LES FONCTIONS (PYTHON)

I- Principe et généralités

En programmation, les fonctions sont très utiles pour réaliser plusieurs fois la même opération au sein d'un programme.
Elles rendent également le code plus lisible et plus clair en le fractionnant en blocs logiques.
Vous connaissez déjà certaines fonctions Python. Par exemple math.cos(angle) du module math renvoie le cosinus de la variable angle exprimé en radian. Vous connaissez aussi des fonctions internes à Python comme range() ou len(). Pour l'instant, une fonction est à vos yeux une sorte de « boîte noire » (voir figure 1) :

- À laquelle vous passez aucune, une ou plusieurs variable(s) entre parenthèses. Ces variables sont appelées arguments. Il peut s'agir de n'importe quel type d'objet Python.
- Qui effectue une action.
- Et qui renvoie un objet Python ou rien du tout.

Par exemple, si vous appelez la fonction len() de la manière suivante :

```
1 >>> len ([0 , 1 , 2])
2 3
```

voici ce qui se passe :

1- vous appelez len() en lui passant une liste en argument (ici la liste [0, 1, 2]) ;

2- la fonction calcule la longueur de cette liste ;

3- elle vous renvoie un entier égal à cette longueur.

Autre exemple, si vous appelez la méthode ma_liste.append() (n'oubliez pas, une méthode est une fonction qui agit sur l'objet auquel elle est attachée par un point) :

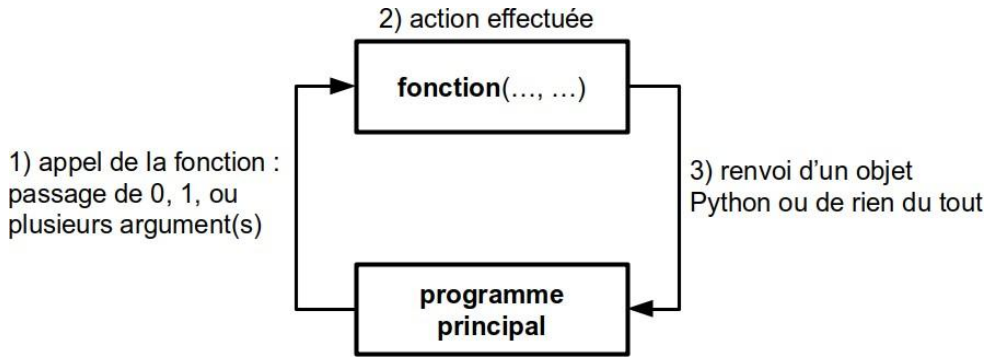

FIGURE 1 – Fonctionnement schématique d'une fonction

```
1 >>> ma_ liste . append (5)
```

1-Vous passez l'entier 5 en argument ;

2-la méthode append() ajoute l'entier 5 à l'objet ma_liste;

3-et elle ne renvoie rien.

II- Définition

Pour définir une fonction, Python utilise le mot-clé def. Si on souhaite que la fonction renvoie quelque chose, il faut utiliser le mot-clé return. Par exemple :

```
>>> def carre(x):
...     return x**2
...
>>> print(carre(2))
4
```

Notez que la syntaxe de def utilise les deux-points comme les boucles for et while ainsi que les tests if, un bloc d'instructions est donc attendu. De même que pour les boucles et les tests, l'indentation de ce bloc d'instructions (qu'on appelle le corps de la fonction) est obligatoire.

Dans l'exemple précédent, nous avons passé un argument à la fonction carre() qui nous a renvoyé (ou retourné) une valeur que nous avons immédiatement affichée à l'écran avec l'instruction print(). Que veut dire valeur renvoyée ? Et bien cela signifie que cette dernière est récupérable dans une variable :

```
>>> res = carre (2)
>>> print(res)
4
```

Ici, le résultat renvoyé par la fonction est stocké dans la variable res. Notez qu'une fonction ne prend pas forcément un argument et ne renvoie pas forcément une valeur, par exemple :

```
>>> def hello ():
...     print("bonjour")
...
>>> hello ()
bonjour
```

Dans ce cas la fonction, hello() se contente d'afficher la chaîne de caractères "bonjour" à l'écran. Elle ne prend aucun argument et ne renvoie rien. Par conséquent, cela n'a pas de sens de vouloir récupérer dans une variable le résultat renvoyé par une telle fonction. Si on essaie tout de même, Python affecte la valeur None qui signifie rien en anglais :

```
>>> var = hello ()
bonjour
>>> print(var)
None
```

Ceci n'est pas une faute car Python n'émet pas d'erreur, toutefois cela ne présente, la plupart du temps, guère d'intérêt.

III- Passage d'arguments

Le nombre d'arguments que l'on peut passer à une fonction est variable. Nous avons vu ci-dessus des fonctions auxquelles on passait 0 ou 1 argument. Dans les chapitres précédents, vous

avez rencontré des fonctions internes à Python qui prenaient au moins 2 arguments. Souvenez-vous par exemple de range(1, 10) ou encore range(1, 10, 2). Le nombre d'argument est donc laissé libre à l'initiative du programmeur qui développe une nouvelle fonction.

Une particularité des fonctions en Python est que vous n'êtes pas obligé de préciser le type des arguments que vous lui passez, dès lors que les opérations que vous effectuez avec ces arguments sont valides. Python est en effet connu comme étant un langage au « typage dynamique », c'est-à-dire qu'il reconnaît pour vous le type des variables au moment de l'exécution. Par exemple :

```
>>> def fois(x, y):
...     return x*y
...
>>> fois(2 , 3)
6
>>> fois (3.1415 , 5.23)
16 .430045000000003
>>> fois(' to ', 2)
'toto '
>>> fois ([1 ,3], 2)
[1 , 3 , 1 , 3]
```

L'opérateur * reconnaît plusieurs types (entiers, floats, chaînes de caractères, listes). Notre fonction fois() est donc capable d'effectuer des tâches différentes ! Même si Python autorise cela, méfiez-vous tout de même de cette grande flexibilité qui pourrait conduire à des surprises dans vos futurs programmes. En général, il est plus judicieux que chaque argument ait un type précis (entiers, floats, chaînes de caractères, etc) et pas l'un ou l'autre.

IV- Renvoi de résultats

Un énorme avantage en Python est que les fonctions sont capables de renvoyer plusieurs objets à la fois, comme dans cette fraction de code :

```
>>> def carre_cube(x):
...     return x**2 , x**3
...
>>> carre_ cube (2)
(4 , 8)
```

En réalité Python ne renvoie qu'un seul objet, mais celui-ci peut être séquentiel, c'est-à-dire contenir lui même d'autres objets. Dans notre exemple Python renvoie un objet de type tuple, type que nous verrons dans le chapitre 13 Dictionnaires et tuples (grosso modo, il s'agit d'une sorte de liste avec des propriétés différentes). Notre fonction pourrait tout autant renvoyer une liste :

```
>>> def carre_cube2(x):
...     return [ x**2 , x **3]
...
>>> carre_ cube2 (3)
```

```
5 [9 , 27]
```

Renvoyer un tuple ou une liste de deux éléments (ou plus) est très pratique en conjonction avec l'affectation multiple, par exemple :

```
1 >>> z1 , z2 = carre_ cube2 (3)
2 >>> z1
3 9
4 >>> z2
5 27
```

Cela permet de récupérer plusieurs valeurs renvoyées par une fonction et de les affecter à la volée à des variables diffé- rentes.

V- Variables locales et variables globales

Lorsqu'on manipule des fonctions, il est essentiel de bien comprendre comment se comportent les variables. Une variable est dite **locale** lorsqu'elle est créée dans une fonction. Elle n'existera et ne sera visible que lors de l'exécution de ladite fonction.
Une variable est dite **globale** lorsqu'elle est créée dans le programme principal. Elle sera visible partout dans le pro gramme.

VI- ACTIVITES PRATIQUES :

Activité 1 : Définir une fonction sans paramètre

Tapez le code suivant et observez le résultat.

```
>>> def fonction():   # Definition de fonction sans parametre - Ne pas oublier les :
    n=10  # Faire une tabulation en debut de ligne
    while n>0:        # Faire une tab. en debut de ligne et ne pas oublier les:
    print(n/2, n%2)  # Faire 2 tabulations en debut de ligne
    n=n-1             # Faire 2 tabulations en debut de ligne
# Tapez encore une fois <Enter> si vous êtes en ligne de commande
>>>fonction() # Appel de la fonction
```

Exercice 11 : Définir une fonction sans paramètre qui appelle une autre fonction

Tapez le code suivant et observez le résultat.

```
>>> def fonction2():          # Ne pas oublier les : et une tab. sur la lignesuivante
print('Affichage du resultat et du reste de la division des entiers de 10\')
fonction()                    # Faire une tabulation en debut de ligne
                              # Tapez encore une fois <Enter> si vous etes en ligne de commande
>>> fonction2()
```

Exercice 12 : Définir une fonction à 1 paramètre

Tapez le code suivant et observez le résultat.

```
def fonction(n): # Définition d'une fonction a un paramètre
...         while n>0:              # Faire une tab. en début de ligne et ne pas oublier les:
...                 print(n/2, n%2)     # Faire 2 tabulations en debut de ligne
...                 n=n-1               # Faire 2 tabulations en début de ligne
# Tapez encore une fois <Enter> si vous êtes en ligne de commande
>>> fonction(3)     # Appel de la fonction avec le paramètre 3
# Tapez encore une fois <Enter> si vous êtes en ligne de commande
>>> fonction(5)     # Appel de la fonction avec le paramètre 5
```

Exercice 13 : Définir une fonction à 2 paramètres

Tapez le code suivant et observez le résultat.

```
>>> def fonction(entree,diviseur): # Definition d'une fct. a plusieurs parametres
...         while(entree>0):
...                 print(entree/diviseur, entree%diviseur)
...                 entree=entree-1
# Tapez encore une fois <Enter> si vous etes en ligne de commande
>>> fonction(10,2)
```

Exercice 14 : Tapez le code suivant et observez le résultat.

```
>>> def afficher3fois(arg):
...         print(arg, arg, arg)
                # Tapez encore une fois <Enter> si vous Ãªtes en ligne de commande
>>> afficher3fois(3)
>>> afficher3fois('exemple')
>>> afficher3fois([3,4])
>>> afficher3fois(3*4)
```

Exercice 17 : Tapez le code suivant et observez le résultat.

```
>>> def fonction(entree=10,diviseur=2):
...         while(entree>0):
...                 print(entree/diviseur, entree%diviseur)
...                 entree=entree-1
# Tapez encore une fois <Enter> si vous êtes en ligne de commande
>>> fonction()
>>> fonction(4,2)
>>> fonction(diviseur=4,entree=12)
```

Affecter une instance de fonction à une variable

```
>>>def multiplication(n,p):
...             return n*p # pour retourner une valeur
# Tapez encore une fois <Enter> si vous êtes en ligne de commande
>>> a=multiplication(3,6)
>>> a
```

Fonction avec un nombre variable de paramètres

```
def fonction_test(*args):
        print("type de args :",type(args))
        for arg in args: print("paramètre:", arg) print()

fonction_test()
fonction_test(1)
fonction_test(1,"a")
fonction_test(1,"a",3)
```

Variable locale/variable globale

Exemple 10 : Mauvaise fonction echange

```
def echange(a,b):
print("adresses des parametres : ", id(a),id(b))
c=a
a=b
b=c
x,y = 2,3
print("adresses de x et de y   : ", id(x),id(y))
echange(x,y)
print("x=",x)
print("y=",y)
```

Si vous exécutez le code précédent, vous verrez que les variables x et y n'ont pas été échangées car il s'agit de variables non immuables.

Exemple 11 : Exemple de fonction echange qui fonctionne

```
def echange2(a,b):
c=a
a=b
b=c
return a,b

x,y = 2,3
x,y=echange2(x,y) # les valeurs de x et y sont bien échangées car # on les modifie leur valeur ici
```

```
print("x=",x)
print("y=",y)
print() # pour sauter une ligne
a=2 b=3
a,b=echange2(a,b) # les valeurs de a et b sont bien échangées car # on les modifie leur valeur ici
print("a=",a)
print("b=",b) print()
```