Python avancé

Skander Zannad et Judicaël Courant

Lycée La Martinière-Monplaisir

2013-12-14

1 Tests automatisés

1.1 Motivation

Cadre : on veut écrire une fonction `nb_carres(n)` calculant le nombre de diviseurs de `n` :

```
def diviseurs(n):
    d = 1
    k = 0
    while d**2 < n:
        # k : nombre de diviseurs trouvés
        if n % d == 0:
            # d et n/d divisent n
            k += 2
        d += 1
    return k
```

Ça a l'air correct mais il convient de tester au moins sur quelques exemples :

```
>>> diviseurs(21) # diviseurs : 1, 3, 7 et 21
4
>>> diviseurs(9) # diviseurs : 1, 3 et 9
2
>>> # Oups...
```

Bug : il faut revoir la fonction...

Problèmes :

1. Corriger un bug risque d'en introduire un autre (risque $\approx 10\%$ peut monter à 30% pour une correction de correction buguée)
2. Personne n'a le courage de refaire tous les tests après chaque correction.
3. Ce serait indigne d'un être humain.

1.2 Tests unitaires automatisés

Module python `unittest` : permet d'automatiser des tests.

Usage :

```
import unittest
class MesTests(unittest.TestCase):
  def test_diviseurs_21(self):
    self.assertEquals(diviseurs(21), 4)
  def test_diviseurs_9(self):
    self.assertEquals(diviseurs(9), 3)
unittest.main(exit=False)
```

Exécution du programme :

```
>>> ================================ RESTART ================================
>>>
.F
======================================================================
FAIL: test_diviseurs_9 (__main__.MesTests)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/judicael/Documents/mpsi/ipt/cours/14-python-avance/tests.py"
    self.assertEquals(diviseurs(9), 3)
AssertionError: 2 != 3

----------------------------------------------------------------------
Ran 2 tests in 0.007s
```

```
FAILED (failures=1)
```

1. Résultats possibles pour chaque test : *Success*, *Failure*, *Error*.
2. Résumé 1ère ligne : «.» pour *Success*, «F» *Failure*, «E» *Error*.
3. Description détaillée des tests en échec ou en erreur.
4. Résultat final : `FAILED` (suivi du compte) ou `OK`.

NB :

1. Le nom `MesTests` n'a aucune importance.
2. Le nom des méthodes effectuant les tests (`test_diviseurs_21`, `test_diviseurs_9`) doit commencer par `test`.

1.3 Organisation pratique des tests

En général :

- fichier contenant les fonctions à tester : `prog.py` ;
- fichier contenant les tests : `testeur.py` ;
- depuis IDLE, on exécute `testeur.py` après chaque modification de `prog.py`.

NB : dans `testeur.py`, importer les fonctions de `prog.py` :

from `prog` **import** `f, g, h`

Sortie lorsque tout va bien :

```
.....................................................................
----------------------------------------------------------------------
Ran 66 tests in 1.144s

OK
```

2 Fonctions de première classe

2.1 Motivation

Calculer la somme des carrés des entiers de $[\![0, n[\![$:

```
def somme_carres(n):
    s = 0
    for i in range(n):
        s += i**2
    return s
```

Et si on veut les cubes ? Ou la somme des sinus ? Ou . . .

2.2 Passer une fonction en argument

```
from math import sin
def somme(f, n):
  s = 0
  for i in range(n):
    s += f(i)
  return s
somme(sin, 10)
def cube(n):
  return n**3
somme(cube, 42)
```

Rem : Devoir introduire la fonction `cube` est un peu pénible.

2.3 Fonctions anonymes

On n'est pas obligé de donner un nom à $n \mapsto n^3$.

Notation :

```
>>> lambda n: n**3
<function <lambda> at 0x7faa26d992a8>
```

Exemple :

```
>>> somme(lambda n: n**3, 42)
741321
```

2.4 Retourner une fonction

«Pour $n \in \mathbb{N}$, on note $f_n : x \mapsto x^n e^x$.»

$f : \mathbb{N} \to \mathbb{R}^{\mathbb{R}}$.

```
from math import exp

def f(n):
    def g(x):
        return x**n * exp(x)
    return g

somme(f(3), 17)
```

3 Listes en compréhension

(*List comprehensions*)

Notation pour définir une liste à partir d'une autre liste :

```
>>> [ 2 * i for i in range(5) ]
[0, 2, 4, 8]
>>> [ 2 * i for i in range(10) if i % 3 == 0 ]
[0, 6, 12, 18]
```

Très pratique. Exemple, calcul de la somme des cubes :

```
>>> sum([ i ** 3 for i in range(42) ])
741321
```