RÉSUMÉ 12 - DOCTESTS : DOCUMENTER ET TESTER

Lien vers l'activité : Doctests : documenter et tester
www.infoforall.fr - Dernière modif. : 26 10 2020

12.1 DÉMARCHE HABITUELLE DE CRÉATION DE FONCTIONS :

1 - ON ÉCRIT OU REÇOIT LA DEMANDE DOCUMENTÉE

```
def note_valide(note) :
    '''Renvoie True si note est dans l'intervalle [ 0 ; 20 ]

    :param note: la note à tester
    :type note: int or float

    :return:: True si note dans [ 0 ; 20 ]
    :rtype: bool

    '''
    return True
```

2 - ON ÉCRIT UNE BASE PERTINENTE DE TESTS DOCUMENTÉS

Ces tests doivent permettre :

- de comprendre la fonction d'utiliser la fonction
- de vérifier les cas un peu particulier qu'on a prédétecté comme problématique

```
def note_valide(note) :
    '''Renvoie True si note est dans l'intervalle [ 0 ; 20 ]

    :param note: la note à tester
    :type note: int or float

    :return:: True si note dans [ 0 ; 20 ]
    :rtype: bool

    :Exemple:

    >>> note_valide(5)
    True
    >>> note_valide(-5)
    False
    >>> note_valide(25)
    False
    >>> note_valide(20.0)
    True

    '''
    return True
```

3 - ON ÉCRIT LE CODE

Et encore... Dans bien des cas, on commence par établir l'algorithme avant de passer à l'implantation dans un langage particulier.

```python
def note_valide(note) :
    '''Renvoie True si note est dans l'intervalle [ 0 ; 20 ]

    :param note: la note à tester
    :type note: int or float

    :return:: True si note dans [ 0 ; 20 ]
    :rtype: bool

    :Exemple:

    >>> note_valide(5)
    True
    >>> note_valide(-5)
    False
    >>> note_valide(25)
    False
    >>> note_valide(20.0)
    True

    '''
    if note > 20 :
        return False
    elif note < 0 :
        return False
    else :
        return True
```

4 - ON RAJOUTE L'ACTIVATION DU DOCTEST

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

Si on désire un mode plus **verbeux** contenant plus d'informations, on utilisera ceci :

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

12.2 ASSERT

Pour rendre une condition plus contraignante qu'une simple indication dans une documentation, on peut utiliser une **assertion**.
Pour cela, on utilise le mot-clé **assert**.
Son utilisation basique se résume à une succession de trois éléments :

- Le mot-clé **assert**
- Un espace suivi de la condition qu'on veut vérifier sous forme d'une expression booléenne
- Une virgule suivi du texte à afficher si l'expression est évaluée à False

En résumé :

```python
assert note >= 0 and note <= 20, "Pas dans [0;20] !"
```

12.3 ASSERT POUR RÉALISER DES TESTS

Finalement, on peut même se passer des modules de tests automatiques et n'utiliser que des asserts contenus dans une fonction qu'on lancera pour tester nos fonctions.

```python
def addition(a, b) :
    return a+b

def test_maison() :
    assert addition(10,5) == 15, 'Echec du test : addition(10,5) == 15'
    assert addition(10,-5) == 5, 'Echec du test : addition(10,5) == 15'

if __name__ == '__main__' :
    test_maison()
```