Chapitre 3 : Les Fonctions

I. Principe et généralités

I.1) Les fonctions en Python

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.

Une fonction peut être :

- Prédéfinie par le langage (print, input, …) ;
- Crée par l'utilisateur afin d'exécuter un certain traitement.

I.2) Principes

Le principe général d'une fonction est le suivant :

- Une fonction peut recevoir de 0 à n variables entre parenthèses. Ces variables sont appelées arguments ;
- Elle effectue une action ;
- Elle *peut* renvoyer un résultat.

Chaque fonction effectue en général une tâche unique et précise. Si cela se complique, il est plus judicieux d'écrire plusieurs fonctions *(qui peuvent éventuellement s'appeler les unes les autres)*. Cette modularité améliore la qualité générale et la lisibilité du code.

II. Définition

La syntaxe utilisée pour la définition d'une fonction est :

```
def nom_fonction(liste_paramètres):

    #Instructions
```

Notons que la syntaxe de *def* utilise les deux-points, un bloc d'instructions est donc attendu. De même que pour les boucles et les conditions, l'indentation de ce bloc d'instructions *(appelé aussi corps de la fonction)* est obligatoire.

Si la fonction retourne un résultat, on utilise le mot clé *return*.

Exemple :

```
def courbe (x):
      #Calculer f(x) = 2x+3
      return ((2*x)+3)
y= courbe (5)
print ("La valeur de y est " +str (y))
```

On aura :

```
La valeur de y est 13
```

Dans l'exemple précédent, on passé un seul argument à la fonction courbe() qui nous a renvoyé une valeur qui sera immédiatement affichée à l'écran avec l'instruction print().

Le résultat renvoyé par une fonction doit être récupérable dans une variable.

Notons qu'il est possible de définir une fonction qui ne renvoie pas forcément une valeur.

Exemple :

```
def fahrenheit(celsius):
    #Conversion degré Celsius en degré Fahrenheit """
    print(celsius*9.0/5.0 + 32.0)
fahrenheit(0)
32.0
fahrenheit(24)
75.2
temperature = 13
```

```
fahrenheit(temperature)

55.4
```

Dans ce cas, la fonction fahrenheit() reçoit le paramètre *celsius*, exécute la conversion nécessaire et affiche le résultat. Par conséquent, inutile de récupérer dans une variable le résultat renvoyé par une telle fonction. Si on essaie tout de même, on aura le résultat suivant :

```
var = fahrenheit (5)

print (var)

41.0 # Affichage à partir de la fonction  fahrenheit()

None # contenu de la variable var
```

III. Passage d'arguments

Comme dans n'importe quel langage de programmation, le nombre d'arguments que l'on peut passer à une fonction est variable.

Python est en effet un langage au *« typage dynamique »*, c'est-à-dire qu'il reconnaît automatiquement le type des variables au moment de l'exécution. Ainsi, il n'est pas obligatoire de préciser le type des arguments passées si les opérations effectuées avec ces arguments sont valides.

Exemple 1 :

```
def produit (x, y):

    print (x*y) # La fonction revoit la valeur x * y


produit (6,4) # Passage de deux entiers

24

produit (3.15,2) # Passage d'un réel et d'un entier

6.3
```

```
produit ("bonjour! ",2) # Passage d'un texte et d'un entier

bonjour! bonjour!
```

L'opérateur « * » reconnaît plusieurs types (entiers, réels, chaînes de caractères). La fonction produit() est donc capable d'effectuer des tâches différentes.

Même si Python autorise cela, ce type d'opérations doit être effectué avec précaution puisque cette flexibilité peut engendrer, par la suite, des résultats inattendus ou inexplicables.

Exemple 2 :

Créer une fonction *distance()* qui calcule la distance euclidienne en trois dimensions entre deux points.

On rappelle que la distance euclidienne *d* entre deux points A et B de coordonnées cartésiennes respectives (xA,yA,zA) et (xB,yB,zB) se calcule comme suit :

$$d = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2 + (z_B - z_A)^2}$$

Solution :

```
def distance (xa,ya,za,xb,yb,zb):

    d= ((xa-xb)**2) + ((ya-yb)**2) + ((za-zb)**2)

    distance = d**0.5

    return distance

xa=ya=za=0 # initialisation multiple

xb=yb=zb=1

d3d = distance(xa,ya,za,xb,yb,zb)

print (d3d)
```

```
→ 1.7320508075688772
```

IV. Renvoi de résultats

Contrairement à d'autres langages, Python permet qu'une fonction revoie plus qu'un résultat.

Exemple :

La fonction produit(x,y) permet de renvoyer deux résultats :

- x*y : opérateur « * » ;
- x^y : opérateur « ** ».

Exemple :

```
def produit (x, y):

    return x*y, x**y

print (produit (2,4))
```

Résultat :

```
(8,16)
```

V. Valeurs par défaut pour les paramètres

Dans la définition d'une fonction, il est possible de définir un argument par défaut pour un ou plusieurs paramètres. On obtient ainsi une fonction qui peut être appelée avec une partie seulement des arguments attendus.

Exemple :

Une fonction qui calcule le prix TTC d'un article.

```
def ttc (prixht,tauxtva=0.15):

    return prixht*(1+tauxtva)

montant = ttc (10,0.1)

print (montant)

a_payer = ttc (20)

print (a_payer)
```

```
→ 11.0

→ 23.0
```

Dans la définition de la fonction, la valeur par défaut de « taux tva » est fixée à 15 %. ainsi, deux scénarios peuvent avoir lieu :

- Lors du premier appel : La valeur « tauxtva » est définie à 10 % ainsi, le calcul se fait sur la base de la valeur définie lors de l'appel et ignore ainsi la valeur par défaut (0.15) ;
- Lors du second appel, on se limite à définir la valeur du « prixht ». Ainsi, le calcul se fait en évaluant la valeur de TVA à celle définie par défaut.

Il est donc possible d'appeler la fonction ttc() soit en lui passant un ou deux paramètres.

VI. 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 la fonction où elle a été définie.

Une variable est dite *globale* lorsqu'elle est créée dans le programme principal. Elle sera visible partout dans le programme.

VII. Appel d'une fonction dans une autre fonction

L'appel d'une fonction en Python peut être :

- Soit à partir du programme principal ;
- Soit à partir d'une autre fonction.

Généralement, on peut appeler une fonction de n'importe où à partir du moment où elle est visible par Python (c'est-à-dire chargée dans la mémoire).

Exemple :

On utilisera une fonction calc() pour calculer les solutions de l'équation $f(x) = x^2+2x+3$ pour les valeurs allant de 4 à 10. Ainsi :

- La fonction calc() initialise une liste vide et la remplit progressivement avec les valeurs calculées ;
- La fonction calc() appelle, pendant chaque itération, la fonction ploynome() pour faire l'opération nécessaire, et récupère le résultat de l'opération afin de l'insérer dans la nouvelle case de la liste.

```
    # définition des fonctions
    def polynome (x):
        return (x**2+2*x+3)#x²+2x+3
    def calc (debut,fin):
        liste = []
        for x in range (debut,fin+1):
            liste.append(polynome(x))
        return liste
    # programme principal
    y=calc(4,10)
    print (y)
→ [27, 38, 51, 66, 83, 102, 123]
```

VIII. Les fonctions lambda

VIII.1) Présentation

Python propose un autre moyen de créer des fonctions extrêmement courtes car limitées à une seule instruction. Ces fonctions sont appelées « fonctions lambda ».

En fait, il n'y a aucune différence réelle entre une fonction classique et une fonction lambda. Elles sont utilisées par goût ou pour des raisons de style.

En effet, les lambdas sont très pratiques pour créer des fonctions *jetables* : quand on a besoin d'une fonction, mais que l'on ne va l'utiliser qu'une seule fois.

Une fonction lambda se caractérise par :

- Utilisation du mot clé lambda au lieu de def ;
- pas de parenthèses ;
- pas de mot clé return.

VIII.2) Syntaxe

VIII.2.A) Déclaration

La définition d'une fonction lambda se fait comme suit :

```
lambda argument 1, argument 2, …, argument n : instruction
```

D'abord, on a le mot-clé *lambda* suivi de la liste des arguments, séparés par des virgules. Ensuite figure un nouveau signe *les deux points* « : » et l'instruction de la fonction lambda. C'est le résultat de l'instruction qui sera placé ici qui sera renvoyé par la fonction.

VIII.2.B) Appel

L'appel d'une fonction lambda peut se faire en stockant la fonction lambda nouvellement définie dans une variable, par une simple affectation.

VIII.3) Exemple

```
f = lambda x: x*x

print (f(10))

print (f(-3))

→ 100

→ 9
```

IX. Fonctions récursives

Une fonction récursive est une fonction qui s'appelle elle-même. Les fonctions récursives permettent d'obtenir une certaine efficacité dans la résolution de certains algorithmes comme le tri rapide.

Exemple :

Parmi les exemples des fonctions récursives les plus connus, on cite celui de la fonction factorielle :

Version non récursive :

```
def factorielle(n):

    fact = 1
```

```
        for i in range (2,n+1):
            fact = fact*i
        return fact
    # programme principal
    nb = 5
    valeur =factorielle(nb)
    print ("La factorielle de " +str(nb)+" est "+str(valeur) )
```

Version récursive :

```
    def factorielle(n):
        if n == 1:
            return 1
        else :
            return n*factorielle(n-1)
    # prog principal
    print (factorielle(5))
```

Dans les deux cas, on obtient le même résultat : 120

En général, la récursivité est utilisée lorsqu'une expression dans la fonction nécessite un résultat qui peut être produit par un appel à la fonction elle-même.

Notons qu'il est important, dans ce genre de fonctions, de s'assurer qu'il y a une condition permettant de stopper les appels récursifs, sinon on peut entrer dans une récursion sans fin *(de la même façon qu'on peut avoir des boucles while sans fin).*

X. Règles LGI

Lorsque Python rencontre une variable, il va traiter la résolution de son nom avec des priorités particulières.

D'abord il va vérifier si la variable est locale, puis si elle n'existe pas localement, il vérifiera si elle est globale et enfin si elle n'est pas globale, il testera si elle est interne *(par exemple la fonction len() est considérée comme une fonction interne à Python, elle existe à chaque lancement du logiciel)*.

On appelle cette règle la règle *LGI* (Locale, Globale, Interne).

Exemple :

```
    from math import pi

    def unefonction ():

        pi = 10

        print ("Dans la fonction, la valeur de pi est : " +str
    (pi))

    # Programme principal

    print ("La valeur initiale de pi depuis la bibliothèque math
    est : " +str(pi))

    pi = 5

    print ("Nouvelle valeur de pi en tant que variable : " +str
    (pi))

    unefonction ()

    print ("La valeur de pi après l'appel de la fonction : " + str
    (pi))
```

```
→ La valeur initiale de pi depuis la bibliothèque math est :
3.141592653589793

→ Nouvelle valeur de pi en tant que variable : 5

→ Dans la fonction, la valeur de pi est : 10

→ La valeur de pi après l'appel de la fonction : 5
```