Tisseur Python

Serge Guelton <serge.guelton@enst-bretagne.fr>

22 janvier 2018

1 Trucs et astuces avant la programmation orientée aspect

1.1 PATH

Les scripts *shell*, c'est comme les céréales du matin : on en mange tout le temps et on adore ça. Regardons comment modifier un script existant sans le toucher...

Exercices

1. Écrivez un bête script d'une ligne qui compte le nombre de lignes dans un fichier, en utilisant `wc` [1]
2. À l'aide de la commande `which`, identifiez le chemin du binaire `wc`.
3. Écrivez un programme qui se comporte comme `wc`, mais qui est plus poli : il commence par vous saluer avant de faire son traitement.
4. En modifiant le `PATH`, faîtes en sorte que ce soit ce programme qui soit appelé par votre script initial.

1.2 LD_PRELOAD

La variable d'environnement `LD_PRELOAD` contient une liste de chemins de bibliothèque dont les fonctions seront chargées **avant** celles du même nom normalement requises par une bibliothèque.

Exercices

1. Écrivez un « hello world » en C, compilez le.
2. À l'aide de `nm` et/ou `strace` déterminez la fonction de la `libc` utilisée pour afficher le sympathique message de salutation.
3. À l'aide de `LD_PRELOAD`, de `gcc -shared` et de votre cerveau, injectez un code dans cette fonction modifiant son comportement.

1.3 Décorateurs Python

En Python, un décorateur est une fonction/classe prenant en argument une fonction et renvoyant une nouvelle fonction. Il est appelé de la façon suivante :

1. En vrai, un alias suffit.

```
def mydecorator(fn):
  def wrapper(arg):
    print("wrapped!")
    return fn(arg)
  return wrapper
@mydecorator
def foo(bar):pass
```

On peut étendre le concept aux classes de la sorte :

```
class mydecorator(object):
    def __init__(self, arg):
        self.arg = arg

    def __call__(self,fn):
      def wrapper(arg):
        print(self.arg)
        return fn(arg)
      return wrapper

@mydecorator("wrapped")
def foo(bar):pass
```

Exercices

1. Examinez le bout de code précédent et inspirez-vous en pour écrire un décorateur qui affiche la trace d'appel (cf. `traceback`) de la fonction décorée.
2. En utilisant la forme classe, modifiez le décorateur précédent pour qu'il ne s'active que si la fonction encapsulée possède une fonction `bar` dans sa pile d'appel

Voyez vous les possibilités vis-à-vis de la programmation orientée aspect ?

2 Découverte de l'introspection en Python

Pour comprendre comment une voiture fonctionne, quoi de mieux que de la démonter et de la remonter ? Bien sûr, cela ne vous apprend pas à conduire . . .

Le but de cette partie du TP est de construire un petit tisseur d'aspect pour le langage python. Il n'aura pas toutes les fonctionnalités d'un tisseur moderne, mais cela vous permettra de mieux comprendre le fonctionnement du tissage.

Le langage de support choisi — Python — facilite grandement certaines manipulations, grâce à l'introspection. Si vous n'êtes pas un familier du langage, dites vous que vous le serez un peu plus après ces 4h `:-)`.

2.1 `dir()` et `globals()`

Ouvrez un interpréteur Python (`ipython` est un bon choix), et appelez les fonctions `globals()` et `dir()`. Vous pouvez obtenir de l'aide sur ces fonctions à l'aide de la fonction `help`.

Définissez quelques fonctions bidons avec paramètres, puis refaites un appel à `globals()` et `dir()`. Qu'observez-vous ?

2.2 `isinstance` et `type`

À l'aide des fonctions `isinstance(objet, type)` ou `type(object)`, filtrez les fonctions des autres variables du dictionnaire produit par `globals()`. Éventuellement vous aurez besoin du module `types`.

2.3 Passage de paramètres en Python

Il existe trois types de paramètres en Python :

1. les paramètres nommés, comme dans

```
def fibo(n):
        return n if n<=1 else fibo(n-1)+fibo(n-2)
```

2. les paramètres non nommés, comme dans

```
def init_list(l,*values):
        for v in values: l.append(v)
```

3. le dictionnaire de paramètres, comme dans

```
def sponge_bob(**kwargs):
    for (k,v) in kwargs.iteritems():
        print(k, "=>", v)
```

Définissez une nouvelle fonction, puis construisez une fonction qui encapsule cette dernière en imprimant tous ses arguments, puis qui passe ces arguments à travers le véritable appel.

Définissez ensuite une fonction générique qui effectue ce travail, avec la signature.

```
def wrap(f,*args, **kwargs):
  # print *args and **kwargs
  # call f with them
```

2.4 Redéfinition de fonction à la volée

Il est possible en Python de créer des fonctions dynamiquement, et même de renvoyer celles-ci (ce sont des objets de premier ordre, au même titre que les entiers, les listes etc.). En vous basant sur le code de `wrap`, écrivez une fonction `fwrap` qui crée une nouvelle version de la fonction donnée en paramètres et la renvoie.

```
def fwrap(f):
  # define a new function based on f and returns it
```

3 Premier tisseur d'aspect

À l'aide de l'expérience accumulée lors de la première section, écrivez dans un fichier `aspect.py` les fonctions suivantes, dont le nom devrait vous aider à savoir leur but

```
def before_call(join_point, advice):
    """ creates a new function based on join_point, which calls
        advice
    before it"""
def after_call(join_point, advice)
    """ creates a new function based on join_point, which calls
        advice
    after it"""
def around_call(join_point, advice_before, advice_after)
    """ creates a new function based on join_point, which calls
    advice_before before it and advice_after after it"""
```

Utilisez maintenant le module `re` combiné avec `__import__` et `setattr(...)` pour tisser certains *advice* dans les fonctions qui satisfont une certaine expression régulière. Autrement dit, définissez des fonctions qui prennent une coupe au lieu d'un point de jonction en paramètre.

4 Découverte de `ast.parse(...)`, `eval(...)` et du module `ast`

4.1 Compilation dynamique

Utilisez l'aide pour comprendre le fonctionnement de la fonction `ast.parse(...)` et de son interaction avec `eval(...)`. Essayez par exemple de définir dynamiquement une fonction à partir d'une chaine de caractères lue en paramètre (cf. module `sys`).

Téléchargez le module `unparse` disponible dans les sources du projet python http://svn.python.org/projects/python/trunk/Demo/parser/unparse.py. Utilisez le pour décompiler une chaîne de caractère compilée avec `ast.parse(...)`.

4.2 Modification de l'arbre de syntaxe abstrait

Affichez à l'écran le résultat d'un appel à `ast.parse(...)` en utilisant `ast.dump(...)`. Vous pouvez aussi utiliser `dir(obj)`, `type(obj)` et la tabulation fournie par `ipython` pour naviguer dans les objets. La documentation du module `ast` est salvatrice ici.

Voyez vous les possibilités vis-à-vis de la programmation orientée aspect ?

Essayez dans un premier temps de parser la chaine de caractères "␣`print('hello')`".

```
import ast
s="print('hello')"
a=ast.parse(s)
eval(compile(a,"","exec"))
a.body[0].values[0].s="salut"
eval(compile(a,"","exec"))
```

En se basant sur cette séquence, et ce que vous avez appris sur les types, écrivez une fonction qui va traverser l'AST et mettre en capitale d'imprimerie toutes les chaines de caractères constantes rencontrées dans l'AST. Pour cela vous aurez besoin de lire attentivement la documentation sur `class ast.NodeTransformer` fournie sur http://docs.python.org/library/ast.html.

4.3 Deuxième tisseur d'aspect

Vous avez maintenant tous les outils pour écrire un tisseur d'aspect plus général. L'objectif sera d'ajouter une trace, du genre « j'entre/sors de telle fonction » sur tous les appels de fonction d'un code. Pour cela vous aurez besoin de modifier tous les appels à `ast.Call`, à moins que vous préfériez vous attaquer à `ast.FunctionDef`.

Améliorez ensuite votre tisseur pour qu'il n'instrumente que les fonctions qui satisfont une expression régulière donnée.

Utilisez `open(...)` combiné avec `ast.parse(...)` et le module `unparse` pour rendre votre tisseur un peu plus générique, dans le sens ou il prend maintenant un fichier en entrée et génère un nouveau fichier python en sortie. Vous venez d'écrire un compilateur source-à-source qui fait un forme restreinte de programmation aspect !

Conclusion

C'est vous qui concluez sur ce que vous avez appris, de mon côté j'ai eu beaucoup de plaisir à écrire ce TP, et j'espère que cette approche par le bas vous aura plu !