TP4 - Fonctions

M. Tellene

1 Création de l'environnement de travail

Avant de commencer le TP, vous allez créer votre environnement de travail. Pour ce faire vous allez dans votre dossier personnel, puis « apprentissage_python ». Une fois arrivé, créer un dossier « TP4 ». C'est dans ce dossier qu'il faudra mettre tous les exercices du TP.

2 Introduction

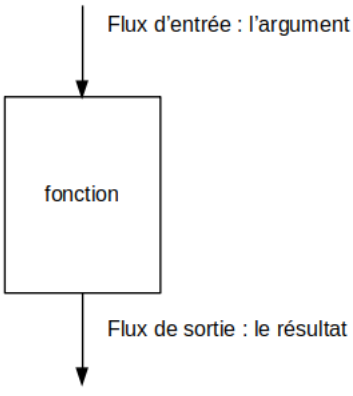
2.1 Une fonction, qu'est-ce que ?

fonction n.f. (lat. *functio*)

1. Ensemble d'opérations concourant au même résultat et exécutées par un organe ou un ensemble d'organes
2. Rôle exercé par quelqu'un au sein d'un groupe
3. (MATH.) Relation qui, à chaque élément de son ensemble de départ, associe au plus une image
4. (INFO) Ensemble d'instructions constituant un sous-programme identifié par un nom, qui se trouve implanté en mémoire morte ou dans un programme
5. (TECH.) Rôle joué par un sous-ensemble dans un système, service rendu par un système

2.2 Caractéristiques d'une fonction

Une fonction se caractérise dans un premier temps par son **nom**. Celui-ci doit refléter au mieux le service rendu.
Dans un deuxième temps, il faut connaître le type de **flux** sur lequel la tâche est réalisée. Ce flux peut être de la matière, de l'énergie, des personnes,...
Dans notre cas, le flux sera sous forme d'informations et le service rendu est un traitement de ces informations. Nous distinguerons un flux de sortie : le **résultat** produit, et un flux d'entrée : l'**argument** (aussi appelé **paramètre**).
Une fonction peut être représentée comme ceci :



3 Syntaxe Python

Comme n'importe quel langage de programmation, Python permet d'organiser un programme à l'aide de fonctions, mais il présente, comme n'importe quel langage de programmation, une syntaxe à connaître :

```
def ma_fonction():
    """
    Ceci est la docstring de votre fonction. Elle sert a donner des indications
     sur la fonction, son ou ses arguments et son ou ses résultats.
    Quand vous écrirez une fonction, pensez à toujours mettre la docstring !
    """

    bloc instruction # corps de la fonction

ma_fonction()
```

3.1 Définition d'une fonction

La **définition de la fonction** permet de lui **donner un nom**, de **préciser les arguments attendus** en entrée, **réaliser le traitement de l'information** et **préciser le résultat renvoyé** par la fonction.

1. Quelle ligne permet de définir le nom de la fonction? .....
2. Comment peut-on savoir quelle ligne permet de définir le nom de la fonction? .....

   .....

3.2 Appel d'une fonction

Une fois définie, **une fonction n'est exécutée que si l'on fait appel à elle**.

1. Recopier le programme suivant :

```
def ma_fonction():
    """
    La fonction affiche 'bonjour'
    """

    print("bonjour")

ma_fonction()
```

2. Lancer le programme précédent avec et sans la ligne 9
   - Quelle est la manière d'appeler une fonction? .....
   - Par rapport à la définition de la fonction, où est située l'appel de la fonction? .....

4 Votre première fonction Python

Passons à l'écriture de votre première fonction.

1. Créer un fichier **premiere_fonction.py**

2. Recopier le code suivant :

```
1  def ma_premiere_fonction():
2      """
3      Ceci est votre première fonction.
4      Pas de paramètre
5      Pas de résultat
6      """
7
8      x = 10
9      a = x ** 2
10     print(a)
11
12 ma_premiere_fonction()
```

— Que fait cette fonction ? .....

.....

— Ajouter l'instruction `print(x)` sous la ligne 12 (sous `ma_premiere_fonction()` et exécuter à nouveau le programme. Que constatez-vous ? .....

.....

4.1 Fonction avec argument(s)

Le fait que la variable `x` n'existe que dans la fonction peut-être intéressant puisque la mémoire nécessaire au fonctionnement du programme sera allégée. Néanmoins, il est parfois utile qu'une fonction traite des informations nécessaires ailleurs dans le programme, auquel cas, il faut que `x` existe en dehors de la fonction et qu'il y entre en cas de besoin. Pour éviter les confusions, nous l'appellerons `x_2` :

```
1  def ma_premiere_fonction(x):
2      """
3      Ceci est votre première fonction.
4      x : int (ou float)
5      Pas de résultat
6      """
7      x = 10
8      a = x ** 2
9      print(a)
10
11 x_2 = 8
12 ma_premiere_fonction(x_2)
13 print(x_2)
```

La saisie de `x_2` se fait à l'extérieur de la fonction et donc, l'information existe toujours après l'exécution de la fonction.

— Comment signifions-nous que la fonction doit attendre un argument lors de sa définition ?

.....

— Où est défini l'argument pris la fonction (`x_2`) par rapport à l'appel de celle-ci ? .....

.....

— Réaliser un appel de cette fonction sans argument en modifiant la ligne : `ma_premiere_fonction(x_2)` par `ma_premiere_fonction()`. Quel est le message renvoyé

par la console ? .....

.....

— En vous renseignant sur l'instruction `return` et indiquer si le résultat du calcul est-il disponible après l'exécution de la fonction ? Pourquoi ? .....

.....

4.2 Fonction avec retour

La fonction `print()` n'est qu'un affichage dans la console et `a` n'est plus disponible à la fin de l'exécution de la fonction. Lorsque le résultat d'une fonction est nécessaire au fonctionnement du programme, il faut que l'information sorte de l'intérieur de la fonction.

Nous repartons du programme précédent et modifions le (regardez la dernière ligne) :

```
1  def ma_premiere_fonction(x):
2      """
3      Ceci est votre première fonction.
4      x : int (ou float)
5      Pas de résultat
6      """
7      x = 10
8      a = x ** 2
9      print(a)
10
11 x_2 = int(input("Entrer un nombre : "))
12 ma_premiere_fonction(x_2)
13 print(type(ma_premiere_fonction(x_2)))
```

La fonction `type` utilisée à la ligne 13 permet d'afficher la nature du flux de sortie de la fonction.

— Que renvoie la console ? Qu'est-ce que cela signifie ? .....

.....

— Modifier la ligne `print(a)` par la ligne `return a`. Quel(s) changement(s) pouvez-vous constater ? .....

— Modifier en conséquence la `docstring` de votre première fonction

5 C'est à vous !

Pour toutes les fonctions que vous allez écrire à partir de maintenant, toujours mettre la docstring associée !

Créer un fichier fonctions.py, c'est dans ce fichier que vous écrirez toutes les fonctions suivantes

EXERCICE 1

Écrire une fonction `bonjour` qui prend en argument une chaîne de caractères et qui **affiche** une message de bonjour.

Exemple de rendu :

```
1 bonjour("Bob")
2
3 #Résultat du programme :
4 Bonjour Bob !
```

Attention ce passage est important!

Taper dans la console (partie du bas) la ligne suivante : `bonjour("Bob")`, marquer en commentaire ce que vous obtenez.

Vous voyez qu'il est possible d'appeler nos fonctions depuis la console de l'IDE. De plus, vous pouvez constater que l'on peut directement mettre des valeurs sans passer par des variables.

L'utilisation des fonctions est à privilégier lorsque l'on programme. Dès à présent, vos programmes devront donc être structurés de la manière suivante :

```
def ...(...) :
    """
    la docstring de votre fonction
    """
    le code de votre fonction

exemple d appel de votre fonction
```

EXERCICE 2
Écrire une fonction `somme` qui prend en argument deux nombres et qui **renvoie** la somme de ces deux nombres.

EXERCICE 3
Écrire une fonction `mult` qui prend en argument deux nombres et qui **renvoie** le produit de ces deux nombres.

EXERCICE 4
En utilisant la fonction `mult`, écrire une fonction `carre` qui prend en argument un nombre et qui **renvoie** le carré de ce nombre.

EXERCICE 5
Écrire une fonction `puissance_n` qui prend en argument 2 nombres : `x` et `n`. La fonction devra **renvoyer** le `x` à la puissance `n`.

EXERCICE 6
Écrire une fonction `max2` qui prend en argument deux nombres et qui **renvoie** le plus grand des deux.

EXERCICE 7
Écrire une fonction `max3` qui prend en argument trois valeurs et qui **renvoie** la plus grande des trois. La fonction `max3` **devra utiliser** `max2`.

EXERCICE 8
Écrire une fonction `calcul_perimetre` qui prend en argument deux valeurs : la longueur et la largeur d'un rectangle. La fonction devra **renvoyer** le périmètre de ce rectangle.

EXERCICE 9
Écrire une fonction `calcul_aire` qui prend en argument deux valeurs : la longueur et la largeur d'un rectangle. La fonction devra **renvoyer** l'aire de ce rectangle.

EXERCICE 10

Écrire une fonction `triangle_rectangle` qui prend en argument 3 valeurs et renvoie `True` si le triangle est rectangle, `False` sinon.

RÉSUMÉ :

- En Python, il est possible de **structurer en programmer en un sous-programme** identifié par un nom. Ces sous-programmes sont des **fonctions**
- Une fonction peut posséder **un flux d'entrée : les arguments (ou paramètres)** et peut posséder **un flux de sortie : le résultat**
- La ligne `def ...(...):` permet de définir une fonction Python
- **La docstring** d'une fonction permet **d'expliciter son utilité, le type de son ou ses paramètres et le type de son ou ses résultats**
- Pour appeler une fonction dans son code, il suffit de mettre : `nom_de_la_fonction(paramètre(s)_de_la_fonction)`
- Le mot-clé `return` sert à **renvoier le résultat** pour qu'il puisse être **utilisable dans la suite du programme**