Introduction

L'objectif de ce TD est de découvrir les fonctions en Python. On pourra pour cela s'appuyer sur :

— le cours ;
— le formulaire du cours ;
— la documentation Python accessible par la commande `help('def')` ;
— https ://docs.python.org/fr/3/tutorial/controlflow.html#defining-functions

Les fonctions permettent de décomposer un programme en sous-programmes désignés par un nom afin de pouvoir les appeler, c'est-à-dire exécuter les instructions qu'elles contiennent, dans le programme principal ou dans d'autres fonctions. Le programme principal ou la fonction appelante, interrompt alors son exécution le temps que les instructions de la fonction soient exécutées.

Les fonctions peuvent avoir zéro ou plusieurs paramètre(s). Les paramètres permettent de passer des données depuis la fonction appelante vers la fonction afin qu'elle puisse effectuer ses instructions.

Structurer le code en définissant des fonctions est une bonne pratique de programmation. En effet, cela permet d'organiser le code en parties élémentaires qui pourront :

1. être appelées plusieurs fois : le code est écrit une fois pour toutes ;
2. être rassemblées dans des bibliothèques de fonctions appelées modules : d'autres programmeurs ou vous-même pourront aussi faire appel à vos fonctions [1].

Il est à noter que les fonctions peuvent aussi s'appeler elles-même, on parle alors de *fonction récursives*.

Python prédéfinit un ensemble de fonctions appelées fonctions natives. Celles-ci sont données dans le formulaire et sont accessibles depuis n'importe quel programme. D'autres fonctions sont définies au sein de modules qui doivent être importés. C'est le cas par exemples des fonctions mathématiques définies dans le module `math`.

Exercice 1 : appel de fonctions prédéfinies

1. Créez une liste de réels, comme par exemple :

```
1 ma_liste = [5.4, 6.3, 1.2, 5.4, -1.2, -8.5]
```

2. À l'aide des fonctions natives de Python :
   (a) déterminez la valeur minimale, la valeur maximale et la somme de la liste,
   (b) Affichez la liste triée dans l'ordre décroissant.

1. Pour cela, celles-ci devront avoir été bien documentées.

Exercice 2 : définition d'une fonction

1. Reprenez le script de conversion de miles par heures en kilomètre par heure vu dans un td précédent mais cette fois-ci :
   — en définissant une fonction `mph2kph`,
   — et en appelant la fonction depuis une fonction principale `main`,
2. À partir de l'étude du formulaire Python, rajouter des commentaires (docstrings) afin de décrire ce que fait la fonction, de définir ces paramètres ainsi que sa valeur de retour ;
3. Rajouter des annotations pour indiquer que le type des paramètres d'entrée et de sortie doivent être réels ;
4. Rajouter des assertions afin de vérifier que la vitesse donnée en argument est positive et que le type de l'argument est un réel ;
5. Écrivez sur le même principe la fonction `kph2mph` qui fait l'opération inverse.

Exercice 3 : création d'un module

1. Définissez un module `conversion` dans lequel vous aurez intégré les deux fonctions précédentes ;
2. Dans un autre fichier `test_conversion` écrivez un programme qui vous permet d'appeler les fonctions de ce module.

Exercice 4 : définition d'une fonction avec paramètres par défaut

La fonction sinus cardinal (`sinc`) peut être définie de 2 manières. Elles sont respectivement données par les équations suivantes :

$$sinc(x) = \frac{\sin(x)}{x} \tag{1}$$

$$sinc(x) = \frac{\sin(\pi x)}{\pi x} \tag{2}$$

La première est la définition mathématique, la seconde est utilisée en traitement du signal.

1. Écrivez une fonction `sinc` qui prend 2 paramètres en entrée : x et un paramètre optionnel booléen *formule_math* qui est vrai par défaut mais qui calcule la deuxième définition lorsque le paramètre est faux. Bien entendu, vous documenterez la fonction comme il se doit avec des commentaires spéciaux et vous pourrez aussi ajouter des annotations.

Exercice 5 : notions de tests unitaires

1. Dans un fichier `my_math.py`, écrivez une fonction `signe` ayant un seul paramètre x qui renvoie 1 si la paramètre x est strictement positif, -1 s'il est strictement négatif, 0 s'il est nul et `None` si l'argument n'est pas un nombre ;
2. En envisageant plusieurs arguments possibles, testez le bon fonctionnement de la fonction `signe` dans un script `test_my_math.py` ;
3. rajouter une fonction `abs` ayant un seul paramètre x qui renvoie la valeur absolue de x si x est un nombre entier ou réel, le module de x si x est un complex et None pour tout autre type ;
4. rajouter des tests dans le script `test_my_math.py` afin de vérifier le bon fonctionnement de votre fonction.

Remarque : il existe des outils tels que les modules Python `unittest` et `pytest` qui permettent d'automatiser les tests de fonction. On appelle ces outils *outils de tests unitaires*.

Indication :

— la fonction native `isinstance` permet de vérifier si une variable est une instance d'un type. Un exemple d'utilisation vous est donné dans la section sur les fonction du formulaire Python.

Exercice 6 : fonctions récursives

```python
def fonction1(n):
    if n == 1:
        return n
    else:
        return n + fonction1(n - 1)

def main():
    print(fonction1(6))

if __name__ == "__main__":
    main()
```

1. Analyser le code ci-dessus et indiquez ce qu'affiche le programme principal dans le terminal ;
2. En vous inspirant du code précédent, écrivez un programme qui permet de calculer la fonction factorielle :

$$n! = n \times (n-1) \times \cdots \times 1 \tag{3}$$

avec $0! = 1$.