Tests et Documentation

Python : docutils et reST

Alexis NEDELEC

LISYC EA 3883 UBO-ENIB-ENSIETA
Centre Européen de Réalité Virtuelle
Ecole Nationale d'Ingénieurs de Brest

Objectifs du cours

Problème : Tester et documenter ... avant de coder

- Analyse : on connaît les services, fonctionnalités
- Conception : on sait comment on va faire
- Tests : on défini les “points de passage”
- Documentation :

“Ce que l’on conçoit bien
s’énonce clairement”
Nicolas Boileau (1636-1711)

Tests unitaires et tests fonctionnels

Définitions

- tests unitaires : valider une fonction de manière isolée
- tests fonctionnels : se mettre à la place de l'utilisateur

A quoi ça sert de faire des tests ?

- éprouver son application
- non-régression : éviter le "pourtant ça marchait bien hier !"
- petite perte de temps à écrire ses tests
- gain énorme au déboguage

On privilégie le préventif au curatif

Tests unitaires en python

Module de calculs : calculs.py

```
def scalaire(p1,p2) :
    return p1[0]*p2[0]+ p1[1]*p2[1]+ p1[2]*p2[2]
```

Définition des tests : test_calculs.py

```
import unittest
import calculs
class TestCalculs (unittest.TestCase) :
    def test_scalaire1(self) :
        self.assertEquals(calculs.scalaire([1,0,1],
                                           [0,1,0]), 0)
    def test_scalaire2(self) :
        self.assertEquals(calculs.scalaire([1,1,1],
                                           [0,1,0]), 1)
```

Tests unitaires en python

Définition des tests : test_calculs.py

```
def test_serie() :
    tests =[unittest.makeSuite(TestCalculs)]
    return unittest.TestSuite(tests)

if __name__ == "__main__" :
    unittest.main(defaultTest="test_serie")
```

Exécution des tests: test_calculs.py

```
$ python test_calculs.py
..
----------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Tests unitaires en python

Documentation : y en a pas !

Solution : la faire en même temps que les tests

- python : `doctring,doctest,docutils`
- reST : `reStructuredText`, langage à balise WYSIWYG

`doctest : test_calculs.py`

```
if __name__=="__main__" :
    import doctest
    doctest.testfile("test_calculs.txt")
```

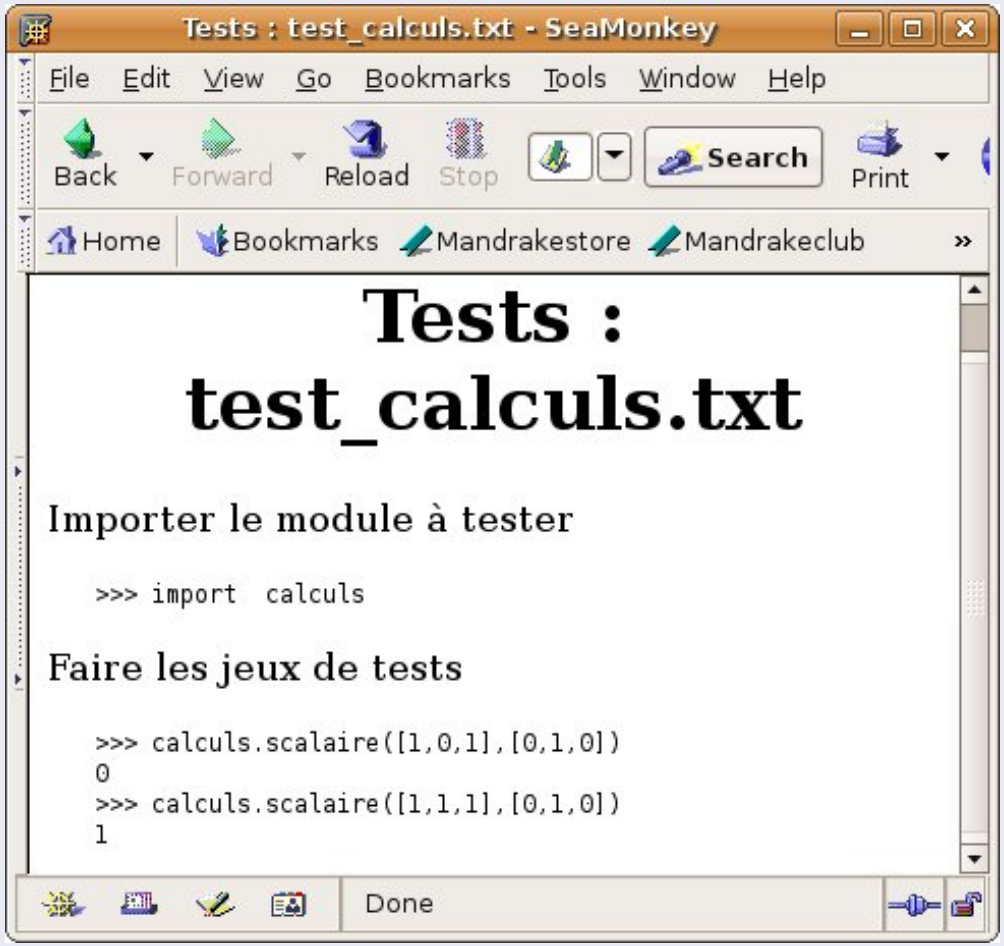
Tests unitaires et documentation en python

reStructuredText : test_calculs.txt

```
==================================
Tests : test_calculs.txt
==================================

Importer le module à tester

>>> import  calculs

Faire les jeux de tests

>>> calculs.scalaire([1,0,1],[0,1,0])
0
>>> calculs.scalaire([1,1,1],[0,1,0])
1
```

python : tests unitaires et documentation

documentation : `rst2html.py calculs.txt calcul.html`



Conversion chiffres romain/nombre entier

Cahier des charges

- une seule façon de représenter un nombre en chiffres romains.
- une chaîne de caractères en chiffres romains doit représenter un nombre valide.
- une chaîne de caractères ne représente qu'un seul nombre.
- on se limite aux nombres de 1 à 3999.
- le zéro n'existe pas.
- on ne représente que des nombres entiers positifs.

Conversion chiffres romain/nombre entier

fonctionnalité : `toRoman()`

- traduire tout entier $n \in [1, 3999]$.
- échec si $n \notin [1, 3999]$.
- échec si valeur non-entière. $n \notin \mathbb{N}$.
- chiffres romains en lettres majuscules.

fonctionnalité : `fromRoman()`

- traduire tout chaîne $\in$ [`'I'`,`'MMMCMXCIX'`].
- échec si chaîne $\notin$ [`'I'`,`'MMMCMXCIX'`].
- chiffres romains en majuscules
- retourne un entier $n \in [1, 3999]$.
- doit être vérifié pour tout entier entre 1 et 3999

Conversion chiffres romain/nombre entier

Spécification des tests

- la réussite : conditions normales d'utilisation
- l'échec : conditions anormales d'utilisation
- la cohérence : réciprocité conversion entier-romain-entier

Première version sans codage : `roman.py`

```
def toRoman(n):
    pass
def fromRoman(s):
    pass
```

Définition des tests : test_roman.txt

Tester la réussite

```
Importer le module à tester

>>> import roman

Initialisation les données à tester

>>> knownValues= ( (1, 'I'),
...               (2, 'II'),
...               (3, 'III'),
...               (4, 'IV'),
...               (5, 'V'),
...               (3999, 'MMMCMXCIX') )
```

Définition des tests : test_roman.txt

Tester la réussite

```
Conditions normales d'utilisation

>>> def testToRoman():
... success = True
...     for integer, numeral in knownValues :
...             result = roman.toRoman(integer)
...             if result != numeral :
...                 success = False
...                 break
...     return success
>>> testToRoman()
True
```

Définition des tests : `test_roman.txt`

Tester l'échec

```
Conditions anormales d'utilisation

>>> roman.toRoman(4000)
Traceback (most recent call last):
    ...
OutOfRangeError: number out of range (must be 1..3999)
```

Définition des tests : `test_roman.txt`

Tester la cohérence

```
Cohérence d'utilisation

>>> for integer in range(1, 4000):
...     numeral = roman.toRoman(integer)
...     result = roman.fromRoman(numeral)
...     assert integer == result
```

Implémentation, passage de tests : roman.py

Tester l'échec et la réussite : toRoman(n)

```
romanNumeralMap = (('M',  1000),
                   ('CM', 900),
                   ('D',  500),
                   ('CD', 400),
                   ('C',  100),
                   ('XC', 90),
                   ('L',  50),
                   ('XL', 40),
                   ('X',  10),
                   ('IX', 9),
                   ('V',  5),
                   ('IV', 4),
                   ('I',  1))
```

Implémentation, passage de tests : roman.py

Tester l'échec et la réussite : toRoman(n)

```
def toRoman(n):
    if not (0 < n < 4000):
        raise OutOfRangeError, \
              "number out of range (must be 1..3999)"
    result = ""
    for numeral, integer in romanNumeralMap :
        while n >= integer:
            result = result + numeral
            n = n - integer
#            print 'subtracting', integer, \
#                  'from input, adding', \
#                   numeral, 'to output'
    return result
```

Passage de tests : roman.py

Tester la réussite : fromRoman(s)

```
def fromRoman(s):
    result,index = 0,0
    for numeral, integer in romanNumeralMap :
        while s[index:index+len(numeral)] == numeral:
            result += integer
            index += len(numeral)
#            print 'found', numeral, \
#                  'of length', len(numeral), \
#                   ', adding', integer
    return result
```

Tester la cohérence

- `fromRoman(toRoman(n))== n` : vrai quel que soit `n`

Documentation : reStructuredText

Ce qu'il faut savoir faire

- structurer un document
- formater le texte
- construire des listes (items)
- présenter des tableaux
- faire des hyperliens
- bas de pages
- ...

Documentation : reStructuredText

Structuration de document

- souligner les titres de section (= - _ : ,) et autres ...
- on peut surligner du texte (visibilité)

```
==================================
Module : roman.py
==================================

~~~~~~~~~

toRoman(n)
~~~~~~~~~

Conversion entier (n) : int -> chiffre romain : string
```

Documentation : reStructuredText

Structuration de document

Donner des exemples de codes :

- il doivent être précédé de `::`
- encadrés par des sauts de ligne
- monotypés : une seule font, une seule taille
- indentés de un à plusieurs caractères

```
Fonction de conversion entier/chiffre romain::

  def toRoman(n) :
       pass

Cette fonction ne fait *rien* **pour l'instant**
```

Documentation : reStructuredText

Les listes

- listes à puces `+, - , *` (`+ bla, bla`)
- listes numérotées à la main (`n. bla bla`)
- listes numérotées automatiquement (`#. bla bla`)

```
Rôle des fonctions :
--------------------
+ Rôles :
 #. traduire tout entier n dans l'intervalle [1,3999]
 #. échouer si l'entier n'est pas à l'intervalle de valeu
 #. échouer si  ce n'est pas une valeur entière.
 #. retourner un chiffres romains en lettres majuscules.
+ paramètres d'entrées :
 #. bla
 #. bla bla
```

Documentation : reStructuredText

Les tableaux : simple table, grid table

```
====  ====
toRoman()
----------
In    Out
====  ====
 1     I
3999  MMMCMXCIX
====  ====
```

```
+-----+-------------+
|   toRoman()       |
+-----+-------------+
| In  |    Out      |
+-----+-------------+
| 1   |  I          |
|3999 |  MMMCMXCIX  |
+-----+-------------+
```

Documentation : reStructuredText

Les liens

- hyperliens externes
- hyperliens internes
- notes de bas de pages

```
Références :
--------------------
Ce programme est inspiré du "livre libre" :
‘Dive Into Python‘_ de Mark Pilgrim [#]_
ainsi que Giddo Van Rossum  [#]_ créateur de python
Retour en début de page ‘Module : roman.py‘_

.. _‘Dive Into Python‘ :  http://diveintopython.org/
.. [#] que nous remercions au passage
.. [#] que nous remercions aussi au passage
```

Documentation : reStructuredText

Autres

- directives : ajouter de nouvelles commandes
- substitution : associer un mot à une commande
- fields : “métadonnées” à ajouter au document

```
.. contents:: Table des matières
.. image:: logenib.jpg

.. |enib| image:: logenib.jpg
.. |date|  date::
.. |time|  date:: %H:%M
Bienvenue à l'ENIB |enib| le  |date| à  |time|

:Date: |date|
:Time: |time|
```

Bibliographie

Ouvrages

- **T. Ziadé** “Python : petit guide à l’usage du développeur agile” ed. Dunod, 2007
- Mark Pilgrim :“plongez au cœur de Python” GNU Free Documentation License

Liens Au Net

- Mark Pilgrim : `http://diveintopython.org`
- doctutils : `http://docutils.sourceforge.net`
- reST : `http://docutils.sourceforge.net/rst.html`