PYTHON AU LYCÉE (4) : LES FONCTIONS

COURS

Python au lycée (4) : Les fonctions

DURÉE ESTIMÉE

10 minutes

OBJECTIFS DU CHAPITRE

- Définir et appeler une fonction avec def
- Utiliser return pour renvoyer une valeur
- Écrire une fonction à plusieurs paramètres
- Programmer une fonction mathématique en Python
- Importer et utiliser un module (math, random)

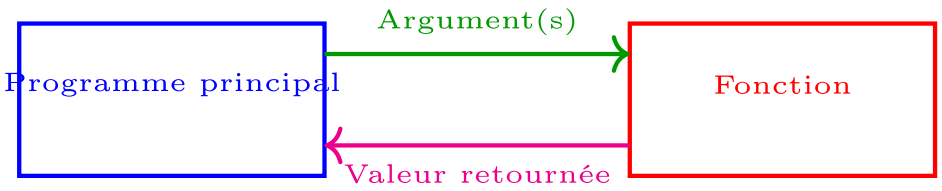
1. Fonctions en programmation

Lorsqu'un groupe d'instructions se répète plusieurs fois dans un programme, il peut être utile de regrouper ces instructions à l'intérieur de *fonctions.*

Cela permet de diminuer la taille du programme, de faciliter sa maintenance et de le rendre plus lisible en fractionnant le code en blocs indépendants. Par ailleurs, un programmeur peut ainsi utiliser un code écrit par une autre personne.

Dans les chapitres précédents, nous avons déjà utilisé des fonctions comme *print()* ou *len()* qui sont des fonctions internes à Python.

L'appel à une fonction peut être schématisé de la manière suivante :



- Le programme appelle la fonction en lui passant éventuellement certaines valeurs appelées *arguments* ou *paramètres*. Il peut y avoir zéro, un ou plusieurs paramètre(s) de n'importe quel type.
- La fonction effectue certaines opérations en utilisant, au besoin, les valeurs passées en paramètre.
- La fonction peut ensuite retourner un résultat au programme qui l'a appelé ; le retour d'une valeur n'est toutefois pas obligatoire.

Considérons, par exemple, l'instruction :

```
x=len("Bonjour")
```

L'exécution de cette commande se déroule de la manière suivante :

- le programme appelle la fonction Python *len()* en lui passant le paramètre « Bonjour » de type chaîne de caractères ;
- la fonction *len()* calcule la longueur de cette chaîne ;
- la fonction renvoie l'entier 7 qui correspond à cette longueur ;
- le résultat est stocké dans la variable *x* qui pourra être utilisé par la suite.

Dans les exemples précédents, les fonctions ont été appelées à partir du programme principal. Cependant, une fonction peut être appelée à partir de n'importe où ; en particulier, une fonction peut très bien être appelée par une autre fonction.

2. Définition et appel d'une fonction en Python

En plus des fonctions prédéfinies par Python, le programmeur a la possibilité de définir ses propres fonctions.

On utilise le mot-clé *def* pour définir une nouvelle fonction avec la syntaxe suivante :

```
def nom_de_la_fonction(liste_des_paramètres):
    # bloc d'instructions
    # calculant le résultat
    return resultat # facultatif
```

Par exemple :

```
def ajoute(x,y):
    somme=x+y
    return somme
```

Les paramètres passés à la fonction sont placés entre parenthèses. Même si la fonction n'admet pas d'argument, il faut utiliser des parenthèses ; dans ce cas on ne place rien entre les parenthèses.

Lorsque l'on définit une fonction en Python, aucune instruction n'est exécutée. L'exécution des instructions situées dans le corps de la fonction n'a lieu que lors de l' **appel** de la fonction.

L'appel d'une fonction en Python obéit à la syntaxe suivante :

```
nom_de_la_fonction(valeurs_des_paramètres)
```

Le résultat est alors souvent stocké dans une variable :

```
resultat=nom_de_la_fonction(valeurs_des_paramètres)
```

ou affiché à l'écran :

```
print(nom_de_la_fonction(valeurs_des_paramètres))
```

Par exemple, pour utiliser la fonction *ajoute* définie précédemment :

```
x=ajoute(4,3) # x contient la valeur 7
```

Exemple : fonction calculant la moyenne de trois notes

Une fonction peut recevoir plusieurs arguments et renvoyer le résultat d'un calcul effectué à partir de ces arguments. Par exemple, la fonction *moyenne* ci-dessous calcule la moyenne de trois notes :

```
def moyenne(a, b, c) :
    m = (a + b + c) / 3
    return m

print(moyenne(12, 15, 9))   # affiche 12.0
print(moyenne(8, 11, 14))   # affiche 11.0
```

Exemple : fonction définie par morceaux

Une fonction Python peut utiliser une instruction conditionnelle pour renvoyer un résultat qui dépend de la valeur de son argument. Par exemple, la fonction *f* mathématique définie par $f(x) = -x + 1$ si $x \leq 0$ et $f(x) = 3x + 1$ si $x > 0$ peut être programmée ainsi :

```
def f(x) :
    if x <= 0 :
        return -x + 1
    else :
        return 3*x + 1

print(f(-2))   # affiche 3
print(f(5))    # affiche 16
```

Fonction avec ou sans *return*

Il est important de distinguer :

- une fonction qui **renvoie** une valeur grâce à *return* : cette valeur peut ensuite être utilisée dans un calcul ou stockée dans une variable.
- une fonction qui se contente d'**afficher** un résultat avec *print* : cette fonction ne renvoie rien (techniquement, elle renvoie la valeur spéciale *None*).

Par exemple :

```
def carre_return(x) :
    return x*x       # renvoie la valeur

def carre_print(x) :
    print(x*x)       # affiche la valeur mais ne la renvoie pas

a = carre_return(5)   # a vaut 25, on peut l'utiliser
b = carre_print(5)    # affiche 25, mais b vaut None
```

3. Les modules Python

En Python, il est possible de regrouper des fonctions (ou d'autres objets) dans des fichiers appelés *modules* afin de pouvoir les réutiliser par la suite dans d'autres programmes.

Un certain nombre de modules sont intégrés à Python mais il est nécessaire de les charger dans son programme afin de pouvoir les utiliser. Pour charger un module on peut utiliser la syntaxe suivante :

```
import nom_du_module
```

Par la suite on peut utiliser une fonction de ce module en la préfixant par le nom du module de la manière suivante :

```
resultat = nom_du_module.nom_de_la_fonction # Noter la présence
```

Par exemple pour utiliser la fonction *sqrt()* (racine carrée) du module *math* :

```
import math
print(math.sqrt(9)) # affiche 3.0
```

Pour éviter de devoir préfixer la fonction à chaque fois qu'on l'utilise, on peut choisir de charger cette fonction lors de l'*import* en utilisant la syntaxe suivante :

```
from math import sqrt
print(sqrt(9)) # affiche 3.0
```

On peut également importer tous les objets d'un module en utilisant une * :

```
from math import *
print(sqrt(9)) # affiche 3.0
print(cos(pi)) # affiche -1.0
```

Parmi les nombreux modules intégrés à Python, les modules *math* et *random* seront fréquemment utilisés au lycée :

- **Le module « math »** :
  Ce module contient différentes fonctions et constantes mathématiques : *pi, sqrt(), exp(), ln(), sin(), cos(), ...*
- **Le module « random »** :
  Ce module, utile en statistiques et en probabilités, permet de générer des nombres aléatoires.
- random() renvoie un nombre réel aléatoire appartenant à l'intervalle [0 ; 1[(borne 1 exclue).

- randint(min, max) renvoie un nombre entier aléatoire compris entre les entiers min et max (bornes incluses).

Par exemple on peut simuler dix lancers d'un dé à six faces grâce au programme suivant :

```
from random import randint
for i in range(10) :
    print(randint(1,6))
```

Application : simulation d'une expérience aléatoire

En combinant une fonction, une boucle et une instruction conditionnelle, on peut simuler un grand nombre d'expériences aléatoires puis en observer la fréquence. Par exemple, la fonction *frequence_six* ci-dessous simule *n* lancers d'un dé équilibré et renvoie la fréquence d'apparition du 6 :

```
from random import randint

def frequence_six(n) :
    nb_six = 0
    for i in range(n) :
        if randint(1, 6) == 6 :
            nb_six = nb_six + 1
    return nb_six / n

print(frequence_six(100))    # par exemple 0.18
print(frequence_six(10000))  # par exemple 0.1672
```

Sur un grand nombre de lancers, la fréquence observée se rapproche de la probabilité théorique $\frac{1}{6} \approx 0{,}1667$: c'est une illustration de la **loi des grands nombres**.

Les questions essentielles

1. Comment définir et appeler une fonction avec def ?

On écrit *def nom_fonction(paramètres) :* suivi du corps indenté. On appelle ensuite la fonction en écrivant son nom suivi des arguments entre parenthèses.

Voir la fiche méthode : Définir et appeler une fonction avec def

2. Comment utiliser return pour renvoyer une valeur ?

return valeur quitte la fonction et transmet la valeur au programme qui l'a appelée. À ne pas confondre avec *print*, qui affiche sans rien renvoyer.

Voir la fiche méthode : Utiliser return pour renvoyer une valeur

3. Comment écrire une fonction à plusieurs paramètres ?

On sépare les paramètres par des virgules dans l'en-tête : *def f(a, b, c) :*. Lors de l'appel, les arguments sont affectés aux paramètres dans l'ordre.

Voir la fiche méthode : Écrire une fonction à plusieurs paramètres

4. Comment programmer une fonction mathématique en Python ?

On traduit la formule en langage Python (en pensant à l'opérateur *** et à la puissance ****), puis on la place dans le corps de la fonction avec un *return*. Pour une fonction par morceaux, on utilise *if ... else*.

Voir la fiche méthode : Programmer une fonction mathématique en Python

5. Comment importer et utiliser un module (math, random) ?

On écrit *import math* (puis on préfixe : *math.sqrt(9)*) ou *from math import sqrt* (puis appel direct : *sqrt(9)*). Même principe pour *random* avec *randint*.

Voir la fiche méthode : Importer et utiliser un module (math, random)

Télécharger en PDF