Chapitre III

Fonctions

III.1 Introduction

Les problèmes complexes nous arrivent parfois à écrire des programmes de centaine ou millième instructions, ainsi nous obtenons des programmes peu compréhensibles et peu structurés. Pour éviter ce problème, on décompose le programme global en plusieurs sous-programmes (**fonctions**) qui peuvent interagir entre eux pour résoudre un problème donné.

III.2 C'est quoi une fonction

Donc une **fonction** est un sous-programme réalise une tache particulière, elle peut être étudiée/ testée séparément et réutilisée dans d'autres programmes.

III.3 Avantage de la subdivision d'un programme en plusieurs fonctions

- Meilleure lisibilité
- Diminution du risque d'erreurs
- Possibilité de tests sélectifs
- Réutilisation de fonctions déjà existantes
- Favorisation du travail en équipe

III.4 Définir une fonction en Python

La syntaxe pour définir une fonction en Python est la suivante

```
def nomDeLaFonction(paramètre1, paramètre2,… paramètren ):
bloc d'instructions      # c'est l'ensemble des instructions qui réalisent l'action de la fonction.
return # objet (valeur, caractère….)
```

Remarques importantes :

- Il est possible de définir une fonction sans paramètres en n'écrivant rien entre les parenthèses de son entête.
- Il est possible de définir une fonction qui ne renvoie rien.

Exemple III.1

Dans le programme suivant, la fonction '*somme*' possède deux arguments (paramètre) (x et y), elle applique l'opérateur '+' approprié et renvoi le résultat de cette opération.

```
1
2   def somme( x, y):
3       return x+y
```

Résultats d'exécution

```
In [46]: somme(3,2)
Out[46]: 5

In [47]: somme("Université de ", "M'sila")
Out[47]: "Université de M'sila"

In [48]: somme( ["Python"],["Inst", "ETLC",2023] )
Out[48]: ['Python', 'Inst', 'ETLC', 2023]
```

Cependant l'instruction suivante n'est pas valable car, il n'y a pas un opérateur + défini pour *int +str*

```
somme (2, "M'sila")
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

III.5 Variables locales d'une fonction

Toutes les variables déclarées au sein du corps de la fonction ne sont pas accessibles au d'autres fonctions, alors elles sont considérées comme des **variables locales**.

Exemple III.2

Dans le programme suivant la variable 'p' est une variable locale pour la fonction 'func', ainsi elle n'est pas accessible de l'extérieur de cette fonction.

```
1
2   def func(x):
3       p=x/2
4       print ("p=",p)
```

Résultats d'exécution

```
In [54]: func(10)
p= 5.0

In [55]: print(p)
Traceback (most recent call last):

  Cell In[55], line 1
    print(p)

NameError: name 'p' is not defined
```

III.6 Variables globales

Toute variable définie à l'extérieur d'une fonction est une variable globale, son contenu est visible de l'intérieur d'une fonction, mais la fonction ne peut pas le modifier.

Exemple III.3

```
1   x=100
2   def modif(x):
3       x=x/2
4       print ("x=",x)
5
6   print(x)
```

Dans le programme précédent la fonction 'modif' lit la variable *x* mais elle ne peut pas la modifier. Ainsi, l'instruction 6 (*print (x))* donne toujours la valeur 100 et pas 50, car la variable x est une variable locale. Cependant la variable x à l'intérieur de la fonction est une variable locale et contient la valeur 50.

III. 7 Argument par défaut d'une fonction

Dans les paragraphes précédents, nous avons constaté qu'il est nécessaire de préciser les arguments (paramètres) de la fonction lors de son appel. Cependant ; il est possible de définir une valeur par défaut pour un ou plusieurs argument/s, dans ce cas, lors de l'appel de la fonction, il suffit de préciser seulement les arguments qui n'ont pas des valeurs par défaut.

Exemple III.4

Dans le programme suivant, la fonction '*factoriel'* a un argument par défaut n=1, ainsi il est possible d'appeler la fonction sans préciser la valeur de n, Dans ce cas elle retourne la valeur 1

```
def factoriel(n=1):
    x=n
    while n>1:
        x=x*(n-1)
        n=n-1
    return x
```

Résultats d'exécution

```
In [8]: factoriel(5)
Out[8]: 120

In [9]: factoriel(1)
Out[9]: 1

In [10]: factoriel()
Out[10]: 1
```

Exemple III.5

```
def Bonjour(nom, pren="Mohammed"):
    print(f"Bonjour {nom} {pren}")
```

Résultats d'exécution

```
In [15]: Bonjour("Ben M'hidi","Larbi")
Bonjour Ben M'hidi Larbi

In [16]: Bonjour("Ben M'hidi")
Bonjour Ben M'hidi Mohammed
```

III. 7 Argument avec étiquette

Dans la plupart des langages de programmation, il est nécessaire de respecter l'ordre d'ordre d'affectation des arguments leur correspondent dans la définition de la fonction. Cependant, en Python il est possible d'affecter les arguments dans n'importe quel ordre en désignant le nom de chaque argument.

Exemple III.6

```
def Multi3val(x=1, y=0.1 , z=5 ):
    return x*y*z
```

Résultats d'exécution

```
In [18]: Multi3val(3,10,5)
Out[18]: 150
```

```
In [19]: Multi3val(x=3)
Out[19]: 1.5000000000000002

In [20]: Multi3val(y=3)
Out[20]: 15

In [21]: Multi3val(z=3)
Out[21]: 0.30000000000000004

In [22]: Multi3val(z=3,x=10)
Out[22]: 3.0
```

Exercice III.1

Définissez une fonction '*chercher_lettre*' qui détermine le nombre de répétition d'un caractère dans une chaine de caractère. Si aucun caractère est spécifié la fonction renvoi le nombre de répétions du caractère 'a'

Solution

```
1  def chercher_lettre(ph, ch="a"):
2      nbr=0
3      i=0
4      while i<len(ph):
5          if ph[i]==ch:
6              nbr=nbr+1
7          i=i+1
8      print(f"la phrase :\n {ph} contient {nbr} de {ch}")
```

Résultats d'exécution

```
In [36]: chercher_lettre("Département de l'Electronique", 'é')
la phrase :
 Département de l'Electronique contient 1 de é

In [37]: chercher_lettre("Département de l'Electronique", 'e')
la phrase :
 Département de l'Electronique contient 5 de e

In [38]: chercher_lettre("Département de l'Electronique", 't')
la phrase :
 Département de l'Electronique contient 3 de t

In [39]: chercher_lettre("Département de l'Electronique")
la phrase :
 Département de l'Electronique contient 1 de a
```