Python les Fonctions

Def (), def(alb, c), return, variable « locale », « globale », récursivité …

Présentation des fonctions

À quoi servent les fonctions en pratique ? Elles permettent de réutiliser du code et évitent ainsi de la duplication, tout en rendant le programme plus lisible. Les fonctions permettent également de structurer un programme, fournissant ainsi une documentation implicite de ce dernier.

Lorsque vous devez écrire du code très similaire à du code déjà écrit, on pourrait sans doute croire que la meilleure solution consiste à copier-coller un bout de code, puis de l'adapter. Il s'agit pourtant d'une très mauvaise pratique, pour deux raisons principales :

- les chances de se tromper à un moment lors des copier-coller ou des adaptations de ces derniers sont grandes ;
- si une erreur est détectée dans le code qui a été copié-collé, il va falloir la corriger partout là où ce code a été copié-collé.
- Les fonctions permettent de est de structurer un programme. Grâce à ces dernières, on va pouvoir découper un gros programme en petits blocs, chacun plus simple à comprendre.

Nous avons déjà rencontré diverses fonctions prédéfinies : `print()`, `input()`, `range()`, `len()`.

On sait maintenant écrire des programmes Python avancés, mais ils vont vite devenir longs en nombre de lignes de code. De plus, on risque très vite de se retrouver avec des répétitions de codes similaires. Avec les fonctions on va pouvoir écrire du code plus compact, lisible et réutilisable.

Syntaxe

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

```
def nom_fonction(liste de paramètres):
    bloc d'instructions
```

Une fonction se définit avec le mot réservé `def`, suivi de son nom, d'une liste de paramètres (qui peut être vide), du caractère deux-points (:) et enfin d'un bloc de code représentant son corps. Une fois définie, elle peut être utilisée autant de fois qu'on le souhaite, en l'appelant.

On peut classifier les fonctions selon deux critères. Une fonction peut renvoyer une valeur ou non, au terme de son exécution, et une fonction peut admettre ou non des paramètres.

Remarque :

Comme les instructions `if`, `for` et `while`, l'instruction `def` est une instruction composée. La ligne contenant cette instruction se termine obligatoirement par un deux-points `:`, qui introduisent un bloc d'instructions qui est précisé grâce à l'indentation. Ce bloc d'instructions constitue le **corps de la fonction**.

Remarque :

Lorsqu'on définit des fonctions, c'est évidemment dans le but de les utiliser. Il est donc très important de les documenter, c'est-à-dire d'ajouter des commentaires expliquant ce qu'elles font, et comment les utiliser.

```
1.  # Fonction permettant de tester si le
2.  nombre entier d
3.  # est un diviseur du nombre entier n
4.  def isDivisor(d, n):
        return n % d == 0
```

Les fonctions qui ne renvoie pas de valeur et n'admet aucun paramètre

Ces lignes de code définissent donc une fonction dont le nom est table7. La fonction initialise une variable n à 1, puis une boucle se répète tant que la condition n <= 10 est vraie. Le corps de la boucle affiche une ligne de la table de multiplication, puis incrémente la valeur de n d'une unité.

```
1.  def table7():
2.      n = 1
3.      while n <= 10:
4.          print(n, "x 7 =", n * 7)
5.          n += 1
```

Pour exécuter cette fonction, il suffit simplement d'utiliser son nom, suivi d'une parenthèse ouvrante et d'une fermante. L'instruction d'appel de fonction `table7()`

```
1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
5 x 7 = 35
```

```
6 x 7 = 42
7 x 7 = 49
8 x 7 = 56
9 x 7 = 63
10 x 7 = 70
```

Attention :

Un fichier Python est analysé par l'interpréteur ligne par ligne. Dès lors, un appel de fonction ne peut pas se faire avant que celle-ci n'ait été définie, sans quoi l'interpréteur génèrera une erreur vous signalant qu'il ne parvient pas à trouver la fonction demandée :

```
Traceback (most recent call last):
  File "program.py", line 1, in <module>
    table7()
NameError: name 'table7' is not defined
```

Les fonctions à un paramètre

Le grand avantage de cette fonction est qu'elle est beaucoup plus générique, c'est-à-dire qu'elle pourra fonctionner dans beaucoup plus de cas. Pour l'appeler, on utilise de nouveau son nom, sans oublier de fournir une valeur à son paramètre(l'argument), entre parenthèses.

```
1.  def table(base):
2.      n = 1
3.      while n <= 10:
4.          print(n, "x", base, "=", n
5.  * base)
            n += 1
```

Les fonctions à plusieurs paramètre

La fonction suivante utilise trois paramètres : `start` qui contient la valeur de départ, `stop` la borne supérieure exclue et `step`

```
def compteur_complet(start, stop, step):
    i = start
    while i < stop:
        print(i)
        i = i + step

compteur_complet(1, 7, 2)
```

→

```
1
3
5
```

Note :

- Pour définir une fonction avec plusieurs paramètres, il suffit d'inclure ceux-ci entre les parenthèses qui suivent le nom de la fonction, en les séparant à l'aide de virgules
- Lors de l'appel de la fonction, les arguments utilisés doivent être fournis dans le même ordre que celui des paramètres correspondants (en les séparant eux aussi à l'aide de virgules). Le premier argument sera affecté au premier paramètre, le second argument sera affecté au second paramètre, et ainsi de suite. Il est nécessaire de fournir autant d'argument que de paramètre qu'il faut lors de son appel.
- S'il manque un argument ou que celui-ci est mal placé,

un message d'erreur apparaitra !

```
1.  Traceback (most recent call last):
2.    File "program.py", line 7, in <module>
3.      table(8)
4.  TypeError: table() missing 2 required
    positional arguments: 'start' and
    'length'
```

Les fonctions à plusieurs paramètre avec valeur par défaut

- Pour éviter les messages d'erreur lorsque l'on oubli de renseigner tous les arguments d'une fonction à plusieurs paramètre, il est nécessaire de définir dans les paramètres des valeurs par défaut. C'est-à-dire celle qu'ils auront lors de l'appel si on n'en spécifie pas une autre. En Python, c'est simple, il suffit de déclarer ces valeurs par défaut lors de la définition de la fonction :

```
def compteur_complet(start, stop=5, step=1):
    i = start
    while i < stop:
        print(i)
        i = i + step

compteur_complet(1)
```

```
>>>
1
2
3
4
```

- Les valeur par défaut sont 5 « exclu » pour stop et 1 pour le pas.

Les fonctions avec valeur de retour

Une fonction peut également renvoyer une valeur qu'il est possible de récupérer lorsqu'on l'appelle.

```
1.  def multiply(a, b):
2.      return a * b
```

Pour avoir accès à la valeur de retour, il faut la stocker dans une variable lors de l'appel sinon on ne récupère pas cette valeur renvoyée lors de l'appel, et dès lors ce dernier est complètement inutile. Le résultat calculé par la fonction est tout simplement perdu à jamais.

```
1.  res = multiply(7, 9)
2.  print(res)
```

Les variables locales et globales:

Lorsqu'on travaille avec des fonctions, il faut distinguer deux sortes de variables : les locales et les globales. Une variable globale est définie pour tout le programme ; elle est initialisée en dehors de toute fonction. Une variable locale est définie uniquement dans le corps d'une fonction, celle où elle a été initialisée.

```
def Prix_TCC(Prix_HT):
    n = 1 + Taux_TVA/100
    return Prix_HT*n

Taux_TVA=21 # en pourcent
n=25 #prix hors taxe
print(Prix_TCC(n))
```

```
>>> %Run
30.25
```

Lors de l'exécution de ce programme, il y a en fait deux variables n qui vont exister en même temps :

- Celle initialisée à la deuxième ligne est une variable locale à la fonction Prix_TCC. Elle n'existe que dans le corps de la fonction, durant le temps où elle est exécutée et disparait ensuite de la mémoire.
- Celle initialisée à l'avant-dernière instruction est une variable globale. Elle existe dans tout le programme, depuis son initialisation jusque la fin de l'exécution du programme.

Les fonctions récursives

On peut également appeler une fonction depuis son propre corps. Une telle fonction, qui s'appelle elle-même, est appelée fonction récursive.

La récursivité est un moyen de répéter des blocs d'instructions sans utiliser de boucle while ou for.

Exemple
En Python, la fonction puissance(y, n) implémente le calcul de y^n (pour y un nombre et n un entier positif)

Python	Explication
def puissance(y, n):	On définit la fonction puissance.
if n == 0:	Si n est égal à 0, alors
return 1	on retourne 1 (car y^0=1).
else:	Sinon
return y*puissance(y, n-1)	on retourne : y*puissance(y,n-1)

Remarque :
Lors de l'exécution d'un algorithme récursif, les appels récursifs successifs sont stockés dans une pile, c'est la **pile d'exécution**.

Plus précisément, la pile d'exécution est un emplacement mémoire destiné à stocker les paramètres, les variables locales ainsi que les adresses mémoires de retour des fonctions en cours d'exécution.

Une fonction récursive peut rapidement être gourmande en mémoire. C'est pourquoi on peut obtenir le message d'erreur :

RuntimeError: maximum recursion depth exceeded

La taille de la pile d'exécution est limitée. Par défaut, le nombre maximum d'appels récursifs autorisés est fixé à 1000.

Utiliser une fonction récursive n'est pas toujours judicieux, il est préférable parfois plus efficace d'utiliser des itérations.