4 – Les fonctions

Une fonction est un ensemble d'instructions qui peut recevoir un ou plusieurs arguments (valeurs ou variables) et qui peut renvoyer une ou plusieurs valeurs de retour.

On définit une fonction nommée ***fonction1***, qui prend en entrée des ***arguments***, execute toutes les instructions indentées puis renvoie ***valeurs_renvoyées*** comme suit :

```
def fonction1(arguments):
        bloc_instructions
        return valeurs_renvoyées
```

Attention, le langage Python est très sensible à l'indentation, càd à l'alignement en début de ligne.

Exemple :

```
def conversion(duree_en_minutes):
        heures = duree_en_minutes // 60
        minutes = duree_en_minutes % 60
        return heures, minutes
```

Cette fonction, nommée *conversion*, prend comme argument *duree_en_minutes* et renvoie 2 valeurs qui sont contenu dans les variables nommées *heures* et *minutes*.

Pour appeler une fonction dans une console d'exécution (avec pycharm), il faut (pour notre exemple) selectionner les 4 lignes de la fontion, faire un clic droit, puis selectionner :

- Execute selection in Python console (Alt + Maj + e)
- Run file in Python console

Peu importe le choix, en bas de l'IDE (Integrated Development Environment), vous aurez une console qui s'affiche.
Il faudra ensuite indiquer son nom, ainsi que ses arguments entre parenthèses.

A partir de là, nous avons 2 possibilitées :

- Ecrire directement le nom de notre fonction et son argument :
  conversion(90)
  Qui nous renverra :
  (1, 30)
- Récupérer le résultat dans 2 variables afin de pouvoir le réutiliser plus tard :
  date_h, date_m = conversion(90)
  Qui nous renverra :
  ... rien ! Enfin pas tout à fait ! Visuellement rien n'apparait mais il s'est quand même passé quelque chose. On va appeler notre fonction conversion() en lui passant 90 en arguments, et mettre directement les valeurs retournées dans les variables date_h et date_m.
  On peut visualiser ce que contiennent les variables *date_h* et *date_m* en demandant

leur affichage :
date_h affiche 1
date_m affiche 30.

On pourrait aller un peu plus loin en demandant un affichage un peu plus user friendly en écrivant par exemple :

```
print("{} minutes = {}h et {}minutes".format(90, date_h, date_m))
```

La commande print(...) (qui est une des fonctions de base de Python), va demander l'affichage d'une chaîne de caractères.
Ensuite "{} minutes = {}h et {}minutes" sera la chaîne de caractères que nous verrons à l'écran.
Pour finir, .format(...) est une classe du module string, présent de base dans Python, qui va nous permettre, comme son nom l'indique, de formater notre chaine de caractères.
Ainsi, format(90, date_h, date_m) va nous permettre de remplacer les "{}" dans notre chaine de caractères, respectivement, les premières {} sera 90, puis les deuxièmes {}, la valeur contenu dans la variable date_h, et enfin, les dernières {}, la valeur contenu dans la variable date_m.
Ce formatage nous affichera donc : 90 minutes = 1h et 30minutes.

C'est une façon parmi tant d'autre de formatter l'affichage, on aurait pu tout aussi bien écrire :

```
print(str(90) + " minutes = " + str(date_h) + "h et " + str(date_m) + "minutes")
```

Ici, nous avons utilisé la concaténation (l'utilisation de signe "+"), afin de formatter notre chaine de caractères.
2 légers problèmes se posent, il nous faut utiliser la concaténation, ce qui pourrait augmenter le risque de faire des erreurs, mais aussi, nous somme obligé de *caster* nos variables, afin de pouvoir les utiliser dans une chaines de caractères.

L'utilisation de .format() est donc plus agréable et plus intuitive (puis elle est là, autant s'en servir !).

Pour appeler cette fonction dans un programme, il suffit d'écrire le nom de notre fonction en bas de notre fichier, en lui passant l'argument voulus, ce qui nous donnerait :

```
def conversion(duree_en_minutes):
    heures = duree_en_minutes // 60
    minutes = duree_en_minutes % 60
    return heures, minutes

conversion(90)
```

Faire bien attention à l'indentation, écrit comme suit, conversion(90) ne fait pas partie de la fonction car il est 2 lignes plus bas et indenté en tout début de ligne.
Lançons le programme en faisant un clic droit sur l'onglet avec le nom "main.py".
Rien ne se passe... Et pourtant !

La fonction à bien fonctionnée, mais pour des humains comme nous, il manque quelque chose nous permettant de visualiser les résultats.

Comme vue précédemment, pour afficher quelque chose dans la console, et donc à l'écran, nous devons le spécifier à Python, grâce à la fonction print()
Ainsi, print(conversion(90)) affichera (1, 30)
Comme dans l'exemple précédent, il est possible de rentrer le résultat directement dans des variables, en utilisant la forme date_h, date_m = conversion(90)
Puis afficher le résultat contenu dans les variables date_h et date_m, toujours en utilisant la fonction print() -> print(date_h) affichera 1 et print(date_m) affichera 30.

L'interet d'utiliser la fonction print() et de voir ce qu'il se passe dans notre code, entre autre en permettant de visualiser ce qui se trouve dans tel ou tel variable, pour vérifier qu'elle contient la valeur souhaité. Ce sera surtout très utile pour faire du débogage.

Une des autres fonctions de base, utile notamment quand nous travaillerons sur les chaines de caractères, et la fonction len(), elle renvoi la taille d'une chaînes de caractères.
Exemple :

len("test") renvoie 4

Il est aussi possible de l'utiliser avec une variable
Exemple :

maChaine = "test"
len(maChaine) renvoie 4

Bien évidamment, il est possible de mettre ce résultat dans une variable, afin de l'utiliser plus tard.
Exemple :

maChaine = "test"
longeurMaChaine = len(maChaine)
print(longueurMaChaine) -> affichera 4

Idéalement, une fonction devrait se limiter au calcul et au renvoi d'un résultat. Respecter cette règle vous aidera à la compréhension, la relecture, et le débugage ce celle-ci.

Prenons quelques bonnes habitudes :

Lors de la définition d'une fonction, il est de bonne pratique de :

- Donner un nom explicite et compréhensible a sa fonction
- Prendre l'habitude d'annoter les types (paramètres et retour)
- Documenter sa fonction (sera vu plus en profondeur en T°)
- Ecrire des tests (sera vu plus en profondeur en T°)
- Commenter sa fonction

Si l'on reprend notre fonction de conversion d'une durée en minutes, en heures / minutes :

```
def conversion(duree_en_minutes: int) -> int:
    """
        Converti une durée en minutes, en heures / minutes
    """
    # Quotient de la division euclidienne de notre paramètre
```

```
    heures = duree_en_minutes // 60 # On met le quotient dans la variable 'heures'

    # Reste de la division euclidienne de notre paramètre
    minutes = duree_en_minutes % 60 # On met le reste dans la variable 'minutes'

    # Valeurs de retour de notre fonction
    return heures, minutes # On renvoie la valeur contenu dans la variable heures, et celle de
minutes

assert conversion(90) == (1 , 30) # On teste si la conversion(90) renvoie bien (1, 30)
assert conversion(120) == (2 , 0) # On teste si la conversion(120) renvoie bien (2, 0)
```