CHAPITRE 5

Les fonctions

La Machine Analytique n'a nullement la prétention de créer quelque chose par elle-même. Elle peut exécuter tout ce que nous saurons lui ordonner d'exécuter. Elle peut suivre une analyse ; mais elle n'a pas la faculté d'imaginer des relations analytiques ou des vérités. Son rôle est de nous aider à effectuer ce que nous savons déjà dominer.

Ada Lovelace [1]

Thème : Algorithmique

Contenus	**Capacités attendues**
Constructions élémentaires	Mettre en évidence un corpus de constructions élémentaires.

I. Utilité d'une fonction

Il arrivera souvent qu'une même séquence d'instructions doive être utilisée à plusieurs reprises dans un programme, et on souhaitera bien évidemment ne pas avoir à la reproduire systématiquement.
D'autre part, l'approche efficace d'un problème complexe consiste souvent à le décomposer en plusieurs sous-problèmes plus simples qui seront étudiés séparément. Or il est important que cette décomposition soit représentée fidèlement pour que ceux-ci restent clairs.

Dans les langages de programmation, une fonction est une instruction isolée du reste du programme, qui possède un nom, et qui peut être appelée par ce nom à n'importe quel endroit du programme et autant de fois que l'on veut.

La notion de fonction en informatique est comparable à la notion de fonction en mathématiques.

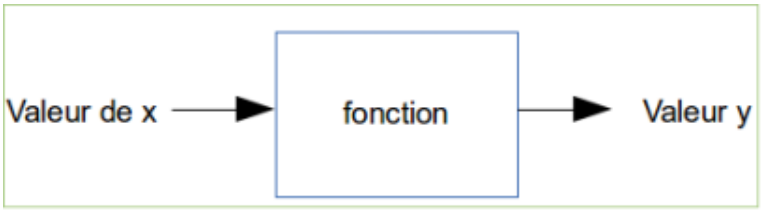

FIGURE 5.1 – Une fonction mathématique prend un antécédent (c) et renvoie une image (y)

Si nous avons le monôme y = 3x+2, pour une valeur donnée de x, nous aurons une valeur de y. Par exemple, pour x = 4 on aura y = 14. La fonction en informatique est basée sur la même idée :

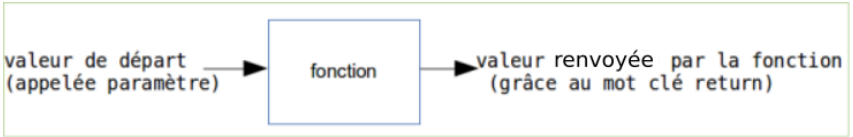

FIGURE 5.2 – Une fonction informatique prend une valeur et renvoie une nouvelle valeur

1. https://dicocitations.lemonde.fr/citations-auteur-ada_lovelace-0.php

II. Structure d'une fonction

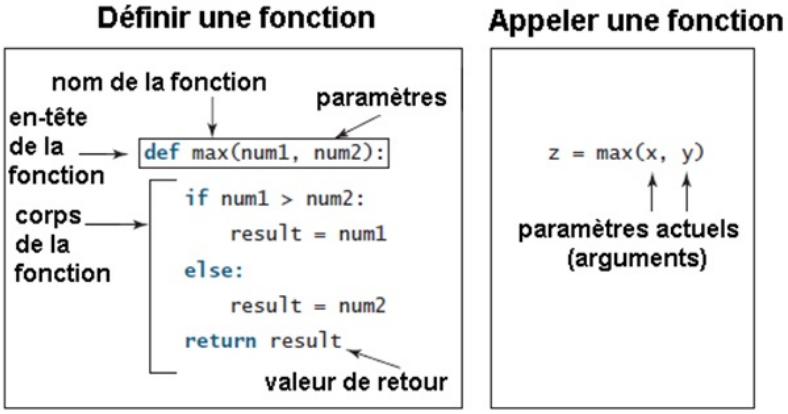

FIGURE 5.3 – Les deux étapes en informatique : la déclaration de la fonction et l'appel de la fonction

II. 1. Déclaration d'une fonction

La syntaxe Python pour la définition d'une fonction est la suivante :

```
def nomDeLaFonction(liste d'arguments):
        ...
        bloc d'instructions
        ...
```

Propriété 1

1. le mot-clé `def` permet de définir une fonction ;
2. le bloc d'instructions est indenté, c'est à dire décalé d'une tabulation.

Pour reprendre l'exemple précédent, la fonction s'écrirait :

```
1 def monome(x) :
2     resultat = 3*x + 2
3     return resultat
```

Bonnes pratiques

Les noms de fonctions doivent être en minuscules, les mots séparés par des underscores (_) si necessaire.

```
def ma_fonction():
```

II. 2. Passer des arguments dans une fonction

On appelle « **argument** » d'une fonction une variable particulière, utilisée dans le corps de la fonction, et dont la valeur est donnée dans le programme principal au moment où la fonction est appelée. Cela permet de communiquer des informations depuis le programme principal vers la

fonction.
Les arguments sont optionnels. On peut en avoir zéro, un ou plusieurs.

Exemple 1 (Exemple avec zéro argument)

```
def puissance():
    resultat = 5**2
    return resultat
```

Exemple 2 (Exemple avec un argument)

```
def puissance_2(x):
    resultat = x**2
    return resultat
```

Exemple 3 (Exemple avec plusieurs arguments)

```
def puissance_2(x, exposant):
    resultat = x**exposant
    return resultat
```

Activité 1.

1. Écrivez une fonction *polynome*, prenant x en argument, et renvoyant la valeur du polynome $3x^7 + 24x - 2$.
2. Écrivez une fonction *polynome_2*, prenant x, a, exp_a, b, exp_b, c en argument, et renvoyant la valeur du polynome $ax^{exp\_a} + bx^{exp\_b} + c$.

II. 3. Utilisation d'une fonction dans le corps du programme

L'exécution d'une fonction se fait en appelant la fonction, suivie de ses arguments éventuels.

Exemple 4 (Exemple avec zéro argument)

```
>>> puissance()
25
```

Exemple 5 (Exemple avec un argument)

```
>>> puissance_2(5)
25
>>> puissance_2(2)
4
```

Exemple 6 (Exemple avec plusieurs arguments)

```
>>> puissance_3(5,2)
25
>>> puissance_2(5,3)
125
```

Activité 2.
En utilisant la fonction *polynome_2*, écrivez l'instruction Python permettant de renvoyer la valeur du polynôme $3x^7 + 24x - 2$ lorsque x prend la valeur 5.

II. 4. Arguments par défaut

Pour appeler une fonction, il faut que le nombre d'arguments à passer soit respecté.
Dans la fonction :

```
def puissance_3(x, exposant):
    resultat = x**exposant
    return resultat
```

deux arguments sont utilisés. Pour appeler la fonction, il faut donc passer deux arguments. Dans le cas où l'utilisateur oublie un arguments, le programme plante !

Exemple 7 (Seul un argument a été passé au lieu de deux)

```
>>> puissance_3(5)
Traceback (most recent call last):

File "<ipython-input-6-9ee0b781a940>", line 1, in <module>
puissance_3(5)

TypeError: puissance_3() missing 1 required positional argument: 'exposant'
```

Python vous permet de définir des fonctions avec des valeurs d'argument par défaut. Les valeurs par défaut sont transmises aux paramètres lorsque la fonction est appelée sans arguments.

```
def puissance_3(x, exposant=2):
    resultat = x**exposant
    return resultat
```

Cette structure indique que si le paramètre « **exposant** » n'est pas indiqué dans l'appel de fonction, il prend par défaut la valeur 2. `puissance_3(5)` renvoie le même résultat que `puissance_3(5,2)`.

II. 5. Récupérer une valeur dans le corps du programme

On veut souvent communiquer les informations de la fonction au programme principal. L'instruction `return` permet de renvoyer un résultat dans le corps du programme, afin d'être utilisée pour un calcul, ou pour être stockée dans une variable. Il est possible de renvoyer plusieurs valeurs.

Exemple 8 (Deux valeurs sont retournées par la fonction)

```
1 def monome(x, a, b):
2     total = a*x + b
3     double_total = 2*total
4     return total, double_total
```

Attention

L'exécution du corps d'une fonction **S'ARRÊTE** lorsque l'instruction `return` est exécutée. Il peut y avoir plusieurs instructions `return` dans une fonction, mais la première qui est exécutée arrête la fonction.

Exemple 9 (Les lignes 4 et 5 ne sont JAMAIS exécutées)

```
1 def monome(x, a, b):
2     total = a*x + b
3     return total
4     double_total = 2*total
5     return double_total
6
7 >>> monome(5, 3, 2)
8 17
```

Activité 3.
Écrivez l'instruction Python permettant de stocker dans une variable *image*, la valeur du polynôme $3x^7 + 24x - 2$ lorsque x prend la valeur 5.

II. 6. Fonction et procédure

Un fonction ne renvoyant pas de valeur (absence du mot-clé `return`) est appelé une procédure. Python ne fait pas de différence entre la déclaration d'une fonction et la déclaration d'une procédure. Ce n'est pas le cas de tous les langages (Java par exemple).

```
void afficher(){
        System.out.println("Coucou !");
        }
```

Définition d'une procédure par le mot clé `void`

```
int somme3(int a, int b, int c){
        return a+b+c;
}
```

Définition d'une fonction par le mot clé `int`

III. Portée des variables

Une variable créée à l'intérieur d'une fonction est appelée une variable locale. Les variables locales ne sont accessibles que dans une fonction. La portée d'une variable locale commence à partir de sa création et continue jusqu'à la fin de la fonction qui contient la variable.

En Python, vous pouvez également utiliser des variables globales. Ils sont créés en dehors de toutes les fonctions et elles sont accessibles à toutes les fonctions dans leur champ d'application.

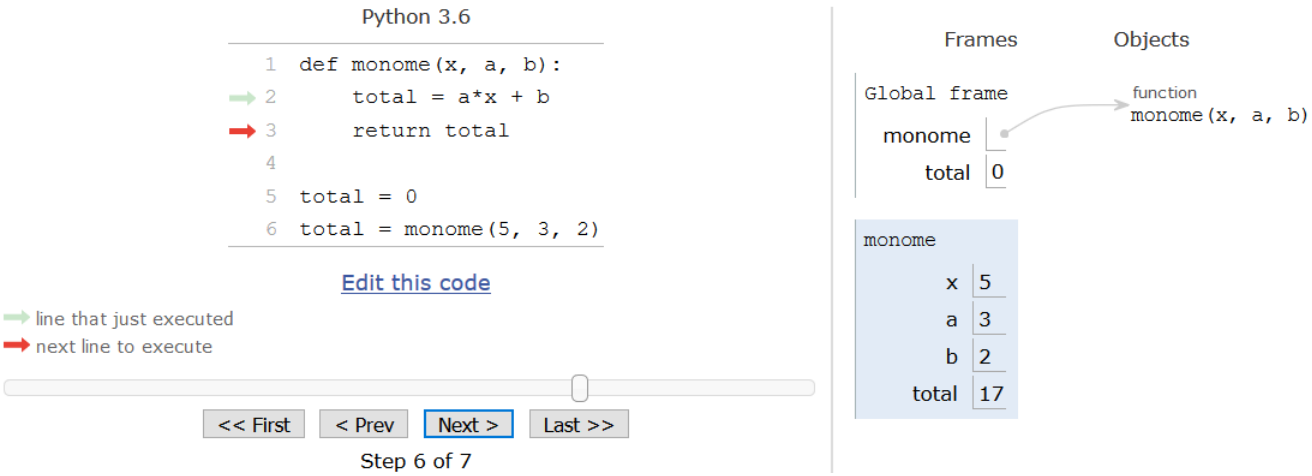

FIGURE 5.4 – La variable total est définie deux fois : une première fois comme variable globale ("Global frame") ; une deuxième fois comme variable locale ("monome" dans le cadre grisé)
`https://urlz.fr/dRva`

Propriété 2

- Si une variable est déclarée en dehors de la fonction, alors elle est globale ;
- si une variable est déclarée à l'intérieur d'une fonction, alors sa portée est limitée à cette fonction ;
- si une variable fait partie des arguments d'une fonction, alors sa portée est limitée à cette fonction ;
- si deux variables de même nom sont déclarés dans deux fonctions différentes, alors elles représentent en réalité deux boîtes distinctes, et la portée de chacune est limitée à la fonction correspondante. Dans ce cas, elles peuvent même avoir des types différents.

Bonnes pratiques

Pour des raisons de lisibilité du code, il faut éviter d'avoir le même nom pour une variable locale et pour une variable globale.

Je retiens

- ▷ Une fonction permet d'utiliser la même séquence d'instructions à plusieurs reprises dans le corps du programme ;
- ▷ une fonction permet de décomposer un problème complexe en plusieurs sous-problèmes plus simples ;
- ▷ la déclaration d'une fonction se fait avec le mot-clé `def`, suivi du nom de la fonction, de ses arguments placés entre les parenthèses, et d'un double-point pour finir la ligne ;
- ▷ le bloc d'instructions à exécuter dans la fonction est indenté, c'est-à-dire décalé d'une tablulation ;
- ▷ l'utilisation de la fonction dans le corps du programme se fait en saisissant le nom de la fonction, suivi des arguments placés entre paranthèses. Il faut respecter le nombre d'arguments passés ;
- ▷ si une variable est déclarée à l'intérieur d'une fonction, alors sa portée est limitée à cette fonction ;
- ▷ si une variable est déclarée à l'extérieur d'une fonction, alors sa portée est globale ; elle peut être utilisée dans et hors de la fonction ;

IV. Exercices

Exercice 1 (QCM).
On considère la fonction suivante :

```
1 def diff(val1,val2):
2     return (val2 - val1)
```

Qu'affiche la console lors de l'exécution de `diff(3.0,-3.0)` ?

- ☐ A : 6.0
- ☐ B : -6.0
- ☐ C : 0
- ☐ D : "diff(3.0,-3.0)"

Exercice 2 (QCM).

```
1 def func(val):
2     if val<0.0:
3         return 0
4     return val
```

Qu'affiche la console lors de l'exécution de `func(-2.5)` ?

- ☐ A : 0
- ☐ B : -2.5
- ☐ C : "val"
- ☐ D : False

Exercice 3.
On considère la fonction suivante :

```
1 def func(a):
2     a = a + 3.0
3     return a
```

Qu'affiche la console lors de l'exécution de la séquence suivante ?

```
1 >>> a = 7.0
2 >>> b = func(a)
3 >>> print(a,b)
```

Exercice 4 (Écriture de fonctions simples).

1. Écrivez une fonction *le_plus_grand* qui prend deux nombres en argument, et qui renvoie le plus grand des deux.
2. Écrivez une fonction *le_plus_grand_2* qui prend en argument trois nombres et renvoie le plus grand des trois.

Exercice 5 (Année bisextile).
Selon Le Petit Robert, une anné bissextile revient généralement tous les quatre ans . Plus sérieusement, et depuis l'instauration du calendrier grégorien le 15 octobre 1582, est bissextile une année, soit divisible par 4 mais pas par 100, soit divisible par 400.
Écrivez une fonction *est_bisextile* qui prend une année en argument, et qui renvoie `True` si l'année est bisextile et `False` sinon.

Exercice 6 (Polynôme de degré 2).
La racine d'un polynôme est la valeur de x qui annule ce polynôme. Dans le cas d'un polyôme du deuxième degré $P(s) = ax^2 + bx + C$, on calcule le déterminant $\Delta = b^2 - 4ac$.

- si $\Delta < 0$, le polynôme n'admet pas de racine ;
- si $\Delta = 0$, le polynôme admet une racine unique : $x = -\dfrac{b}{2a}$;
- si $\Delta > 0$, le polynôme admet deux racines : $x_1 = -\dfrac{b+\sqrt{\Delta}}{2a}$ et $x_1 = -\dfrac{b-\sqrt{\Delta}}{2a}$.

Écrivez une fonction *racines*, prenant a, b, c en paramètre, et renvoyant les valeurs de la racine si elle existe, et -1 sinon.
la fonction $\sqrt{x}$ s'écrit en Python `math.sqrt(x)`, après avoir écrit au préalable `import math`.

Exercice 7 (Géométrie plane).

1. Écrivez une fonction $est\_triangle$, prenant a, b, c les trois côtés du triangle, et qui renvoie `True` si la construction d'un tel triangle est possible, et `False` sinon.
2. Écrivez une fonction $est\_triangle\_rectangle$, prenant a, b, c les trois côtés du triangle, et qui renvoie `True` si la construction d'un tel triangle est possible ET si le triangle est rectangle, et `False` sinon.
3. Écrivez une fonction $est\_triangle\_isocle$, prenant a, b, c les trois côtés du triangle, et qui renvoie `True` si la construction d'un tel triangle est possible ET si le triange est isocèle, et `False` sinon.