Tester ses programmes

Tester ses programmes

De toute évidence, le code qu'on écrit n'a aucune assurance de fonctionner si on ne le teste pas...

Il existe plusieurs approches en Python pour s'assurer qu'un code fonctionne.

Écrire les tests soi même

Écrire les tests soi même

Le moyen le plus simple consiste à écrire un jeu de test après `if __name__="__main__":`

```
def ma_fonction(n):
  ...


if __name__ == '__main__':
  print(ma_fonction(5))
```

C'est généralement ce qu'on fait quand on développe. Ces tests doivent couvrir tous les cas possibles et être compréhensibles.

Selon les contextes (devoir, projet, développement en cours...) on peut les laisser ou les effacer.

Il est préférable de les remplacer par de vrais tests...

Assert

Assert

Python intègre un mot clef `assert` qui va lever une exception `AssertionError` si la condition qui suit est fausse:

```
>>> assert 1 == 1 # ne fait rien


>>> assert 1 == 2 # plante le programme
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
```

C'est le moyen le plus efficace et rapide de tester un programme ou une fonction.

Il ne faut pas intégrer les assertions à la fonction elle même. Il est préférable de les intégrer à des fonctions de tests indépendantes du programme.

Un exemple

Reprenons notre fonction Fibonacci

```
def fibonacci(n):
    '''
    Liste des termes de la suite de Fibonacci de l'indice 0 
    @param n: (int) l'indice maximal voulu
    @return: (list) la liste des termes
    '''
    if type(n) != int or n < 0:
        return None
    x = 1
    y = 1
    suite_fibonacci = [x]
    indice = 0
    while indice < n:
        x, y = y, x + y
        suite_fibonacci.append(x)
        indice += 1
```

Doctest

Doctest

Python permet grâce au module `doctest` d'intégrer les tests à la documentation.

Il est parfois délicat de tester certaines fonctions, en particulier les affichages.

Pour les fonctions qui réalisent des calculs cela est pratique.

Un exemple :

```
def multiply(a, b):
    """
    Calcule produit de a et b
    @param a: (number, str, list) premier facteur
    @param b: (number, str, list) second facteur
    @return: (number, str, list) le produit

    >>> multiply(4, 3)
    12
    >>> multiply('a', 3)
    'aaa'
    """
    return a * b


if __name__ == '__main__':
  import doctest
  doctest.testmod() # s'il ne se passe rien, les test sont
```

Quand on exécute le programme, il ne se passe rien

Un exemple qui échoue :

```
def Fonction_mal_testee():
    '''
    Simple fonction qui echoue
    >>> Fonction_mal_testee()
    3
    '''
    return 2


if __name__ == "__main__":
    import doctest
    doctest.testmod()  # s'il ne se passe rien, les tests
```

Voici la sortie d'un exemple qui échoue

```
>>> python3 2_tester_doctest.py
**********************************************************
File "/home/quentin/realiser_des_tests/2_tester_doctest.py"
in __main__.Fonction_mal_testee
```

Unitest, tesmod

Unitest, tesmod

Il existe une librairie dédiée aux tests : `Unitest` et qui permet de tester toutes les propriétés possibles d'un objet.

Elle est un peu vaste et trop complexe pour nos objectifs aussi nous ne l'utiliserons pas.

Voici sa documentation et un guide détaillé.