Les tests avec Python

Loïc Gouarin

Laboratoire de mathématiques d'Orsay

10 décembre 2010

Plan

Plan

Les différents tests

Tests unitaires : niveau 0

Le but est de tester chaque petit bout de code : fonctions, méthodes, ...

Ils permettent d'être sûr que chaque brique de votre programme fonctionne correctement indépendemment des autres.

Néanmoins, ils ne permettent pas d'assurer le bon fonctionnement du programme dans sa globalité.

Les différents tests

Tests d'intégration : niveau 1

Le but est de commencer à tester de petites interactions entre les différentes unités du programme.

Ces tests peuvent être réalisés avec les mêmes outils que ceux utilisés dans les tests unitaires.

Mais il y a une différence importante : on suppose que les unités prises une à une sont valides.

Les différents tests

Tests du système complet : niveau 2

Le but est de tester le programme dans sa globalité.

On assemble à présent toutes les briques pour un problème concret.

Là encore, si les 2 premiers niveaux sont négligés les tests du système complet ne servent à rien.

Pourquoi écrire des tests ?

- s'assurer que la fonction ou la méthode réagit correctement (bons résultats, erreur si on sort du cadre de sa fonction,...),
- éviter de chercher un bug pendant des heures,
- donner toutes les spécificités que l'on souhaite,
- peut servir de démonstration aux autres développeurs et aux utilisateurs,
- ...

Remarque : habituez vous à écrire vos tests le plus tôt possible ! !

Les modules Python

- unittest,
- doctest,
- nose,
- ...

Plan

Présentation

- Recherche dans les sources des bouts de texte qui ressemblent à une session interactive Python.
- Recherche dans des fichiers textes des bouts de texte qui ressemblent à une session interactive Python.
- Exécution de ces bouts de session pour voir si le résultat est conforme.

Exemple

monmodule.py

```
def addition(a, b):
    """
    renvoie l'addition de 2 nombres

    >>> addition(4, 5)
    9
    """
    return a + b

if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Plan

Présentation

- appelé également PyUnit
- reprend l'esprit de JUnit
- ce module supporte
  - les tests automatiques
  - les fonctions d'initialisation et de finalisation pour chaque test
  - l'aggrégation des tests
  - l'indépendance des tests dans le rapport final

unittest.TestCase

- Les tests doivent faire partie d'une classe héritée de la classe unittest.TestCase.
- Les noms des méthodes de cette classe doivent avoir le prefixe test pour être considérés comme tests.
- Les tests sont exécutés par ordre alphabétique.
- La fonction exécutée avant chaque test doit avoir le nom setUp.
- La fonction exécutée après chaque test doit avoir le nom tearDown.

Exemple

testmonmodule.py

```
import unittest
from monmodule import *

class TestSimple2(unittest.TestCase):
    def setUp(self):
        self.a = 4.
        self.b = 5.

    def test01_Division(self):
        self.assertEqual(division(self.a, self.b), self.a/self.b)

    def test02_DivisionByZero(self):
        self.assertRaises(ZeroDivisionError, division, self.a, 0.)

    def test03_Division(self):
        self.assertAlmostEqual(division(1., 3), 0.33333, 5)

if __name__ == '__main__':
    unittest.main()
```

unittest.TestSuite

Exemple

```
import unittest

def allTests():
    from testSimple import TestSimple
    from testSimple2 import TestSimple2

    suite = unittest.TestSuite()
    suite.addTest(unittest.makeSuite(TestSimple))
    suite.addTest(unittest.makeSuite(TestSimple2))

    return suite

if __name__ == '__main__':
    unittest.TextTestRunner(verbosity=2).run(allTests())
```

Plan

Présentation

- reconnaît automatiquement les tests doctest et unittest.
- offre beaucoup plus de possibilités :
  - tests de couverture,
  - tests de profiling,
  - ensemble de plugins,
  - ...

Commandes utiles

- `nosetests -v`
- `nosetests -v --with-doctest`
- `nosetests -v --with-coverage --cover-html`
- `nosetests -v -A='full'`