Informatique 3

Chapitre 5 – Test unitaire (Unit Testing)

Plan du cours

1. Algorithmes de tri, complexité algorithmique
2. Récursivité
3. Gestion des fichiers
4. Interface en ligne de commande
5. **Test unitaire (Unit Testing)**
6. Récupération des données d'une page Web
7. Utilisation d'API d'un ou de plusieurs services : Twitter/Youtube/GoogleMaps

Chapitre 5 : Unit Testing

Unit Testing

- Principe
- Fonctionnement
- Exécution d'un fichier test
- Méthodes assert
- Découverte automatique des fichiers test
- Exemples
- Liens utiles

Unit Testing - Principe

- Le **test unitaire**, ou Unit testing (**U.T.)** ou **test de composants** est une procédure permettant de vérifier le bon fonctionnement d'une partie d'un programme (une fonction par exemple).

- Avec le U.T on peut faire correspondre un état d'entrée donné à un résultat ou à une sortie. Le test permet de vérifier que la relation d'entrée/sortie donnée par la spécification est bien réalisée.

Unit Testing - Exemple

- Supposons qu'on a une fonction qui fait la somme de deux nombres, définie dans le fichier « operation.py »

```
def add(a,b):
    return a+b
```

- Pour tester cette fonction, on impose que la somme de 6 et 6 doit être égale à 12, alors on fait :

```
import unittest #module du unittesting
from operations import * #module qui contient le code a tester

#definir une classe qui herite de la classe TestCase du module unittest
class OperationsTest(unittest.TestCase):

    # methode qui permet de tester la fonction add,commence par test
    def test_Addition(self):
        self.assertTrue(add(6,6)==12)
```

Unit Testing - Fonctionnement

- importer le module « unittest »
- importer le module qui contient les fonctions à tester
- définir une classe qui hérite de la classe « TestCase » du module « unittest »
- définir dans cette classe des méthodes dont le nom commence par « test »
- ajouter à ces méthodes, des méthodes « assert » pour prévenir le système d'une erreur dans le code à tester
  - Exemples:

```
self.assertEqual(add(6,6), 12)
self.assertTrue(add(6,6)==12)
```

Unit Testing – Exécuter le test

- Pour exécuter le test, on fait :
  - python –m unittest testop.py
  - python –m unittest –v testop.py (pour plus de détails)

```
C:\Users\706390\Desktop\Python3\LigneCommande\unittesting>python -m unittest testop.py
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK

C:\Users\706390\Desktop\Python3\LigneCommande\unittesting>
```

```
C:\Users\706390\Desktop\Python3\LigneCommande\unittesting>python -m unittest -v testop.py
test_Addition (testop.OperationsTest) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Unit Testing – Les méthodes assert

Méthode	Vérifie que
assertEqual(a, b)	a == b
assertNotEqual(a, b)	a != b
assertTrue(x)	bool(x) is True
assertFalse(x)	bool(x) is False
assertIs(a, b)	a is b
assertIsNot(a, b)	a is not b
assertIsNone(x)	x is None
assertIsNotNone(x)	x is not None
assertIn(a, b)	a in b
assertNotIn(a, b)	a not in b
assertIsInstance(a, b)	isinstance(a, b)
assertNotIsInstance(a, b)	not isinstance(a, b)

Unit Testing – Tester les méthodes de la classe str

- Tester la méthode upper(). Les 2 tests passent avec succès

```
import unittest

# python -m unittest test0.py

# methode recommendee pour maintenant:
# python -m unittest -v test0.py

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('toto'.upper(), 'TOTO')

    def test_isupper(self):
        self.assertTrue('TOTO'.isupper())
        self.assertFalse('Toto'.isupper())
```

Unit Testing – Tester les méthodes de la classe « str »

- Le premier test conduit à un échec

```
import unittest
# result: 2 tests: 1 failure
class TestWrongStringMethods(unittest.TestCase):
    #les fonctions doivent commencer par test
    def test_wrong_upper(self):
        # string.upper ne retourne pas un entier!
        self.assertEqual('toto'.upper(), 3)
    def test_isupper(self):
        self.assertTrue('TOTO'.isupper())
        self.assertFalse('Toto'.isupper())
```

```
C:\Python34\unittesting>python -m unittest test1.py
.F
======================================================================
FAIL: test_wrong_upper (test1.TestWrongStringMethods)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Python34\unittesting\test1.py", line 11, in test_wrong_upper
    self.assertEqual('toto'.upper(), 3)
AssertionError: 'TOTO' != 3

----------------------------------------------------------------------
Ran 2 tests in 0.000s

FAILED (failures=1)
```

Unit Testing – Tester les méthodes de la classe « list »

```
import unittest


class TestListMethods(unittest.TestCase):
    def test_noItems(self):
        liste = []
        self.assertEqual(len(liste), 0)
    def test_append(self):
        liste = []
        liste.append('a')
        self.assertEqual(len(liste), 1)
        self.assertEqual(liste[0],'a')
        ###
        self.assertIn('a',liste)
        self.assertNotIn('b',liste)
        ###
    def test_IndexOutOfBound(self):
        liste = []
        with self.assertRaises(IndexError):
            liste[1]
```

Permet de vérifier la production de l'erreur « IndexError » au cas où on référence un élément qui n'existe pas

Unit Testing – plus de tests

```
import unittest

def f():
    # fonction qui ne retourne rien
    pass
def g():
    return 1

class TestMethods(unittest.TestCase):
    def test_is_instance(self):
        notes = {'x':10, 'y':20}
        self.assertIsInstance(notes,dict)
        self.assertNotIsInstance(notes,list)
    def test_none(self):
        a = f()
        b = g()
        self.assertIsNone(a)
        self.assertIsNotNone(b)
```

Notes est une instance de la classe « dict »

La fonction f ne retourne rien

Unit Testing – plus de tests

```python
import unittest


class TestIdentity(unittest.TestCase):

    def test_is(self):
        notes = {'x':10, 'y':20}
        notes2 = notes
        notes3 = dict(notes2)

        self.assertIs(notes,notes2)

    def test_is_not(self):
        notes = {'x':10, 'y':20}
        notes2 = notes
        notes3 = dict(notes2)

        self.assertIsNot(notes,notes3)
```

```
>>> id(notes)
58667176
>>> id(notes2)
58667176
>>> id(notes3)
63564496
```

id : adresse de l'objet en mémoire

Unit Testing – découverte automatique

- La commande : python -m unittest discover -v permet de découvrir et d'exécuter tous les fichiers dont le nom commence par « test »

```
C:\Python34\unittesting>python -m unittest discover -v
test_isupper (test0.TestStringMethods) ... ok
test_upper (test0.TestStringMethods) ... ok
test_isupper (test1.TestWrongStringMethods) ... ok
test_wrong_upper (test1.TestWrongStringMethods) ... FAIL
test_IndexOutOfBound (test2.TestListMethods) ... ok
test_append (test2.TestListMethods) ... ok
test_noItems (test2.TestListMethods) ... ok
test_is_instance (test4.TestMethods) ... ok
test_none (test4.TestMethods) ... ok
test_is (test5.TestIdentity) ... ok
test_is_not (test5.TestIdentity) ... ok
test_Addition (testop.OperationsTest) ... ok
test_Multiplication (testop.OperationsTest) ... ok
```

Liens utiles

- https://docs.python.org/3.4/library/unittest.html
- https://fr.wikipedia.org/wiki/Test_unitaire
- https://en.wikipedia.org/wiki/Unit_testing