ISM MPSI/PCSI, Informatique
Année 2023/2024

TP Informatique 5

⚠ Tous les exercices seront traités dans la même feuille de script que l'on nommera par exemple `TP5`, à enregistrer dans votre espace personnel (disque U).

⚠ Commencer par importer `pytest` dans votre feuille de script.

Exécution de pytest

On exécutera `pytest` 5 minutes avant la fin du TP. Pour cela
- ouvrir l'invite de commande en tapant *cmd* dans la barre de recherche windows ;
- aller dans le répertoire contenant votre fichier script `TP5.py` en tapant `U:` suivi éventuellement du chemin du répertoire dans lequel vous avez enregistré le fichier ;
- exécuter le script en tapant `python TP5.py`, puis exécuter `pytest` en tapant `pytest TP5.py`.

Exercice 1

1. Dans l'éditeur, saisir la fonction

```
def f(x):
    if x>0:
        return x
    else:
        return -x
```

   puis la tester dans la console sur différentes valeurs de flottants. Que réalise la fonction `f` ?

2. Écrire une nouvelle version de la fonction `f2` avec un test sans alternative.

Exercice 2

1. Écrire une fonction `arr1(x)` d'argument `x` un flottant qui renvoie x si $|x| > \varepsilon$ et 0 sinon avec $\varepsilon = 10^{-6}$. On écrira un test avec alternative.
2. Écrire une nouvelle version `arr2(x)` qui renvoie le même résultat que `arr1(x)` mais avec un test sans alternative.

Exercice 3

1. Dans l'éditeur, saisir la fonction :

```
def somme(n):
    res=0
    for k in range(n):
        res+=k+1
    return res
```

puis la tester sur différentes valeurs d'entiers. Que réalise la fonction `somme` ?

2. En s'inspirant de l'exemple, écrire une fonction `somme2(n)` qui calcule $\sum_{k=1}^{n} k^2$ pour n entier naturel non nul.
3. Écrire une fonction `test_somme2()` permettant de tester la fonction `somme2` en utilisant l'instruction `assert`.

Exercice 4

1. Dans l'éditeur, saisir la fonction :

```
def somme_liste(L):
    res=0
    for x in L:
        res+=x
    return res
```

puis la tester sur différentes listes de nombres. Que réalise la fonction `somme_liste` ?

2. En s'inspirant de l'exemple, écrire une fonction `prod_liste(L)` d'argument `L` une liste de nombres et qui renvoie le produit de ceux-ci.
3. Écrire une fonction `test_prod_liste()` permettant de tester la fonction `prod_liste` en utilisant l'instruction `assert`.

Exercice 5

1. Écrire une fonction `somme_cube(n)` d'argument `n` un entier naturel non nul et qui renvoie la somme des cubes parfaits inférieurs stricts à n. Le code proposé ne manipulera que des entiers et utilisera une boucle `for`.

   Par exemple `somme_cube(65)` donne $1 + 8 + 27 + 64 = 100$, et `somme_cube(64)` donne $1 + 8 + 27 = 36$.
2. Écrire une fonction `test_somme_cube()` permettant de tester la fonction `somme_cube` en utilisant l'instruction `assert`.

Exercice 6

Écrire une fonction `mult(n,p)` d'arguments `n` et `p` des entiers non nuls et qui renvoie le multiple de `p` le plus proche de `n`. Par exemple, l'appel `mult(31,4)` renvoie `32` tandis que l'appel `mult(29,4)` renvoie `28`. L'appel de `mult(30,4)` peut renvoyer indifféremment `28` ou `32`, les deux réponses étant correctes.

Exercice 7

Un triplet pythagoricien est un triplet d'entiers naturels (m, n, p) **non nuls** tels que $m \leqslant n$ et $m^2 + n^2 = p^2$. Par exemple, $(3, 4, 5)$ est un triplet pythagoricien car $3^2 + 4^2 = 5^2$.
Proposer une fonction `pytha(N)` prenant en argument un entier naturel N et la liste des triplets pythagoriciens `[m,n,p]` tels que $p \leqslant$ N. On les affichera par ordre croissant de p.
On vérifiera que l'appel `pytha(20)` renvoie `[[3, 4, 5], [6, 8, 10]]`.