

# **Structure d'un programme - fonctions**

# Table des matières

<b>I. Fonctions def et return</b>	<b>3</b>
A. Définition d'une fonction .....	3
B. Valeurs de retour .....	5
<b>II. Exercice : Quiz</b>	<b>5</b>
<b>III. Fonctions natives et les fonctions lambda</b>	<b>6</b>
A. Fonctions natives .....	6
B. Fonction Lambda .....	7
C. Compléments .....	8
<b>IV. Exercice : Quiz</b>	<b>9</b>
<b>V. Essentiel</b>	<b>10</b>
<b>VI. Auto-évaluation</b>	<b>10</b>
A. Exercice .....	10
B. Test .....	10
<b>Solutions des exercices</b>	<b>11</b>

## I. Fonctions def et return

### Contexte

Les fonctions Python sont simples à définir et essentielles à la programmation de niveau intermédiaire. Les critères exacts s'appliquent aux noms de fonctions, comme aux noms de variables. Le but est de regrouper certaines actions souvent réalisées, et de définir une fonction. Plutôt que de réécrire le même bloc de code, encore et encore, pour diverses variables d'entrée, nous pouvons appeler la fonction et réutiliser le code qu'elle contient, avec différentes variables.

Dans Python, il existe des fonctions natives, telles que `str()`, `input()`, `print()`, `len()`. Ces fonctions sont créées nativement dans Python, vous pouvez donc les utiliser très rapidement. Elles vous seront très utiles dans l'écriture de vos programmes. Cependant, comme tout langage de programmation, vous pouvez écrire vos propres fonctions, pour ensuite les réutiliser dans votre programme.

Une fonction permet de réaliser une action bien précise. Un programme, lui, est constitué de plusieurs fonctions qui, mises bout à bout, effectuent une action bien précise. Dans ce cours, nous allons voir comment écrire un programme de la meilleure manière qui soit.

### A. Définition d'une fonction

#### Définition

#### Une fonction

En Python, une fonction est un groupe d'instructions liées qui exécutent une tâche spécifique. Les fonctions aident à diviser notre programme en morceaux plus petits et modulaires. Au fur et à mesure que notre programme grandit, les fonctions le rendent plus organisé et plus gérable. De plus, cela évite les répétitions et cela rend le code réutilisable.

#### Exemple

#### Syntaxe de la fonction

```
1 def nom_fonction(paramètres) :  
2     """docstring"""  
3     déclaration(s)
```

Vous voyez ici une définition de fonction composée des composants suivants :

- Un mot-clé `def` qui marque le début de l'en-tête de la fonction.
- Un nom de fonction pour identifier de manière unique la fonction (la dénomination des fonctions suit les mêmes règles d'écriture que les identifiants en Python).
- Des paramètres (arguments) par lesquels nous passons des valeurs à une fonction (ils sont facultatifs).
- Deux points (:) pour marquer la fin de l'en-tête de la fonction.
- Une chaîne de documentation facultative (`docstring`) pour décrire ce que fait la fonction.
- Une ou plusieurs instructions Python valides qui composent le corps de la fonction : les instructions doivent avoir le même niveau d'indentation (généralement 4 espaces).
- Une instruction facultative `return` pour renvoyer une valeur de la fonction.

L'objectif majeur des fonctions est de regrouper du code qui est exécuté plusieurs fois. Sans une fonction définie, il faudrait copier ce code à chaque fois.

En général, vous voulez éviter la duplication de code. En effet, si vous décidez de mettre à jour le code, par exemple si vous trouvez un bug, vous devrez le corriger partout. Au fur et à mesure que vous acquérez de l'expérience en programmation, vous vous retrouverez souvent avec du code dupliqué dont vous voudrez vous débarrasser.

La déduplication rend vos programmes plus courts, plus faciles à lire et plus faciles à mettre à jour.

## Méthode Mise en pratique

Essayez de créer vos propres fonctions en utilisant le mot-clé `def` avec des paramètres entre parenthèses.

```
1 def hello() :
2     print("Salut")
3     print("Hola")
4     print("Buongiorno")
5 hello()
6 hello()
7 hello()
```

La première ligne est une instruction `def` qui définit une fonction nommée `hello()`. Le code dans le bloc qui suit l'instruction `def` est le corps de la fonction. Ce code est exécuté lorsque la fonction est appelée, pas lorsque la fonction est définie.

Les lignes `hello()` après la fonction sont des appels de fonction. Dans le code, un appel de fonction est simplement le nom de la fonction suivi de parenthèses. Il contient éventuellement un certain nombre d'arguments entre les parenthèses.

Quand l'exécution du programme atteint ces appels de fonction, il commencera par exécuter la première ligne de la fonction. Quand il atteint la dernière, l'exécution revient à la ligne qui a appelé la fonction et continue de parcourir le code à nouveau. Comme ce programme appelle `hello()` trois fois, le code dans `hello()` est exécuté trois fois.

Lorsque vous appelez la fonction `print()` ou `len()`, vous passez des valeurs, appelées arguments, dans ce contexte, en les tapant entre parenthèses. Vous pouvez également définir vos propres fonctions qui acceptent des arguments.

```
1 def hello(name):
2     print("Hello " + name)
3 hello("Alice")
4 hello("Bob")
5 #print(name)
6
```

La définition de la fonction `hello()` dans ce programme a un paramètre appelé `name`. Un paramètre est une variable dans laquelle un argument est stocké lorsqu'une fonction est appelée. La première fois que la fonction `hello()` est appelée, c'est avec l'argument `Alice`. L'exécution du programme entre dans la fonction et le prénom contenu dans la variable est automatiquement défini sur « *Alice* », qui est ensuite imprimé par la fonction `print()`.

## Remarque

Il faut noter que la valeur stockée dans les paramètres est oubliée lorsque la fonction est réexécutée. Si vous essayez d'imprimer la variable `name`, vous obtiendrez une `NameError` puisqu'elle n'existe pas.

Elle a été détruite après le retour de l'appel de fonction `hello('Bob')`, donc utiliser une fonction `print()` ferait référence à une variable `name` qui n'existe pas.

Cela ressemble à la façon dont les variables d'un programme sont oubliées lorsque le programme se termine. Nous approfondirons cette notion dans la portée locale d'une fonction.

## Fondamental Structurer un programme avec les fonctions

## B. Valeurs de retour

Dès l'obtention d'un résultat, au lieu d'utiliser `break`, le résultat d'une fonction peut être retourné immédiatement. C'est un autre moyen de terminer une itération.

```
1 import random
2 def getAnswer(answerNumber):
3     if answerNumber == 1:
4         return "C'est certain"
5     elif answerNumber == 2:
6         return "Qu'il en soit ainsi"
7     elif answerNumber == 3:
8         return 'Oui'
9     elif answerNumber == 4:
10        return 'Réponse hasardeuse, essaye encore'
11    elif answerNumber == 5:
12        return 'Réessaye plus tard'
13    elif answerNumber == 6:
14        return 'Concentre-toi et redemande'
15    elif answerNumber == 7:
16        return 'Ma réponse est non'
17    elif answerNumber == 8:
18        return 'Les perspectives sont mauvaises'
19    elif answerNumber == 9:
20        return "J'en doute"
21
22 r = random.randint(1,9)
23 fortune = getAnswer(r)
24 print(fortune)
```

Le programme suivant définit une fonction qui renvoie une chaîne de caractères différents, selon le nombre passé en argument. Lorsque ce programme démarre, Python importe d'abord le module `random`. Ensuite, la fonction `getAnswer()` est définie. Comme la fonction est définie (et non appelée), l'exécution saute le code qu'il contient.

La fonction `random.randint()` est alors appelée avec les deux arguments 1 et 9. Il évalue un entier aléatoire compris entre 1 et 9 (dont 1 et 9 eux-mêmes), et cette valeur est stockée dans une variable nommée `r`.

La fonction `getAnswer()` est appelée avec `r` comme argument. L'exécution du programme se déplace vers le haut de la fonction `getAnswer()` et la valeur `r` est stockée dans un paramètre nommé `answerNumber`.

### Rappel

En général, la valeur évaluée par un appel de fonction est appelée *valeur return* de la fonction. Lors de la création d'une fonction à l'aide de l'instruction `def`, vous pouvez spécifier la valeur de retour avec une déclaration `return`.

## Exercice : Quiz

[solution n°1 p.13]

### Question 1

Quel mot-clé sert à définir une fonction ?

- ☐ def
- ☐ function
- ☐ array

### Question 2

Comment appelle-t-on l'endroit où l'on écrit les lignes de code qui définissent la fonction ?

- ☐ Le corps de la fonction
- ☐ Le header de la fonction
- ☐ Le footer de la fonction

Question 3

Les valeurs que l'on passe à une fonction s'appellent des paramètres.

- ☐ Vrai
- ☐ Faux

Question 4

À quoi sert le mot-clé `return` ?

- ☐ À terminer la fonction et à retourner la valeur
- ☐ À retourner à la ligne
- ☐ À sauter une ligne lors de l'impression avec `print()`

Question 5

Lors de la définition de la fonction, on peut indiquer son type de retour.

- ☐ Vrai
- ☐ Faux

### III. Fonctions natives et les fonctions lambda

#### A. Fonctions natives

Python fournit une grande quantité de fonctions natives, comme la plupart des langages de programmation modernes. Ces fonctions ont des actions diverses et variées qui vont de la simple impression sur la sortie standard, jusqu'au tri dans les tableaux ou la création de collections. Pour commencer, nous allons voir les fonctions natives les plus utiles sur les types communs.

#### La fonction `min()`

La fonction `min()` est utilisée dans les tableaux ou les itérables Python. On lui passe un argument qui peut être une liste d'entiers (Integer) séparés d'une virgule ou un tableau. Cette fonction se charge de retourner la plus petite valeur de la suite d'arguments ou du tableau.

##### Exemple

Voici un exemple d'utilisation de cette fonction.

```
1 min(1, 5, 3, 9, 2)
2 1
```

On voit que la fonction retourne bien la valeur minimale de la suite d'arguments.

### La fonction `pow()`

La fonction `pow()` sert à calculer la valeur d'une puissance. Elle prend deux arguments **x** et **y**, où **x** est le nombre à évaluer et **y** sa puissance. La fonction peut également prendre un troisième argument optionnel qui se chargera de calculer le modulo.

#### Remarque

Voici un exemple simple de calcul d'une puissance de 10.

```
1 pow(10, 3)
2 1000
```

On sait que  $10^3$  donne 1 000 et c'est bien le résultat que l'on obtient avec la fonction `pow()`.

### La fonction `range()`

La fonction `range()` est très utile lors d'une boucle, par exemple lorsqu'on veut que l'index itère sur un intervalle défini. La fonction `range()` va permettre de générer cet intervalle. Elle prend deux arguments et un troisième optionnel. Les deux premiers vont donner respectivement le début et la fin de l'intervalle ; le troisième, lui, sera chargé de donner l'écart entre chaque valeur à l'intérieur de l'intervalle.

#### Exemple

Voici un exemple d'utilisation de la fonction `range()`.

```
1 list(range(3, 10) )
2 [3, 4, 5, 6, 7, 8, 9]
3 list(range(1, 10, 2) )
4 [1, 3, 5, 7, 9]
```

On doit lancer la fonction `range()` dans une fonction nommée `list()`, pour que la valeur de retour soit correctement formatée à l'écran. On voit qu'à chaque fois, on récupère bien une liste d'entiers dans l'intervalle fournie par `range`. Pour le deuxième exemple, c'est le même résultat que pour le premier, mais on a bien un pas de 2 entre chaque valeur.

## B. Fonction Lambda

Les fonctions lambda forment un groupe de fonctionnalités spécifiques dans Python. Il ne s'agit pas vraiment d'une fonction, mais plutôt d'une expression. En effet, il est possible de déclarer une fonction et sa logique en une seule et unique ligne de code.

#### Méthode Déclarer une fonction lambda

Pour déclarer une fonction lambda, il n'est pas nécessaire d'utiliser le mot-clé `def`. En effet, il suffit de déclarer une variable portant le nom que vous voulez à la suite de l'opérateur. Autrement dit, il suffit de noter le mot-clé `lambda`, puis de saisir les paramètres séparés d'une virgule. Enfin, il faut noter l'opérateur, à la suite duquel nous déclarons la logique de notre fonction qui doit tenir sur une ligne.

Voici comment on déclare une fonction lambda.

```
1 ma_fonction = lambda a,b : a+b
```

Ici, la fonction additionne  $a + b$ , on lui passe les paramètres juste après le mot-clé `lambda`. Vous constaterez l'absence du mot-clé `return`. En effet, celui-ci est imbriqué dans les fonctions lambda.

Pour appeler la fonction, il suffit de le faire grâce à la variable dans laquelle est déclarée notre fonction lambda.

```
1 >>>ma_fonction = lambda a,b : a+b
2 >>>ma_fonction(3, 3)
3 6
```

Les fonctions lambda permettent de gagner des lignes de code. On les utilise pour remplacer les petites fonctions. Voyons des exemples concrets d'utilisation de la fonction lambda dans des situations pertinentes.

#### Exemple Utilisation d'une fonction lambda pour gagner en nombre de lignes dans un algorithme simple

Ici, le but est de filtrer les valeurs d'un tableau pour ne récupérer que les valeurs supérieures à 10. On utilise donc la fonction native `filter()` qui prend en argument une fonction qui définit ce que doit faire le filtre et un tableau de valeurs à filtrer. Au lieu de définir une fonction et de la passer à `filter()`, nous allons utiliser une fonction lambda.

```
1 def my_custum_filter(x):
2     return x >10
3 some_numbers_list= [1,2,3,87,40,49,303,34,32,98]
4 filtered_list = list(filter(my_custom_filter, some_numbers_list))
```

Voici la fonction initiale : nous sommes obligés de définir une fonction et de la passer en argument pour gagner du temps, nous allons donc utiliser lambda.

```
1 some_numbers_list= [1,2,3,87,40,49,303,34,32,98]
2 filtered_list = list(filter(lambda x: x > 10, some_numbers_list))
```

On voit qu'en utilisant lambda, on peut tout écrire sur une seule ligne. On pourrait comparer lambda à la fonctionnalité qui porte le même nom sur Java, ou encore aux anonymous functions de JavaScript.

## C. Compléments

### Portée et durée de vie des variables

La portée d'une variable fait référence au domaine d'un programme, où qu'il soit déclaré. Les arguments et les variables d'une fonction ne sont pas accessibles, en dehors de la fonction définie. Par conséquent, ils n'ont qu'un domaine local.

La période d'existence d'une variable dans la RAM est appelée durée de vie. Les variables d'une fonction ont la même durée de vie que la fonction elle-même.

Lorsque nous sortons de la fonction, les variables sont supprimées. Par conséquent, une fonction ne conserve pas la valeur d'une variable des exécutions précédentes.

#### Exemple

Voici un exemple simple de la portée d'une variable dans une fonction.

```
1 #defining a function to print a number.
2 def number( ):
3     num = 30
4     print( "Valeur de la variable num: ", num)
5
6 num = 20
7 number()
8 print( "Valeur de la variable num:", num)
9
10 Sortie:
11 >>>Valeur de la variable num: 30
12 >>>Valeur de la variable num: 20
```



Ici, nous pouvons observer que la valeur initiale de `num` est de 20. Même si la fonction `number()` a modifié la valeur de `num` à 30, la valeur de `num` en dehors de la fonction est restée inchangée.

En effet, la variable `num` dans la fonction est distincte de la variable extérieure à la fonction (locale à la fonction). Malgré leur nom de variable identique, ce sont deux variables distinctes ayant des portées distinctes.

Les variables au-delà de la fonction, au contraire, sont accessibles à l'intérieur de la fonction. Ces variables ont une portée mondiale.

Nous pouvons récupérer leurs valeurs à l'intérieur de la fonction, mais nous ne pouvons pas les modifier. Si nous déclarons une variable globale à l'aide du mot-clé `global`, nous pouvons également modifier la valeur de la variable en dehors de la fonction.

### Fonction Python dans une autre fonction

Les fonctions sont considérées comme des objets de première classe dans Python. Dans un langage de programmation, les objets de première classe sont traités de la même façon, partout où ils sont utilisés. Ils peuvent être utilisés dans des expressions conditionnelles, comme arguments, et enregistrés dans des structures de données intégrées. Si un langage de programmation gère des fonctions comme des entités de première classe, on dit qu'il implémente des fonctions de première classe. Python prend en charge la notion de fonctions de première classe.

La fonction interne ou imbriquée fait référence à une fonction définie dans une autre fonction définie. Les fonctions internes peuvent accéder aux paramètres de la portée externe. Les fonctions internes sont construites pour les couvrir des changements qui se produisent en dehors de la fonction. De nombreux développeurs considèrent ce processus comme une encapsulation.

```
1 def function1():
2     string = 'Python functions tutorial'
3
4     def function2():
5         print( string )
6
7     function2()
8 function1()
```

### Complément La notion de récursivité

## Exercice : Quiz

[solution n°2 p.14]

### Question 1

Une fonction lambda peut être considérée comme une expression.

- ☐ Vrai
- ☐ Faux

### Question 2

Une fonction lambda peut avoir plusieurs arguments.

- ☐ Vrai
- ☐ Faux

### Question 3

Une fonction lambda nécessite un mot-clé `return`.

- ☐ Vrai
- ☐ Faux

#### Question 4

De quoi a besoin une fonction lambda pour être valide ?

- ☐ Du mot-clé `lambda`
- ☐ D'un ou plusieurs paramètres
- ☐ Du mot-clé `return`

#### Question 5

On ne peut passer des chaînes de caractères en tant que paramètres, à une fonction `lambda`.

- ☐ Vrai
- ☐ Faux

## V. Essentiel

Python est donc un langage complet. Il dispose de fonctions natives très utiles qui lui permettent un champ d'action divers et varié.

Définir une fonction en Python est quelque chose de relativement simple à l'aide du mot-clé `déf.` De plus, notre fonction peut utiliser le mot-clé `return` pour retourner une valeur, et éventuellement, pour pouvoir stocker cette valeur dans une variable par la suite. Pour finir, le fonctionnement des fonctions `lambda` permet de gagner en lisibilité dans le code, ainsi qu'en performance.

Cela donne une vision globale des fonctions sur Python vous êtes prêt à exploiter.

## VI. Auto-évaluation

### A. Exercice

Vous devez développer un programme permettant de calculer le coût d'un voyage.

#### Question 1

[solution n°3 p.15]

Comment développer une fonction qui va calculer le coût total de votre séjour en hôtel ?

#### Question 2

[solution n°4 p.15]

Maintenant que votre fonction a été écrite, mettez-la sous la forme d'une fonction `lambda`.

### B. Test

#### Exercice 1 : Quiz

[solution n°5 p.15]

#### Question 1

Pourquoi les fonctions sont-elles avantageuses dans vos programmes ?

- ☐ Les fonctions réduisent le code doublon.
- ☐ Les fonctions rendent le code plus facile à lire et à mettre à jour.
- ☐ Les fonctions permettent de vérifier des conditions.

#### Question 2

Quand le code d'une fonction s'exécute-t-il ?

- ☐ Lorsque la fonction est appelée.
- ☐ Lorsque la fonction est définie.

#### Question 3

Une fonction se compose de l'instruction `def` et du code dans sa clause. Un appel de fonction est ce qui déplace l'exécution du programme dans la fonction.

- ☐ Vrai
- ☐ Faux

#### Question 4

Parmi ces propositions, lesquelles sont vraies pour la fonction Python ?

- ☐ Une fonction Python ne peut renvoyer qu'une seule valeur.
- ☐ Une fonction Python peut accepter un nombre illimité d'arguments.
- ☐ Une fonction Python peut renvoyer plusieurs valeurs.

#### Question 5

Une fonction lambda est toujours la meilleure solution pour simplifier son code.


- ☐ Vrai
- ☐ Faux

## Solutions des exercices




**Exercice p. 5 Solution n°1****Question 1**

Quel mot-clé sert à définir une fonction ?

- ☒ def
- ☐ function
- ☐ array
-  C'est le mot-clé `def` qui sert à définir une fonction.


**Question 2**

Comment appelle-t-on l'endroit où l'on écrit les lignes de code qui définissent la fonction ?

- ☒ Le corps de la fonction
- ☐ Le header de la fonction
- ☐ Le footer de la fonction
-  Le corps de la fonction est l'endroit où l'on écrit les différentes lignes qui définissent la fonction.


**Question 3**

Les valeurs que l'on passe à une fonction s'appellent des paramètres.

- ☐ Vrai
- ☒ Faux
-  Les valeurs passées à une fonction s'appellent des arguments.

**Question 4**


À quoi sert le mot-clé `return` ?

- ☒ À terminer la fonction et à retourner la valeur
- ☐ À retourner à la ligne
- ☐ À sauter une ligne lors de l'impression avec `print()`
-  Le mot-clé `return` sert à terminer la fonction et à retourner sa valeur.

**Question 5**

Lors de la définition de la fonction, on peut indiquer son type de retour.

- ☒ Vrai
- ☐ Faux

-  On peut indiquer le type de retour avec le mot-clé `return`, directement au moment de la déclaration de la fonction.


### Exercice p. 9 Solution n°2

#### Question 1

Une fonction lambda peut être considérée comme une expression.

☒ Vrai

☐ Faux


 Elle peut être considérée comme telle car elle est déclarée dans une expression.

#### Question 2

Une fonction lambda peut avoir plusieurs arguments.

☒ Vrai

☐ Faux


 C'est en effet le cas, il suffit de les séparer d'une virgule.

#### Question 3

Une fonction lambda nécessite un mot-clé `return`.

☐ Vrai

☒ Faux

 Le mot-clé `return` est compris dans la fonction lambda.


#### Question 4

De quoi a besoin une fonction lambda pour être valide ?

☒ Du mot-clé `lambda`

☒ D'un ou plusieurs paramètres

☐ Du mot-clé `return`


 Pour être valide, une fonction lambda a besoin du mot-clé `lambda` et d'un ou plusieurs paramètres.

#### Question 5

On ne peut passer des chaînes de caractères en tant que paramètres, à une fonction `lambda`.

☐ Vrai

☒ Faux

 Il est possible de passer de toutes sortes de paramètres à une fonction lambda.

**p. 10 Solution n°3**

Vous allez devoir définir une fonction grâce au mot-clé `def`. Donnez-lui un nom qui correspond à son utilité. Cette fonction devra avoir deux arguments : un pour le prix d'une nuit et un pour le nombre de nuits. La fonction devra donc faire la multiplication et retourner le prix total.

```
1 >>> def hotel_price (price, night)
2     return price * night
```

**p. 10 Solution n°4**


Pour cela, c'est très simple, il suffit de reprendre la fonction précédente et de l'adapter à la syntaxe lambda comme suit :

```
1 >>> hotel_price = lambda price, night : price*night
```

La fonction est maintenant écrite sur une ligne et fonctionne très bien.


**Exercice p. 10 Solution n°5****Question 1**

Pourquoi les fonctions sont-elles avantageuses dans vos programmes ?

- ☒ Les fonctions réduisent le code doublon.
- ☒ Les fonctions rendent le code plus facile à lire et à mettre à jour.
- ☐ Les fonctions permettent de vérifier des conditions.
-  Les fonctions permettent de bien séparer le code, de faciliter sa lecture et aussi d'améliorer ses performances si elles sont bien écrites.


**Question 2**

Quand le code d'une fonction s'exécute-t-il ?

- ☒ Lorsque la fonction est appelée.
- ☐ Lorsque la fonction est définie.
-  Le code s'exécute lorsque la fonction est appelée.


**Question 3**

Une fonction se compose de l'instruction `def` et du code dans sa clause. Un appel de fonction est ce qui déplace l'exécution du programme dans la fonction.

- ☒ Vrai
- ☐ Faux
-  L'appel de fonction est le moment où la fonction est appelée ou exécutée.

**Question 4**


Parmi ces propositions, lesquelles sont vraies pour la fonction Python ?

- ☐ Une fonction Python ne peut renvoyer qu'une seule valeur.
- ☒ Une fonction Python peut accepter un nombre illimité d'arguments.
- ☒ Une fonction Python peut renvoyer plusieurs valeurs.
-  Une fonction Python accepte plusieurs arguments et peut aussi retourner plusieurs valeurs.

### Question 5

---

Une fonction lambda est toujours la meilleure solution pour simplifier son code.

- ☐ Vrai
- ☒ Faux
-  Certaines fois, il faut faire attention : la fonction lambda, dans sa syntaxe, peut complexifier le code et le rendre incompréhensible en fonction du contexte.