Fonctions et tests Unitaires

Sauvegarder tous les programmes individuellement dans un répertoire nommé « tests-unitaires »

1. Les assertions

En Python un « `assert` » est une aide au débogage qui vérifie des conditions. Si la condition n'est pas vérifiée alors une *`AssertionError`* est soulevée avec, si besoin, un message d'erreur.

Les tests unitaires sont corrects	Les tests unitaires sont erronés
`"""` *`Fonction linéaire`* `"""` **`def fctLineaire(x):`** **`    return x * 2`** **`assert fctLineaire(2)==4`** **`assert fctLineaire(5)==10`**	`"""` *`Fonction linéaire`* `"""` **`def fctLineaire(x):`** **`    return x * 2`** **`assert fctLineaire(2)==4`** **`assert fctLineaire(5)==12`**

Résultat affiché dans la console
`*** Console de processus distant Réinitialisée ***`
`>>>`
	Résultat affiché dans la console :
	*** Console de processus distant Réinitialisée ***
	Traceback (most recent call last):
	File "E:\python mooc\terminale\Sujet 0\pratique\21_NSI_02\21_NSI_EX1.py", line 8, in <module>
	assert fctLineaire(5)==12
	AssertionError
	>>>

Rien n'est affiché Les tests unitaires sont corrects	Effectivement 2x5 n'est pas égal à 12. Un test unitaire n'est pas correct.

Exercice 1 : (D'après un sujet pratique NSI)

Écrire une fonction qui prend en paramètre un tableau d'entiers non vide et qui renvoie la moyenne de ces entiers. La fonction est spécifiée ci-après et doit passer les assertions fournies.

```
def moyenne (tab):
    """
    moyenne(list) -> float
    Entrée : un tableau non vide d'entiers
    Sortie : nombre de type float
    Correspondant à la moyenne des valeurs présentes dans le tableau
    """

assert moyenne([1]) == 1
assert moyenne([1,2,3,4,5,6,7]) == 4
assert moyenne([1,2]) == 1.5
```

2. Les doctests

Exercice 2 : Fonction linéaire (Exemple corrigé)

Les tests unitaires sont corrects	Les tests unitaires sont erronés

"""
Fonction linéaire
"""
import doctest
 	
def fctLineaire(x):
  """
  fonction lineaire
  retourne le résultat de 2x
  Argument:
  x -- nombre
  >>> fctLineaire(2)
  4
  >>> fctLineaire(5)
  10
  """
  return x * 2
 	
doctest.testmod()
	"""
	Fonction linéaire
	"""
	import doctest
	
	def fctLineaire(x):
	  """
	  fonction lineaire
	  retourne le résultat de 2x
	  Argument:
	  x -- nombre
	  >>> fctLineaire(2)
	  4
	  >>> fctLineaire(5)
	  12
	  """
	  return x * 2
	
	doctest.testmod()

Résultat affiché dans la console
*** Console de processus distant Réinitialisée ***
>>>
	Résultat affiché dans la console :
	**********
	File "E:\python mooc\terminale\Révisions\tests unitaires\Ex1-fonction.py", line 15, in __main__.fctLineaire
	Failed example:
	  fctLineaire(5)
	Expected:
	  12
	Got:
	  10
	**********
	1 items had failures:
	  1 of 2 in __main__.fctLineaire
	Test Failed 1 failures.
	>>>

Rien n'est affiché Les tests unitaires sont corrects	Effectivement 2x5 n'est pas égal à 12. Un test unitaire n'est pas correct.

Exécuter la fonction linéaire avec les doctests et confirmer les résultats affichés dans la console.

Exercice 3 : Distance d'arrêt d'un véhicule

Pour calculer la distance d'arrêt d'un véhicule, on applique la formule :

distanceArret = reaction + freinage

- « reaction » est la distance parcourue par la voiture pendant le temps de réaction du conducteur ;
- « freinage » est la distance parcourue par la voiture pendant le temps de freinage ;

$$reaction = \frac{vitesse}{3,6} \qquad freinage = \frac{vitesse^2}{200}$$

Remarque :

Pour les différentes fonctions à écrire, vous devez incorporer au moins 2 tests unitaires par fonctions. Les résultats devront être arrondis au mètre près.

3.1. Écrire une fonction « **`reaction(v)`** » qui à partir de la vitesse du véhicule retourne la distance de réaction.

3.2. Écrire une fonction « **`freinage(v)`** » qui à partir de la vitesse du véhicule retourne la distance de freinage.

3.3. Demander la vitesse du véhicule à l'utilisateur et afficher la distance d'arrêt.

3.4. Sur route mouillée, on estime que la distance de freinage est multipliée par 2.
Demander l'état de la route à l'utilisateur et modifier la fonction « **`freinage(v)`** » **pour** calculer la distance de freinage en conséquence.

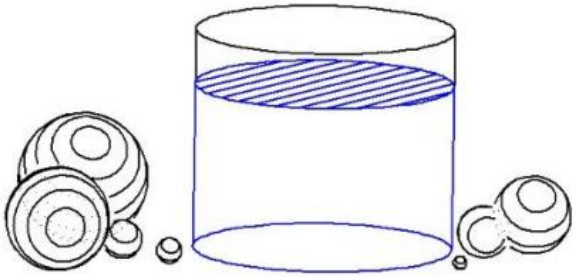
Exercice 4 : Surface d'un disque (Exemple corrigé)

```
"""
calcul la surface d'un disque en fonction du rayon
"""
import doctest
from math import pi


def surfaceDisque(rayon):
    """
    fonction qui retourne la surface d'un disque en fonction du rayon
    argument:
    rayon -- nombre
    >>> surfaceDisque(5)
    78.53981633974483
    """
    return pi*rayon**2

print(surfaceDisque(5))
doctest.testmod()
```

Exercice 5 : Une histoire de volume



Le verre cylindrique et quelques billes

Des billes dans le verre

On considère un récipient cylindrique de 30 mm de rayon et de 58 mm de hauteur. On possède une série de 30 billes : la première bille a un rayon de 1 mm, la seconde de 2 mm, etc...

Le verre est rempli d'eau jusqu'à une hauteur de 50 mm. On néglige l'épaisseur du récipient.

On dépose une par une les billes dans le verre, dans l'ordre croissant de leur rayon et on les laisse dans le verre. On suppose que les billes ne s'empilent pas à la verticale en une colonne mais qu'elles tombent au fond du verre en occupant l'espace disponible. A partir de quelle bille l'eau devrait-elle déborder du verre ?

5.1. Commencer par écrire les deux fonctions suivantes avec au moins un test unitaire.

`def volumeCylindre(rayon,hauteur):`	`def volumeSphere(rayon):`

5.2. Conseil : pour résoudre le problème, utiliser une boucle « tant que » (while).