

Les fonctions



BENMANSOUR Asma

Table des matières



Introduction	3
I - Objectifs spécifiques du chapitre	4
II - Notion de sous programme	5
III - Notion de fonction: un ou plusieurs paramètres avec un retour	7
IV - Notion de procédure: un ou plusieurs paramètres pas de retour	8
V - Exercice : Procédure ou fonction ?	10
VI - Mécanisme de passage des arguments(paramètres)	11
VII -	
Exercice : Mettez dans l'ordre les instructions de la procédure miroir d'un nombre !	12
VIII - Paramètres avec valeur par défaut	13
IX - Portée des variables : variables locales/globales	15
X - Exercice : Quels sont les variables locales et les variables globales ?	18
Solutions des exercices	19
Bibliographie	21

Introduction



Il s'agit de vous présenter les éléments structurants de base de tout langage procédural: "les fonctions". Elles offrent de nombreux avantages comme le fait d'éviter la répétition, mettre en relief les données et les résultats, la réutilisation et enfin décomposer une tâche complexe en tâches plus simples.

Dans ce chapitre, nous allons étudier la syntaxe d'écriture d'une fonction, ensuite, nous allons expliquer le mécanisme de passage d'argument dans une fonction et traiter les différents cas à savoir: un ou plusieurs paramètres et pas de retour, un ou plusieurs paramètres avec utilisation du retour, le passage d'une fonction en paramètre, les paramètres avec valeur par défaut, etc.... Enfin nous allons étudier une notion fondamentale nécessaire à la compréhension du fonctionnement des variables à l'intérieur et à l'extérieur des fonctions, ils 'agit de la portée des variables. Dans ce contexte, nous allons expliquer la différence entre la portée globale et la portée locale.



Objectifs spécifiques du chapitre



A l'issu de ce chapitre l'apprenant sera capable de :

- Comprendre l'utilisation des sous programme (sous problème) dans le programme principale (problème globale)
- Mémoriser la syntaxe de définition d'une fonction et d'une procédure
- Comprendre la différence entre une fonction et une procédure lors de la résolution d'un sous problème
- Détecter la solution adéquate (fonction ou procédure) permettant de résoudre un sous problème du problème globale.
- Mettre en pratique la syntaxe d'écriture d'une fonction/procédure dans l'écriture d'un sous programme.
- Reconnaître et manipuler les variables locales et les variables globales dans le programme principale.

Notion de sous programme



Les programmes qu'on a écrits jusqu'à présent étaient courts car leur objectif était d'assimiler les premiers éléments du langage Python, les instructions qui les composaient s'exécutaient dans l'ordre de haut en bas.

Lorsque on commence à développer des programmes sophistiqués, on sera confronté à des problèmes souvent fort complexe et les lignes de programme vont commencer à s'accumuler, nous allons voir aussi que certains blocs d'instructions du programme se répètent tout au long du programme.

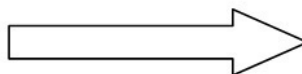
La solution consiste à définir des sous programmes contenant les parties de codes qui se répètent tout au début de notre programme principal et leur attribuer des noms; il s'agit d'une « mise en facteur commun » afin de pouvoir appeler le sous programme au besoin. Nous pouvons aussi utiliser ce même principe pour décomposer un programme long et complexe en sous programmes c'est-à-dire décomposer le problème en sous problèmes et faire un appel du sous programme défini plus haut au besoin.

Ces sous programmes qu'on va définir au tout début de nos programme afin de les appeler plus bas sont soit des fonctions soit des procédures. *p.21* ☺

Forme générale

```

instruction1
instruction 2
instruction 3
instruction 4
instruction 2
instruction 3
instruction 5
instruction 6
instruction 2
instruction 3
instruction 7
instruction 8
.
.
instruction 2
instruction 3
.
.
.
instruction N
  
```



Forme avec définition d'une fonction

```

def fonction(paramètres)
    instruction 2
    instruction 3

Instruction1
Appel fonction
instruction 4
Appel fonction
instruction 5
instruction 6
Appel fonction
instruction 7
instruction 8
.
.
Appel fonction
.
.
.
instruction N
  
```

Notion de sous programmes

Notion de fonction: un ou plusieurs paramètres avec un retour



Ensemble d'instructions regroupées sous un nom, pouvant contenir un ou plusieurs paramètres ou aucun paramètre et s'exécutant à la demande. On doit définir une *fonction* à chaque fois qu'un bloc d'instructions se répète plusieurs fois dans le programme et que ce bloc d'instruction *retourne un résultat*. p.21 ☺

Syntaxe : C'est une instruction composée :

```
def nom_fonction(paramètre1, paramètre2,..., paramètreN):
```

```
<insérez tabulation><bloc_instructions>
```

```
<insérez tabulation>return résultat
```

Exemple

Écrire une fonction permettant de le calcul du factorielle d'un entier passé en paramètres:

```
1 def factorielle(n):
2     f=1
3     for i in range(1,n+1):
4         f=f*i
5     return(f)
6
```

Complément

L'exemple ci-dessus montre une fonction qui retourne une seule valeur, mais les fonctions peuvent retourner plus d'une seule valeur, voici l'exemple de la définition de la même fonction ci-dessous avec un retour multiple:

```
1 def factorielle(n):
2     f=1
3     for i in range(1,n+1):
4         f=f*i
5     return(f,i,n)
6
```

Notion de procédure: un ou plusieurs paramètres pas de retour

IV

Ensemble d'instructions regroupées sous un nom pouvant contenir un ou plusieurs paramètres ou aucun paramètre et s'exécutant à la demande. On doit définir une procédure à chaque fois qu'un bloc d'instructions se répète plusieurs fois dans le programme et que ce bloc d'instruction ne retourne pas un résultat (sert à effectuer des calculs, à afficher un ou plusieurs messages). p.21

Syntaxe : C'est une instruction composée :

```
def nom_procédure (paramètre1, paramètre2,..., paramètreN):
```

```
<insérez tabulation><bloc_instructions>
```


Exemple

Écrire une procédure permettant d'afficher la table de multiplication d'un nombre entier donné en paramètre:

```
1 def table_multiplication(nombre):
2     for i in range(1,10):
3         print(nombre,"x",i,"=",nombre*i)
```

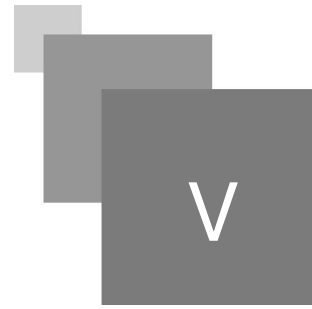
Voici un exemple d'appel de la procédure ci-dessus pour le calcul de la table de multiplication du nombre 4:

```
1 >>> table_multiplication(4)
2 4 x 1 = 4
3 4 x 2 = 8
4 4 x 3 = 12
5 4 x 4 = 16
6 4 x 5 = 20
7 4 x 6 = 24
8 4 x 7 = 28
9 4 x 8 = 32
10 4 x 9 = 36
11
```


 Remarque

1. Remarquer que la syntaxe de définition d'une fonction et le syntaxe de définition d'une procédure sont identiques à l'exception de l'instruction `return résultat` qui n'apparaît pas lors de la définition d'une procédure.
2. Le bloc d'instructions est une suite ordonnée d'instructions par exemple, `instruction1` puis retour à la ligne `instruction2` puis retour à la ligne `instruction3`. Le bloc d'instruction à l'intérieur de la définition d'une fonction ou d'une procédure est obligatoire. S'il est vide, on emploie l'instruction *pass*.
3. Que ce soit pour les fonctions ou les procédures, la définition des paramètres n'est pas obligatoire.

Exercice : Procédure ou fonction ?



[solution n°1 p.19]

Un sous programme permettant le calcul d'une surface se répète tout au long du programme principale, vous choisiriez une fonction ou une procédure pour le calcul de la surface d'un rectangle.

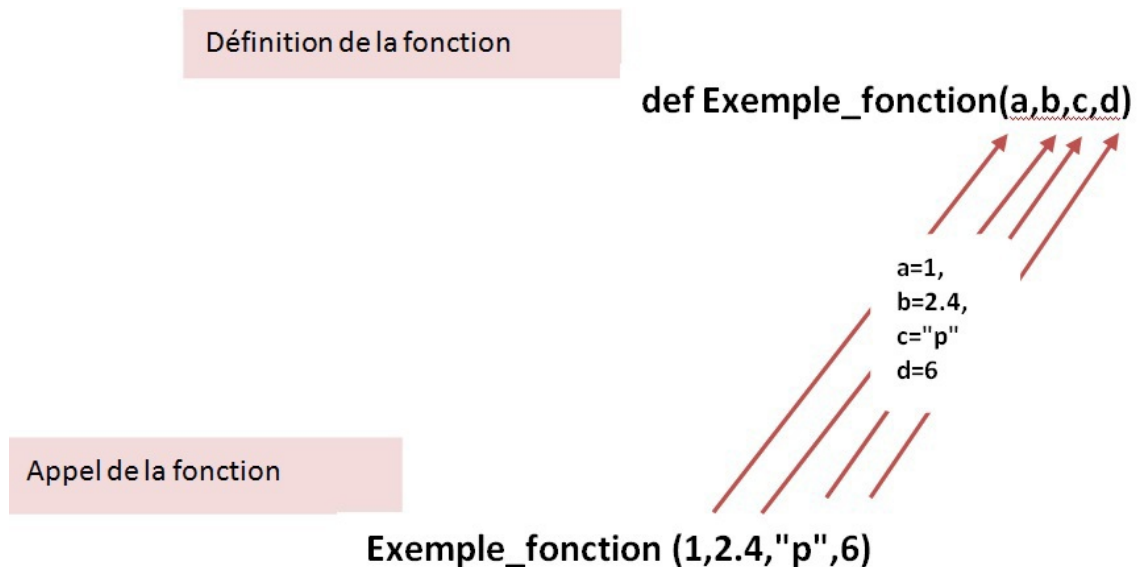
- Procédure
- Fonction

Mécanisme de passage des arguments(paramètres)

VI

Passage par affectation :

chaque argument inclut entre parenthèse lors de la définition de la fonction ou de la procédure correspond, dans l'ordre, à un paramètre de l'appel. La correspondance se fait par affectation (voir la figure ci-dessous):



Mécanisme de passage des arguments dans une fonction ou procédure

Exercice : Mettez dans l'ordre les instructions de la procédure miroir d'un nombre !

VII

[solution n°2 p.19]

Ci dessous les instructions servant à la définition d'une procédure qui retourne le miroir d'un nombre entier passé en paramètre, par exemple lors de l'appel de la procédure :

miroir(3245) va retourner *5423*

```
nombre=nombre//10
```

```
while nombre>0:
```

```
print(reste,end="")
```

```
reste=nombre%10
```

```
def miroir(nombre):
```

Paramètres avec valeur par défaut

VIII

Que ce soit pour les fonctions ou les procédures, on peut affecter à leur paramètres des valeurs par défaut:

- Si lors de l'appel de la fonction ou de la procédure il y'a un passage d'argument, ce sont les arguments passés en paramètres qui seront traités par la fonction ou la procédure.
- S'il n'y a pas d'arguments passés en paramètres lors de l'appel de la fonction ou de la procédure, le traitement se fera avec les valeurs par défaut affectées aux paramètres.

Exemple

Nous allons affecter la valeur 4 au paramètre nombre de la procédure `table_multiplication` de l'exemple utilisé précédemment:

```
1 def table_multiplication(nombre=4):
2     for i in range(1,10):
3         print(nombre, "x", i, "=", nombre*i)
4
```

Voici un exemple d'appel de la fonction sans argument:

```
1 >>> table_multiplication()
2 4 x 1 = 4
3 4 x 2 = 8
4 4 x 3 = 12
5 4 x 4 = 16
6 4 x 5 = 20
7 4 x 6 = 24
8 4 x 7 = 28
9 4 x 8 = 32
10 4 x 9 = 36
```

Python va afficher la table de multiplication du nombre 4 puisque le 4 est la valeur par défaut, par contre si on fait un appel à la fonction avec le nombre 7 par exemple:

```
1 >>> table_multiplication(7)
2 7 x 1 = 7
3 7 x 2 = 14
4 7 x 3 = 21
5 7 x 4 = 28
6 7 x 5 = 35
7 7 x 6 = 42
```

Paramètres avec valeur par défaut

$$8\ 7\ x\ 7 = 49$$

$$9\ 7\ x\ 8 = 56$$

$$10\ 7\ x\ 9 = 63$$

La procédure va tenir compte de la valeur passée en paramètre et non de la valeur affectée par défaut au paramètre.

Portée des variables : variables locales/globales

IX

Définitions

Nous parlons de portée des variables car les noms des objets sont créés lors de leur première affectation, mais ne sont visibles que dans certaines régions de la mémoire.

- Les variables locales sont les variables défini à l'intérieur d'une procédure ou d'une fonction ainsi que les paramètres associés à la procédure ou à la fonction.
- Les variables globales sont les variables défini dans le programme principal c'est à dire en dehors de la fonction ou de la procédure.

Exemple : Fonction qui calcule la factorielle d'un nombre

Écrire le programme python correspondant à une fonction qui calcule le factorielle d'un nombre entier passé en paramètre. Il faut ensuite utiliser cette fonction pour faire le calcul du factorielle de trois nombres entiers saisis pas l'utilisateur.

```
1 def factorielle(n):
2     f=1
3     for i in range(1,n+1):
4         f=f*i
5     return(f)
6
7 for i in range(3):
8     a=int(input("entrez le nombre dont vous voulez calculer le factorielle\t"))
9     print("le factoriel de a est:",factorielle(a))
```

- Les variables locales sont n,f
- Les variables globales sont a et i

cela veut dire qu'après l'exécution du programme si on fait un appel aux variables locales, leurs contenus ne sera pas affiché, comme suit :

```
1 entrez le nombre dont vous voulez calculer le factorielle 5
2 le factoriel de a est: 120
3 entrez le nombre dont vous voulez calculer le factorielle 4
4 le factoriel de a est: 24
5 entrez le nombre dont vous voulez calculer le factorielle 6
6 le factoriel de a est: 720
```

```

7 >>> n
8 Traceback (most recent call last):
9   File "<pyshell#4>", line 1, in <module>
10    n
11 NameError: name 'n' is not defined
12 >>> f
13 Traceback (most recent call last):
14   File "<pyshell#5>", line 1, in <module>
15    f
16 NameError: name 'f' is not defined
17 >>>

```

Dans ce qui suit l'affichage des variables globales après l'exécution du programme :

```

1 entrez le nombre dont vous voulez calculer le factorielle 2
2 le factoriel de a est: 2
3 entrez le nombre dont vous voulez calculer le factorielle 3
4 le factoriel de a est: 6
5 entrez le nombre dont vous voulez calculer le factorielle 4
6 le factoriel de a est: 24
7 >>> i
8 2
9 >>> a
10 4

```

1. Le dernier nombre utilisé comme argument pour la fonction est le nombre 4, donc la dernière valeur de a c'est 4
2. le i varie de 0 à 2 dans la boucle donc la dernière valeur que va prendre le i est 2



Attention

Si la même variable est utilisée dans la fonction ou procédure et dans le programme principal, Python considère la variable comme étant globale comme pour la variable i de l'exemple précédent.



Complément : Utilisation du mot clé global

Si on souhaite qu'un variable utilisé au sein d'une procédure ou une fonction soit considéré comme étant global, il suffit d'ajouter le mot global avant la variable lors de la définition de la procédure/fonction comme ceci :

```

1 def factorielle(n):
2     global f
3     f=1
4     for i in range(1,n+1):
5         f=f*i
6     return(f)
7
8 for i in range(3):
9     a=int(input("entrez le nombre dont vous voulez calculer le factorielle\t"))
10    print("le factoriel de a est:",factorielle(a))
11
12

```

Après l'exécution du programme nous tentons d'afficher f comme ceci :

```


```



```
1 entrez le nombre dont vous voulez calculer le factorielle 2
2 le factoriel de a est: 2
3 entrez le nombre dont vous voulez calculer le factorielle 3
4 le factoriel de a est: 6
5 entrez le nombre dont vous voulez calculer le factorielle 4
6 le factoriel de a est: 24
7 >>> f
8 24
```

Exercice : Quels sont les variables locales et les variables globales ?



[solution n°3 p.19]

```
def miroir(nombre):
```

- global reste
- while nombre>0:
 - ● reste=nombre%10
 - nombre=nombre//10
 - print(reste,end=")

```
a=int(input("entrez le nombre dont vous voulez calculer le miroir\t"))
```

```
miroir(a)
```

reste nombre a

variables locales	variables globales

Solutions des exercices



> Solution n° 1

Exercice p. 10

Un sous programme permettant le calcul d'une surface se répète tout au long du programme principale, vous choisiriez une fonction ou une procédure pour le calcul de la surface d'un rectangle.

- Procédure
- Fonction

> Solution n° 2

Exercice p. 12

Ci dessous les instructions servant à la définition d'une procédure qui retourne le miroir d'un nombre entier passé en paramètre, par exemple lors de l'appel de la procédure :

miroir(3245) va retourner 5423

```
def miroir(nombre):
    while nombre>0:
        reste=nombre%10
        nombre=nombre//10
    print(reste,end="")
```

> Solution n° 3

Exercice p. 18

```
def miroir(nombre):
```

- global reste
- while nombre>0:
 - ● reste=nombre%10
 - nombre=nombre//10
 - print(reste,end="")

```
a=int(input("entrez le nombre dont vous voulez calculer le miroir\t"))
```

```
miroir(a)
```

```
|-----|-----|
```



variables locales	variables globales
<div data-bbox="140 197 376 264" style="border: 1px dashed black; padding: 2px;">nombre</div>	<div data-bbox="399 197 651 264" style="border: 1px dashed black; padding: 2px;">reste</div> <div data-bbox="399 286 651 353" style="border: 1px dashed black; padding: 2px; margin-top: 10px;">a</div>

Bibliographie



[1] Swinnen Gérard, 02/02/2012, "Apprendre à programmer avec python 3", (3e édition), Eyrolles, 435 p, Noire

[2] Jachym, Marc, "Cours de Programmation avec le langage Python: Niveau débutant en programmation", Licence professionnelle – Métrologie dimensionnelle et qualité IUT de St Denis, Université Paris 13.

[3] Cordeau, Bob, Introduction à Python 3, version 2.71828.