Les fonctions sont les éléments structurants de base de tout langage procédural.

Elles offrent différents avantages :

- Évite la répétition : on peut « factoriser » une portion de code qui se répète lors de l'exécution en séquence d'un script ;
- Met en relief les données et les résultats : entrées et sorties de la fonction ;
- Permet la réutilisation : mécanisme de l'import ;
- Décompose une tâche complexe en tâches plus simples : conception de l'application.

Définition

Une fonction est un ensemble d'instructions regroupées sous un nom et s'exécutant à la demande.
On doit définir une fonction à chaque fois qu'un bloc d'instructions se trouve à plusieurs reprises dans le code ; il s'agit d'une « mise en facteur commun ».

Syntaxe

```
def nom_de_la_fonction (parametres):
  #ce que doit faire la fonction
```

- def pour expliquer que l'on définit une fonction.

- nom_de_la_fonction : vous pouvez mettre des lettres et chiffres (mais pas en premier) dans le nom mais pas d'espace.
- parametres : on peut préciser les paramètres de votre fonction comme en math pour f(x), le paramètre est x (En mathématique, on l'appelle une variable). S'il y a plusieurs paramètres, on les sépare par une virgule. S'il n'y en a pas, on met juste des parenthèses sans rien dedans.
- : On n'oublie pas les 2 points à la fin de la ligne ni l'indentation dans tout ce qui est à exécuter dans la fonction.

Exemples

```
def addition(a,b):
        print(a+b)
```

Cette fonction prend en entrée a et b et affiche le résultat de a+b. mais si on ne tape que ça, il ne se passe rien. Pour utiliser notre fonction, il faut l'appeler dans le programme principal :

```
def addition(a,b):
    print(a+b)
addition(2,3) #affiche 5
addition("ab","cde") #affiche abcde
addition([1, 2, 3] , [7, 8] ) #affiche [1, 2, 3, 7, 8]
```

Remarque : Nous n'avons pas précisé que a et b étaient des entiers donc python fera une addition de tout ce qu'il sait additionner comme ici les chaines de caractères ou les listes (que nous verrons plus loin).

```
def dire_chut() :
    print("Chhhhhhhhhuuuuuuuut!")

dire_chut() # affiche Chhhhhhhhhuuuuuuuut!
```

Voici une fonction qu'il suffit d'appeler en écrivant dire_chut() car il n'y a pas de paramètre.

On peut combiner les fonctions entre elles :

```
def addition(a,b):
    print(a+b)

def dire_chut() :
    print("Chhhhhhhhhuuuuuuuut!")

def ma_fonction(a) :
    addition(a,3)
    dire_chut()

ma_fonction(4)#affiche 7 et Chhhhhhhhhuuuuuuuut!
```

La commande return

La commande return permet de renvoyer un résultat obtenu par la fonction pour pouvoir l'utiliser dans la suite du programme. Elle est fondamentalement différente de print qui ne fait que l'afficher à l'écran (et donc plus utilisable dans notre programme).

Reprenons notre fonction addition précédente ci-dessus et essayons de la mettre dans une variable puis d'afficher cette variable.

```
def addition(a,b):
    print(a+b)

resultat=addition(2,3)
print(resultat)
```

Quand on met addition(2,3) dans la variable resultat, il affiche 5 et quand on veut afficher resultat il affiche None c'est à dire vide.

Que se passe-t-il ? En fait c'est très simple, quand on lance addition(2,3), il exécute la fonction dans laquelle on demande d'afficher a+b. Mais l'affichage se fait dans la console donc c'est comme si Python nous avait envoyé le résultat à nous et donc ne peut pas l'utiliser pour le mettre dans la variable resultat.

Si on veut qu'un programme renvoie un résultat qu'il puisse réutiliser ensuite, il faut utiliser la commande return au lieu de print. Par exemple :

```
def addition(a,b):
  return a+b

resultat=addition(2,3)
print(resultat)
```

Cette fois-ci, il n'affiche rien lorsqu'on n'exécute que la commande resultat=addition(2,3) et quand on affiche la variable resultat, elle contient bien 5.

Il faut donc faire très attention à ce que l'on veut faire : retourner un résultat pour le réutiliser (dans ce cas on utilise return) ou bien juste afficher pour nous le résultat (dans ce cas on utilise print). Il va de soi que si au final, on veut afficher à l'écran le résultat donné par une fonction, il faudra utiliser print comme on a fait ci-dessus même si dans la fonction on utilise return.

Remarque très importante : return arrête la fonction

Il est très important de savoir que quand on lance un return, la fonction s'arrête. Autrement dit, il faut être sûr d'avoir fait tout ce que l'on souhaitait faire avec notre fonction avant de lancer return pour renvoyer le résultat. Cela peut être cependant très pratique par exemple dans une boucle for, dès qu'on a le résultat voulu, on le renvoie sans avoir besoin de finir la boucle.

Par exemple, si je souhaite créer une fonction qui, à un nombre x renvoie son double et son carré et qu'on écrit :

```
def  f(x):
    double = 2*x
    return double
    carré = x*x
    return carré

print(f(5))
```

On voit que la fonction ne va afficher que le double de 5 et non son carré car la fonction s'arrête après avoir exécuté le premier return.

Pour contourner ce problème, il faut renvoyer les deux variables ensemble en les séparant par une virgule par exemple comme ceci :

```
def  f(x):
    double = 2*x
    carré = x*x
    return double,carré
print(f(5))
```