ceci en Java (qui aura le même comportement).

Autodocumentation (introspection)

- Tout est objet, et donc tout hérite (entre autres) de la chaîne de documentation `__doc__`
  - Syntaxe pour la définir pour les fonctions et pour les classes
  - Par défaut, la documentation d'un objet est celle de sa classe
- `pydoc` : outil qui fait des (genre de) pages `man` ou HTML à partir de ces chaînes.

Voir aussi les fonction `str` (conversion de n'importe quel objet en chaîne de caractères), `dir` (liste le contenu de n'importe quel objet), `type` (done son type), etc.

Modules et Paquetages

- Un module c'est un fichier
- un paquetage c'est un répertoire avec un fichier `__init__.py` qui définit les modules qu'il contient (exemple `/usr/lib/python/xml/`).
- Rien à voir avec la hiérarchie des classes, m'enfin je serais vous je les ferais correspondre
- Il ya un chemin (path) de recherche des modules
- En pratique les modules standards sont tous à plat dans `/usr/lib/python`
- Pas besoin de déclarer qu'une classe `Toto` fait partie d'un paquetage `projetLala.tata`
  - Plus facile qu'en Javal de réorganiser le code après-coup
  - Moins de pression sur vous pour réfléchir à l'avance

Paquetages et nommages

- Le nom complet d'une classe est `paquetage.Module.Class`
- Le nom complet d'un membre est `paquetage.Module.Classe.membre`
  - la fonction sinus : `math.sin(1)`
- Le mot-clé `import` : "importe dans l'espace de nommage courant"
  - `import math`
    `math.sin(1)` est OK
  - `from math import sin` ou bien `from math import *`
    `sin(1)` est OK

Quelques conventions

- J'ai l'impression qu'en Python tout le monde écrit en minuscules
- Vous faites bien ce que vous voudrez.

Pourquoi je crois pas à Python pour PPOOGL

- En Python il n'y a pas moyen de définir un membre privé. C'était bien la peine.
  - Les membres avec deux underscore devant sont un peu cachés (presque privés – mais c'est un bricolage).
- En Python il n'y a pas moyen de définir un membre statique. C'était bien la peine.
- En Python il n'y a pas moyen de définir un membre abstrait. C'était bien la peine.
- En python on compte sur les programmeurs pour être bien élevés. Z'ont jamais vu de L3IF.

Bref. Python fait trop confiance aux programmeurs, et n'invite pas à la modélisation préalable.

Quelques trucs mieux qu'en Java

- Listes, tuples et dictionnaires faciles à utiliser
- Itérateurs
  - `for` variable `in` truc
    marche pour truc étant une liste, un tuple, une chaîne, un dictionnaire...
  - (caché derrière, c'est de l'orienté-objet)
  - générateurs comme `range(17,42)`
  - (Tout cela existe en Java mais faut voir l'usine à gaz)

Le reste est dans le manuel

Références

- Dive into python (existe en français, installé par défaut par Ubuntu)
- Python tutorial et Python Library reference