TD5_fonctions

December 13, 2022

#

NUMERIQUE ET SCIENCES INFORMATIQUES 1ère NSI

0.1 TD les fonctions

[ ]:
```
from metakernel import register_ipython_magics
register_ipython_magics()
```

Une fonction va permettre de compacter le code car elle pourra être utilisée de nombreuses fois en l'appelant. Si la fonction ne renvoie aucun résultat, on l'appelle une procédure.

- Exécutez ce programme et observez les résultats obtenus.

[2]:
```
#%%tutor
# -*- coding: utf-8 -*-
"""
Addition
"""

def addition(x:int,y:int=2)->int: # annotations de typage attendu (non traité␣
 ↪en l'état par l'interpréteur python)
    """
    Additionne les 2 nombres
    passés en arguments
    """
    return x+y # renvoie la somme de x et y

print(addition(10,5)) # appel de la fonction 'addition' et affichage de la somme
resultat=addition(7,3) # appel de la fonction 'addition' et stockage dans une␣
 ↪variable 'resultat'
print(resultat) # affichage de la variable 'resultat'
print(addition(5)) # appel de la fonction 'addition' avec un seul argument, la␣
 ↪fonction utilise la valeur de y par défaut
```

```
15
10
7
```

[39]:
```
#fonction division
def division(x:int,y:int)->float:
    """
    Fonction qui prend en paramètres 2 entiers et renvoie la valeur de la␣
 ↪division du 1er nombre par le 2ème
    """
    #if y == 0:
        #print("Y ne doit pas être égal à 0 !")
        #return
    #assert y != 0, 'Le diviseur doit être différent de 0'
    try:
        print(x,y)
        return x/y
    except ZeroDivisionError:
        print("Le diviseur doit être différent de 0")
```

[41]:
```
division(2,1)
```

```
2 1
```

[41]: 2.0

- Exécutez ce programme et observez les résultats obtenus.

[6]:
```
#%%tutor
# -*- coding: utf-8 -*-
"""
Calcul de volumes
"""

from math import pi

#voici une fonction
def cube(x:float)->float:
    """
    Reçoit en entrée une valeur réelle positive
    Retourne cette valeur à la puissance 3.
    Exemple : x = 3 , la valeur retournée est 9.
    """
    assert x>0,'La valeur doit être supérieure à 0'
    return x**3

#voici une autre fonction
def volume_sphere(r:float)->float:
    """
    Reçoit en entrée une valeur réelle positive
    Retourne le volume d'une spère de rayon égal à la valeur saisie
```

```
    """
    assert r>0,'La valeur doit être supérieure à 0'
    return 4.0 / 3.0 * pi * cube(r)

#Ici débute le programme principal
rayon = float(input("Entrez la valeur du rayon de la sphère en mm : "))
print(f"Le volume de cette sphère vaut : {volume_sphere(rayon):.2f} mm3")
cote = float(input("Entrez la valeur du côté d'un cube en mm : "))
print(f"Le volume du cube vaut : {cube(cote):.2f} mm3")
print(x)
print(r)
```

```
Entrez la valeur du rayon de la sphère en mm : -98
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-6-69eac79710ef> in <module>
     28 #Ici débute le programme principal
     29 rayon = float(input("Entrez la valeur du rayon de la sphère en mm : "))
---> 30 print(f"Le volume de cette sphère vaut : {volume_sphere(rayon):.2f} mm3 )
     31 cote = float(input("Entrez la valeur du côté d'un cube en mm : "))
     32 print(f"Le volume du cube vaut : {cube(cote):.2f} mm3")

<ipython-input-6-69eac79710ef> in volume_sphere(r)
     23     Retourne le volume d'une spère de rayon égal à la valeur saisie
     24     """
---> 25     assert r>0,'La valeur doit être supérieure à 0'
     26     return 4.0 / 3.0 * pi * cube(r)
     27

AssertionError: La valeur doit être supérieure à 0
```

- Que font ces deux fonctions ?
  Elles calculent le volume d'une sphère et d'un cube
- Dans quel ordre et par qui sont appelées les fonctions ?
  la fonction volume_sphere est exécutée, et ensuite, la fonction cube est exécutée.
  Les fonctions sont appelées à travers la fonction print
- Quel est le nom de la variable qui contient le rayon de la sphère dans le programme principal, les fonctions ?
  La variable se nomme "rayon", et les fonctions se nomment cube et volume_sphere
- Quelle est l'instruction qui renvoie la valeur calculée dans la fonction ?
  L'instruction renvoyant la valeur calculée est la fonction print()
- Quel est l'intérêt de définir et d'appeler des fonctions ?
  Cela permet de raccourcir les programmes.
- Que se passe-t-il si en fin de programme, on ajoute l'instruction : print(x) ou print(r) ? Justifiez
  Cela fait des erreurs car r et x ne sont pas définis.

- Que se passe-t-il si vous saisissez une valeur négative ou nulle ?
  Cela fait une erreur : le nombre doit être supérieur à 0
- Tapez ci-dessous l'instruction help(cube), help(volume_sphere)

[8]:
```
help(volume_sphere)
```

```
Help on function volume_sphere in module __main__:

volume_sphere(r:float) -> float
    Reçoit en entrée une valeur réelle positive
    Retourne le volume d'une spère de rayon égal à la valeur saisie
```

Voici l'heure du défi !

Défi

- Écrivez une fonction qui teste si le **nombre** passé en argument de la fonction est **pair**.
- Commentez votre code et la fonction (**docstrings**).
- Testez votre programme en saisissant quelques valeurs.

Remarque :

[30]:
```
"""
fonction pair
    si nombre / 2 est un nb entier
        afficher le nombre est pair
    sinon
        afficher le nombre est impair
fin fonction
"""

def pair(nb):
    """
    Cette fonction vérifie la parité d'un nombre.
    """
    assert type(nb)==int,'Le nombre doit être un entier'
    if nb%2 == 0:
        print(f"le nombre {nb} est pair" )
    else:
        print(f"le nombre {nb} est impair" )
```

[32]:
```
pair(-98)
```

```
le nombre -98 est pair
```

Vous venez de prendre du temps pour développer une fonction. Est-elle correcte ? Vous avez fait des essais, mais y a t-il un cas où la fonction va retourner un résultat faux ?

Après avoir consulté la fonction ci-dessous, exécutez la.

[8]:
```
def test_parite(nb:int)-> bool:
    """
    Cette fonction reçoit en entrée un nombre entier.
    Elle teste si ce nombre est pair
    puis renvoit en sortie 0 ou 1.
    Précondition : Le nombre doit être entier.
    Postcondition : 0 si pair, 1 si impair
    Tests :
    >>> test_parite(8)
    0
    >>> test_parite(7)
    1
    """

    return nb%2
```

[9]:
```
test_parite(45)
```

[9]:
```
1
```

Exécutez maintenant, les 3 cellules de code ci-dessous.

[49]:
```
import doctest
doctest.testmod()
```

[49]:
```
TestResults(failed=0, attempted=2)
```

[50]:
```
import doctest
doctest.testmod(verbose = True)
```

```
Trying:
    test_parite(8)
Expecting:
    0
ok
Trying:
    test_parite(7)
Expecting:
    1
ok
2 items had no tests:
    __main__
    __main__.division
1 items passed all tests:
   2 tests in __main__.test_parite
2 tests in 3 items.
2 passed and 0 failed.
Test passed.
```

[50]:
```
TestResults(failed=0, attempted=2)
```

[3]:
```
help(test_parite)
```

```
Help on function test_parite in module __main__:

test_parite(nb:int) -> bool
    Cette fonction reçoit en entrée un nombre entier.
    Elle teste si ce nombre est pair
    puis renvoit en sortie 0 ou 1.
    Précondition : Le nombre doit être entier.
    Postcondition : 0 si pair, 1 si impair
    Tests :
    >>> test_parite(8)
    0
    >>> test_parite(7)
    1
```

Modifier maintenant le second test de parité : remplacez 1 par 0 et relancez les tests !

[13]:
```
def binaire(n):
    if n > 1:
        binaire(n // 2)
    print(n % 2, end='')
# Demande à l'utilisateur d'entrer un nombre
nb = int(input("Entrez un nombre decimal: "))
binaire(nb)
print("\n")

"""
fonction binaire (n):
    Si n > 1
        binaire(n//2)
    finsi
    afficher n%2, fin ="")
fin fonction

nb <- entrer un nb décimal
binaire(nb)

"""
```

```
Entrez un nombre decimal: 2047
11111111111
```

Pratique pour valider ses fonctions. Vous voyez que le docstring en plus de renseigner l'utilisateur sur le mode d'emploi de la fonction, va permettre de faire des tests.

[ ]: