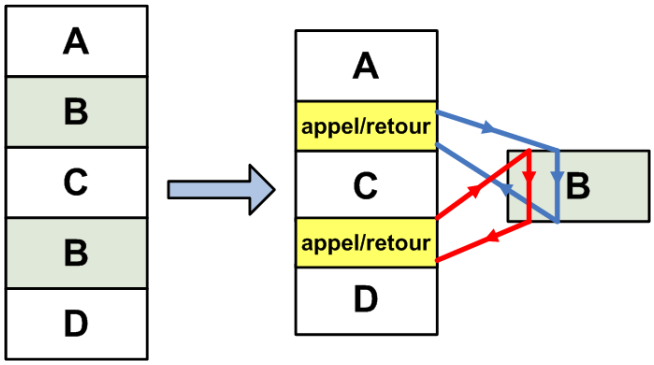
2.2 Les fonctions

Une fonction est un bloc d'instructions que l'on peut appeler à tout endroit d'un programme. Elle est particulièrement utile quand une tâche doit être réalisée plusieurs fois par un programme avec seulement des paramètres différents.

Nous avons déjà vu diverses fonctions prédéfinies : **print()**, **input()**, **range()**, **len()**…

Mais, on peut également créer ses propres fonctions afin d'éviter les répétitions de code et permettre une réutilisation :



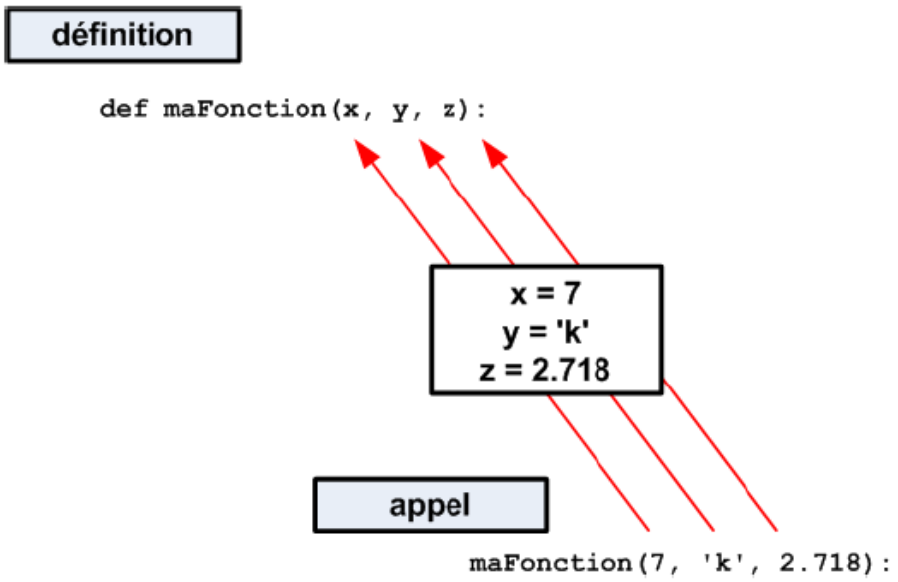
La définition d'une fonction se fait à l'aide du mot clé **def** :

```
def ma_fonction():
    # bloc d'instructions
```

Il est possible de définir des paramètres à la fonction. Ce sont les arguments de la fonction :

def maFonction(x,y,z)

Lors de l'appel de la fonction dans le programme principal, chaque paramètre de l'appel correspond dans l'ordre à chaque argument de la définition de la fonction. La correspondance se fait par affectation :



Exemple :

Dans le programme suivant, La fonction **table** permet d'afficher la table de multiplication d'un nombre. L'argument de la fonction **table** étant ce nombre :

Fonction-table.py - D:/Travail/ArdPyLab/Docs/Python/Fonction-table.py (3.7.7)

File Edit Format Run Options Window Help

```
def table(nb):
    i = 0
    while i < 10:
        print(i + 1, "*", nb, "=", (i + 1) * nb)
        i += 1

nombresaisi = False

while nombresaisi == False:

    nombre = input("Saisissez un nombre : ")

    try:
        nombresaisi = True
        nombre = int(nombre)
        table(nombre)
        break

    except:
        print("vous n'avez pas saisi un nombre!")

    finally:
        nombresaisi = False
```

Résultats dans la fenêtre Python Shell :

```
========== RESTART: D:/Travail/ArdPyLab/Docs/Python/Fonction-table.py ==========
Saisissez un nombre : 12
1 * 12 = 12
2 * 12 = 24
3 * 12 = 36
4 * 12 = 48
5 * 12 = 60
6 * 12 = 72
7 * 12 = 84
8 * 12 = 96
9 * 12 = 108
10 * 12 = 120
```

Les paramètres de la fonction peuvent être nommés et recevoir des valeurs par défaut. Ils peuvent ainsi être donnés dans le désordre et/ou pas en totalité.

Exemple :

Ajoutons à la fonction **table**, l'argument **max**=10 correspondant à la valeur maximale du multiplicateur et donnons une valeur par défaut au nombre à multiplier :

```
def table(nb=1, max=10):
    i = 0
    while i < max:
        print(i + 1, "*", nb, "=", (i + 1) * nb)
        i += 1
```

L'appel de la fonction pourra se faire ainsi :

. **table(nombre)** : la valeur par défaut de max est utilisée

. **table(nombre1, nombre2)** avec **nombre1** le nombre à multiplier et **nombre2** la valeur maximale du multiplicateur

. **table(nombre1, max=nombre2)**

. **table(nb=nombre1, max=nombre2)**

. **table(max=nombre2, nb=nombre1)**

. table() : les valeurs par défaut de nb et max sont utilisées

. <u>Fonctions avec return</u>

Les fonctions peuvent retourner une ou plusieurs données à l'aide du mot clé **return.** A noter qu'une fonction sans **return**, est plutôt appelée procédure.

<u>Exemple</u> :

De façon à pouvoir utiliser la nouvelle fonction **table**, nous allons modifier le programme d'affichage des tables de multiplication en créant une fonction permettant de saisir un nombre et qui retourne ce nombre :

```
def saisie(message):
    nombresaisi = False
    while nombresaisi == False:
        nombre = input(message)
        try:
            nombre = int(nombre)
            nombresaisi = True
        except:
            print("vous n'avez pas saisi un nombre!")
    return nombre
```

Le seul argument de la fonction est le message à afficher lors de la demande de saisie du nombre à l'aide de la fonction **input()**.

Si l'entrée clavier ne correspond pas à un nombre, l'utilisateur en est informé et la demande de saisi d'un nombre est affichée de nouveau.

Si l'entrée clavier est valide, le nombre saisi est retourné par la fonction.

Le programme d'affichage de la table de multiplication souhaitée est alors :

Fonction-table2.py - D:/Travail/ArdPyLab/Docs/Python/Fonction-table2.py (3.7.7)

File Edit Format Run Options Window Help

```
#Fonction affichage table de multiplication

def table(nb=1, max=10):
    i = 0
    while i < max:
        print(i + 1, "*", nb, "=", (i + 1) * nb)
        i += 1

# Fonction saisie d'un nombre

def saisie(message):
    nombresaisi = False
    while nombresaisi == False:
        nombre = input(message)
        try:
            nombre = int(nombre)
            nombresaisi = True
        except:
            print("vous n'avez pas saisi un nombre!")
    return nombre

# Appels des fonctions

nombre=saisie("Veuillez saisir le nombre à multiplier:")
multimax=saisie("Veuillez saisir la valeur maximale du multiplicateur:")

table(nombre,multimax)
```

Résultats dans la fenêtre Python Shell :

```
========== RESTART: D:/Travail/ArdPyLab/Docs/Python/Fonction-table2.py =========
Veuillez saisir le nombre à multiplier:az
vous n'avez pas saisi un nombre!
Veuillez saisir le nombre à multiplier:5
Veuillez saisir la valeur maximale du multiplicateur:d
vous n'avez pas saisi un nombre!
Veuillez saisir la valeur maximale du multiplicateur:9
1 * 5 = 5
2 * 5 = 10
3 * 5 = 15
4 * 5 = 20
5 * 5 = 25
6 * 5 = 30
7 * 5 = 35
8 * 5 = 40
9 * 5 = 45
```

Remarque :

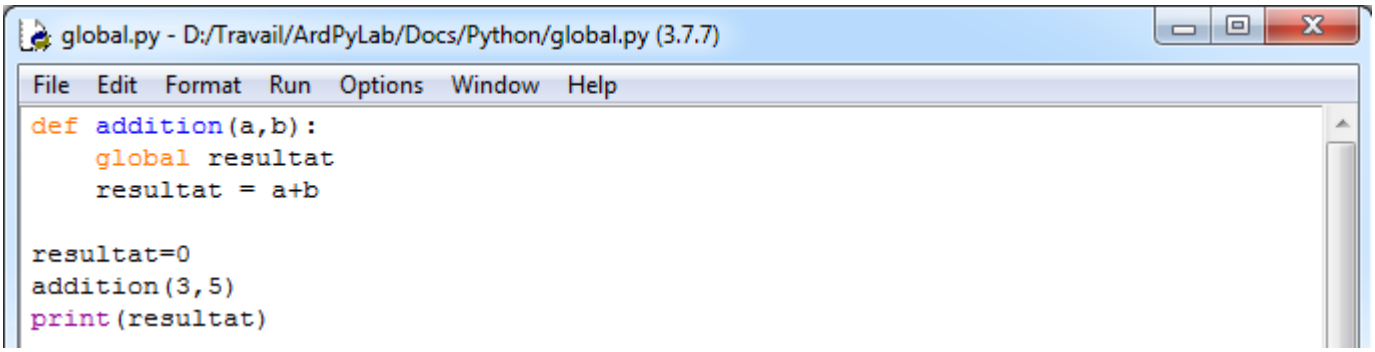
Les variables à l'intérieur du corps d'une fonction ne sont accessibles qu'à la fonction elle-même. On dit que ces variables sont des **variables locales**.

Une variable locale peut avoir le même nom qu'une variable du programme principal mais elle reste néanmoins indépendante.

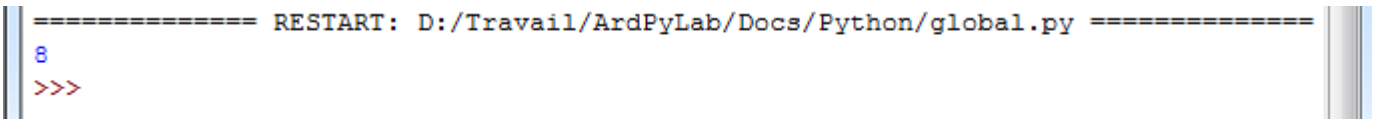
Le contenu des variables locales est inaccessible depuis l'extérieur de la fonction.

Les variables définies à l'extérieur d'une fonction sont des **variables globales**. Leur contenu est « visible » de l'intérieur d'une fonction, mais la fonction ne peut pas le modifier.

Pour modifier une variable globale au sein d'une fonction, il faut utiliser l'instruction **global**. Cette instruction permet d'indiquer quelles sont les variables à traiter globalement :



Résultats dans la fenêtre Python Shell :



. Fonctions lambda

Pour des fonctions très courtes, on peut utiliser des fonctions anonymes, connues aussi sous le nom de fonctions lambda.

Prenons l'exemple d'une fonction retournant la valeur de l'addition de deux nombres :

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

On peut écrire cette fonction en une seule ligne, avec le mot clé **lambda** :

```
Addition = lambda x,y: x+y
```

A noter cependant qu'avec les fonctions lambda :

. On ne peut les écrire que sur une seule ligne.

. On ne peut pas avoir plus d'une instruction dans la fonction.

Voici le programme permettant d'afficher le résultat de l'addition :

Lambda.py - D:/Travail/ArdPyLab/Docs/Python/Lambda.py (3.7.7)

File Edit Format Run Options Window Help

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

Addition = lambda x,y: x+y

def saisie(message):
    nombresaisi = False
    while nombresaisi == False:
        nombre = input(message)
        try:
            nombre = int(nombre)
            nombresaisi = True
        except:
            print("vous n'avez pas saisi un nombre!")
    return nombre

Nb1 = saisie("Saisissez un premier nombre : ")
Nb2 = saisie("Saisissez un deuxième nombre : ")

print ("la somme des deux nombres est :",addition(Nb1,Nb2))
print ("Fonction lambda - la somme des deux nombres est :",Addition(Nb1,Nb2))
```

Résultats dans la fenêtre Python Shell :

```
============== RESTART: D:/Travail/ArdPyLab/Docs/Python/Lambda.py ==============
Saisissez un premier nombre : 45
Saisissez un deuxième nombre : 5
la somme des deux nombres est : 50
Fonction lambda - la somme des deux nombres est : 50
>>>
```