Chapitre 3 : Les fonctions

I) Définitions et exemples

Def : Une fonction est une succession d'opérations faites sur une ou plusieurs variables nommées « entrées de la fonction » et renvoie un résultat appelé « sortie de la fonction ».

La déclaration d'une fonction mathématique se fait dans l'ordre par :

- le mot-clé **def**,
- le nom de la fonction,
- entre parenthèses, le ou les paramètre(s) d'entrée (réels) de la fonction.

On trouve ensuite le corps de fonction (avec indentation) dans lequel des calculs intermédiaires sont réalisés (avec des boucles, des conditions, etc.), ainsi que l'aide de la fonction.

Enfin, la (ou les) valeur(s) de sortie de la fonction se fait à l'aide du mot-clé **return**.

def nom_de_la_fonction(entrées_de_la_fonction) :
    instructions
    return(sortie_de_la_fonction)

Exemple : Le code suivant permet de créer une fonction qui calcule l'aire d'un rectangle de largeur l et de longueur L. On affiche ensuite le résultat obtenu en prenant $l = 3$ et $L = 6$.

```
1  def aire(l,L):
2      A=l*L
3      return(A)
4  print(aire(3,6)) #affiche 18 donc
```

Rq : L'écriture d'une fonction se fait avec des variables (on ne spécifie pas leur valeur au moment de la création de la fonction). L'idée étant de pouvoir appliquer la même fonction plusieurs fois avec des valeurs différentes sans devoir modifier le code.

Exemple : Le code suivant permet de calculer $n! = 1 \times 2 \times 3 \times \ldots \times n$ où n est l'entrée de la fonction.

```
1  def factoriel(n):
2      p=1
3      for k in range(1,n+1):
4          p=p*k
5      return(p)
6  print(factoriel(5)) # affiche 120
```

II) Documentation et signature

Une fonction a vocation à être écrite en début de programme pour pouvoir être appelée à de multiples endroits. Lorsque les programmes seront longs, il sera pertinent d'ajouter des commentaires et explications sur la fonction crée pour s'y retrouver en cas de besoin.

La documentation d'une fonction est un commentaire spécifique, inséré dans le corps de la fonction, et entouré par 3 guillemets. Cette documentation est accessible dans la console, en tapant **help**(nom_de_la_fonction).

Rq : Notons qu'en tapant """ suivi de « entrée » spyder propose par défaut une documentation que l'on peut compléter.

Exemple :

```
1   def factoriel(n):
2       """
3       permet de calculer n!
4       n est l'entrée et c'est un entier positif
5
6       """
7       p=1
8       for k in range(1,n+1):
9           p=p*k
10      return(p)
11  help(factoriel)
```

```
In [31]: runfile('C:/Users/lucdu/.spyder-py3/temp.py', wdir='C:/Users/lucdu/.spyder-py3')
Help on function factoriel in module __main__:

factoriel(n)
    permet de calculer n!
    n est l'entrée et c'est un entier positif
```

Rq : La documentation peut comporter (idéalement) :

- le type et le rôle de chaque paramètre,
- le type de la valeur de retour,
- des détails sur ce que réalise la fonction,
- des exemples si possible.

Def : Pour chaque fonction écrite en Python, on appelle signature de cette fonction la donnée des éléments suivants :

- son nom
- les (noms et) types des entrées
- les (noms et) types des sorties

La syntaxe usuelle est de la forme : nom_de_la_fonction(type_des_entrées) → type_des_sorties

Exemple : Pour la fonction factoriel précédente, sa signature est : factoriel (int) → int

Rq : On peut préciser ces informations lors de l'écriture d'une fonction en ajoutant après chaque entrée « : » suivi du type de l'entrée ainsi que le symbole « -> » suivi du type des sorties une fois la parenthèse des entrées fermée. On peut ensuite y accéder en important le module « inspect » via l'instruction « inspect.signature(nom_de_la_fonction)

Exemple :

```
1   import inspect
2   def factoriel(n:int)->int:
3       """
4       permet de calculer n!
5       n est l'entrée et c'est un entier positif
6
7       """
8       p=1
9       for k in range(1,n+1):
10          p=p*k
11      return(p)
12
13  print(inspect.signature(factoriel))
```

```
In [43]: runfile('C:/Users/lucdu/.spyder-py3/temp.py', wdir='C:/Users/lucdu/.spyder-py3')
(n: int) -> int
```

III) Variables locales et globales

Def : toute variable définie dans l'écriture d'une fonction ne garde sa valeur que dans la fonction. On parle de variable **locale**. Elle n'est plus définie, et n'a donc plus de valeur en dehors de la fonction.
Au contraire une variable définie dans le corps principal du programme peut être utilisée à tout moment du programme ou des fonctions qu'il contient en conservant sa valeur.

Prop : - une **variable locale** est une variable déclarée et utilisée dans une fonction. Elle n'est accessible en lecture ou en écriture que dans cette fonction, d'où sa portée **locale**.

- une **variable globale** est accessible en lecture et en écriture à la fois par les fonctions et par le programme principal. Pour donner une portée globale à une variable, il faut la déclarer avec le mot-clé **global** dans la fonction ou dans le programme principal.

Logique « L-E-G » de Python :

Une fonction python tente d'abord de trouver une variable :
- localement (L),
- puis dans le programme englobant (E) la fonction,
- et enfin globalement (G) dans tout le reste si elle ne l'a pas trouvée avant.

Rq : En pratique Python commencera par chercher une variable dans le corps principal du programme, il la choisira qu'elle soit définie en tant que variable globale ou non. Les cas où il sera nécessaire de spécifier qu'une variable est globale seront rares

Exemple :

```
1   global m
2   m=5 # m est une variable globale
3   print(m) # affiche 5
4   def ajoute3(a):
5       m=3 #m est une variable locale
6       print(m) #affichera 3 en lançant la fonction
7       return(a+3)
8
9   print(ajoute3(10)) # affiche 13
10  m=m+1
11  print(m) # affiche 6
12
13  def test():
14      return(m)
15  print(test()) #affiche 6
```

Rq : On obtiendra exactement le même résultat si on supprime la première ligne.

Exemple : Donner la signature de la fonction suivante et expliquer ce qu'elle fait :

```
1   def tralala(n:int)->list:
2       L=[]
3       for k in range(1,n+1):
4           if n%k==0:
5               L=L+[k]
6       return(L)
```

Rq : Ici L et k sont des variables locales.

Exercices :

1) Ecrire une fonction maximum qui renvoie la plus grande des 3 valeurs entrée par l'utilisateur.

2) Ecrire une fonction qui renvoie « est rectangle » ou « n'est pas rectangle », à partir de la donnée des 3 longueurs d'un triangle

3) Ecrire une fonction qui prend en entrée les coordonnées de deux vecteurs du plan et qui retourne le résultat de leur produit scalaire.