

LES FONCTIONS



01

DÉFINITION

Qu'est ce qu'une fonction?

02

APPELER UNE FONCTION

Pourquoi et comment appeler une fonction?

03

PASSAGE PAR RÉFÉRENCE

Qu'est-ce qu'implique les références?

04

TYPES D'ARGUMENTS

Quels sont les différents types d'arguments de fonction?

05


ARGUMENTS REQUIS

Qu'est ce que les arguments requis?

06

ARGUMENTS PAR MOTS-CLÉS

Qu'est-ce que les arguments par mots-clés?





07

ARGUMENTS PAR DÉFAUT

Qu'est ce qu'un argument
par défaut?

09

FONCTIONS ANONYMES

Qu'est-ce qu'une
fonction anonyme?

11

SCOPE DE VARIABLES

Qu'est ce qu'une
scope de variables?

ARGUMENTS À LONGUEUR VARIABLE

Qu'est ce qu'un argument
à longueur variable?

08

RETURN

Qu'est-ce qu'une
déclaration de retour?

10

VARIABLES GLOBALES & LOCALES

Les différences
entre variables
globales et locales ?

12

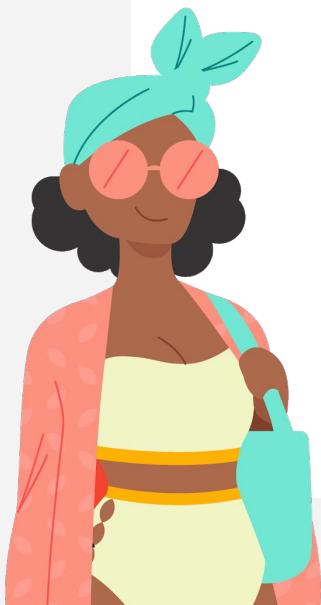




DÉFINITION

Une fonction est un bloc de code organisé et réutilisable qui est utilisé pour effectuer une action. Les fonctions offrent une meilleure modularité à votre programme. Cela optimisera votre code et vous évitera d'écrire plusieurs fois les mêmes blocs de code.

Comme nous l'avons vu dans les chapitres précédents, Python vous offre de nombreuses fonctions, mais vous pouvez également créer vos propres fonctions. Cela va vous permettre de définir vos propres actions pour personnaliser votre programme afin de répondre à des besoins bien précis.



DÉFINITION

Voici des règles simples pour définir une fonction en Python :

- Les blocs de fonction commencent par le mot-clé **def** suivi du nom de la fonction et des parenthèses "**()**" suivi de deux points "**:**".
- Tout **paramètre** ou **argument** d'entrée doit être placé **entre ces parenthèses**. Vous pouvez également **définir des paramètres** à l'intérieur de ces parenthèses.
- La première instruction d'une fonction peut être une instruction optionnelle : la **documentation de la fonction** ou **docstring**).
- Le **bloc de code** de chaque fonction est indenté.
- L'instruction **return <expression>** permet de quitter une fonction, et de renvoyer une expression à l'appelant si nécessaire.
Une instruction **return** sans argument est identique à **return None**.

```
def function_name( parameters ):
    """function_docstring"""
    function_code
    return expression
```

```
def print_me( my_str ):
    """Print the given string"""
    print(my_str)
    return
```





APPELER UNE FONCTION

Une fonction est définie par son **nom**, et les **paramètres** qui doivent être inclus dans celle-ci.

Une fois que la structure de base d'une fonction est finalisée, vous pouvez l'exécuter en l'appelant depuis une autre fonction ou directement depuis l'invite Python dans votre terminal.

```
def print_me( my_str ):  
    """Print the given string"""  
    print(my_str)  
    return  
  
print_me("I love apples and pineapples!")  
print_me("I hate spinach and cucumbers!")  
  
# Resultats :  
  
I love apples and pineapples!  
I hate spinach and cucumbers!
```



PASSAGE PAR RÉFÉRENCE

Tous les paramètres ou arguments du langage Python sont passés par référence. Cela signifie que si vous changez ce à quoi un paramètre fait référence dans une fonction, le changement se répercute dans la fonction appelante.

```
def change_me( my_list ):  
    my_list.append([1,2,3,4])  
    print("Values inside the function: ", my_list)  
    return  
  
my_list = [10, 20, 30]  
change_me( my_list )  
print("Values outside the function: ", my_list)
```

Resultats :

Values inside the function: [10, 20, 30, [1, 2, 3, 4]]
Values outside the function: [10, 20, 30, [1, 2, 3, 4]]

```
def change_me( my_list ):  
    my_list = [1, 2, 3, 4]  
    print("Values inside the function: ", my_list)  
    return  
  
my_list = [10, 20, 30]  
change_me( my_list )  
print("Values outside the function: ", my_list)
```

Resultats :

Values inside the function: [1, 2, 3, 4]
Values outside the function: [10, 20, 30]



TYPES D'ARGUMENTS

REQUIS

Les arguments requis sont les arguments transmis à une fonction dans un ordre de position correct.

PAR DÉFAUT

Un argument par défaut est un argument qui prend une valeur par défaut si une valeur n'est pas fournie dans l'appel de fonction pour cet argument.

PAR MOTS-CLÉS

Les arguments par mots-clés sont liés aux appels de fonction.

À LONGUEUR VARIABLE

Ces arguments sont appelés arguments à longueur variable et ne sont pas nommés dans la définition de la fonction.



ARGUMENTS REQUIS

Les **arguments requis** sont les arguments transmis à une fonction dans un **ordre de position précis**. Évidemment, le nombre d'arguments dans l'appel de fonction doit **correspondre** exactement à la **définition de la fonction**.

Par exemple, pour appeler la fonction `print_me()`, vous devez absolument passer un argument, **sinon cela donne une erreur de syntaxe**.

```
def print_me( my_str ):
    """Print the given string"""
    print(my_str)
    return
```

```
print_me()
```

Resultats :

Traceback (most recent call last):

File "test.py", line 11, in <module>

print_me()

TypeError: print_me() takes exactly 1 argument
(0 given)



ARGUMENTS PAR MOTS-CLÉS

Les arguments par mots-clés sont liés aux appels de fonction. Lorsque vous utilisez des arguments de mots-clés dans un appel de fonction, l'appelant identifie les arguments par le nom du paramètre.

Cela vous permet de passer des arguments ou de les mettre en désordre car l'interpréteur Python est capable d'utiliser les mots-clés fournis pour faire correspondre les valeurs avec les paramètres.

```
def print_info( fruit, weight):  
    """Print passed info into this function"""  
    print("Fruit: ", fruit)  
    print("Weight ", weight)  
    return
```

```
print_info(weight=50, fruit="cherry")
```

Resultats :

```
Fruit: cherry  
Weight 50
```



ARGUMENTS PAR DÉFAUT

Un argument par défaut est un argument qui prend une valeur par défaut, si une valeur n'est pas fournie dans l'appel de fonction pour cet argument.

L'exemple suivant donne une idée sur les arguments par défaut.

```
def print_info( fruit, weight= 35 ):  
    """Print passed info into this function"""  
    print("Fruit: ", fruit)  
    print("Weight ", weight)  
    return
```

```
print_info(weight=50, fruit="cherry")  
print_info(fruit="grape")
```

Resultats :

```
Fruit: cherry  
Weight 50  
Fruit: grape  
Weight 35
```



ARGUMENTS À LONGUEUR VARIABLE

Il se peut que vous deviez traiter une fonction pour **plus d'arguments que ceux que vous avez spécifiés lors de la définition de la fonction**. Ces arguments sont appelés **arguments à longueur variable** et ne sont **pas nommés** dans la définition de la fonction.

Un astérisque “*” est placé devant le nom de la variable qui contient les valeurs de **tous les arguments de la variable autre qu'un mot-clé**. Ce tuple reste vide si aucun argument supplémentaire n'est spécifié lors de l'appel de la fonction.

```
def print_info( arg1, *values ):  
    """Print passed info into this function"""  
    print("Output is: ", arg1)  
    for value in values:  
        print(value )  
    return  
  
print_info(10)  
print_info(70, 60, 50)
```

Resultats :

Output is: 10

Output is: 70

60

50



FONCTIONS ANONYMES

Ces fonctions sont dites **anonymes** car elles ne sont **pas déclarées de manière standard** en utilisant le mot-clé `def`. Pour créer une **fonction anonyme** vous devez utiliser le mot-clé `lambda`.

Elles peuvent prendre un **nombre quelconque d'arguments** mais ne renvoient qu'une seule valeur sous la **forme d'une expression**.

Elles **ne peuvent pas contenir de commandes** ou d'expressions multiples.

Elles possèdent leur **propre espace de nom local** et **ne peuvent pas accéder** à des variables autres que celles de leur **liste de paramètres** et celles de l'espace de nom global.

```
sum = lambda arg1, arg2: arg1 + arg2
print("Value of total : ", sum(10, 20))
print("Value of total : ", sum(20, 20))
```

```
full_name = lambda first, last: f"Full name: {first.title()} {last.title()}"
print(full_name('joe', 'the mango'))
```

Resultats :

Value of total : 30

Value of total : 40

Full name: Joe The Mango



RETURN

La déclaration `return <expression>` permet de sortir d'une fonction, en transmettant éventuellement une expression à l'appelant.

Une déclaration `return` sans argument est identique à une déclaration `return None`.

```
def sum(arg1, arg2):  
    """Add both the parameters and return them."""  
    total = arg1 + arg2  
    print("Inside the function : ", total)  
    return total
```

```
total = sum( 10, 20 );  
print("Outside the function : ", total)
```

Resultats :

Inside the function : 30

Outside the function : 30

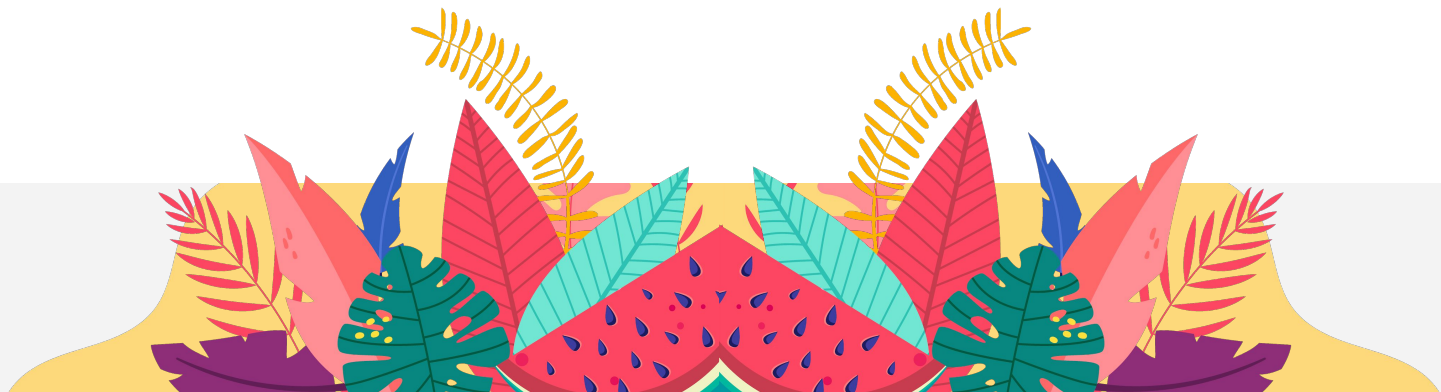
SCOPE DE VARIABLES

Toutes les variables d'un programme peuvent ne pas être accessibles à tous les endroits de ce dernier. Cela dépend de l'endroit où vous avez déclaré votre variable.

L'étendue ou la scope d'une variable détermine la partie du programme où vous pouvez accéder à un identifiant particulier. Il existe deux champs d'application de base des variables en Python :

Variables globales

Variables locales





VARIABLES GLOBALES & LOCALES

Les variables qui sont définies à l'intérieur d'une fonction ont une **portée locale**, et celles qui sont définies à l'extérieur ont une **portée globale**.

Cela signifie que les **variables locales** ne sont accessibles qu'à l'intérieur de la fonction dans laquelle elles sont déclarées, alors que les **variables globales** sont accessibles dans tout le corps du programme par toutes les fonctions.

```
total = 0 # global variable
def sum( arg1, arg2 ):
    """Add both the parameters and return them."""
    total = arg1 + arg2 # local variable
    print("Inside the function local total : ", total)
    return total
```

```
sum( 10, 20 )
print("Outside the function global total : ", total)
```

Resultats :

local variable

Inside the function local total : 30

global variable

Outside the function global total : 0

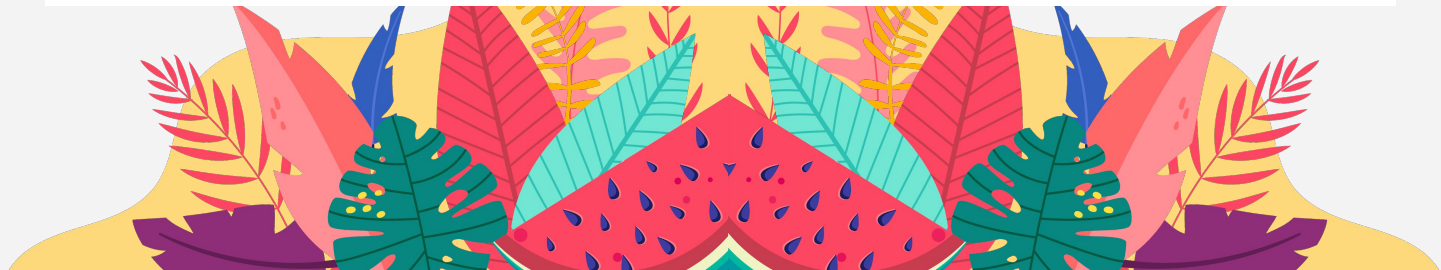
The background of the slide is a vibrant, colorful pattern of various tropical leaves and foliage. The leaves are in shades of green, yellow, orange, red, and purple, creating a dense and lively border around the central text. Some leaves are large and detailed, while others are smaller and more stylized. The overall effect is a tropical and energetic aesthetic.

EXERCICES

EXERCICES

1. Créez une fonction pour trouver le nombre le plus grand parmi 3 nombres.
2. Créez une fonction qui multiplie tous les éléments d'une liste entre eux.
3. Créez une fonction qui affiche de nombre de majuscule et minuscule dans un texte donné.
4. Créez une fonction qui prend une liste et renvoie une nouvelle liste avec des éléments uniques de la première liste.
5. Créez une fonction permettant de vérifier si un nombre est premier ou non.

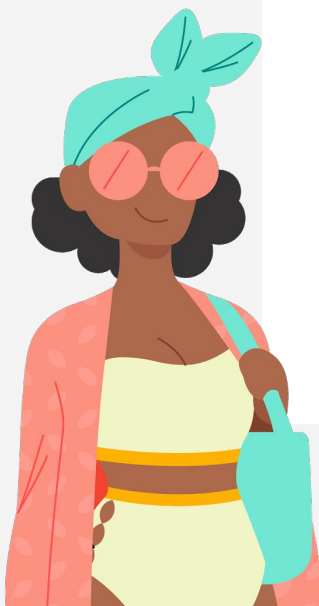
Remarque : Un nombre premier est un nombre naturel supérieur à 1 et qui n'a pas d'autres diviseurs positifs que 1 et lui-même.



EXERCICES

6. Créez une fonction qui retourne la liste des nombres premiers compris entre 1 et 100 inclus.
7. Créez une fonction qui retourne une liste de nombres pairs en fonction de la liste donnée.
8. Créez une fonction qui teste si un mot donné est un palindrome ou non.
9. Créez une fonction qui affiche le carré suivant :

```
1 2 3 4 5
2 2 3 4 5
3 3 3 4 5
4 4 4 4 5
5 5 5 5 5
```
10. Créez une fonction lambda pour afficher la liste des nombres impairs de 1 à 100.



The background of the slide is a vibrant, colorful pattern of various tropical leaves and foliage. The leaves are in shades of green, yellow, orange, red, and purple, creating a dense and lively border around the central text. Some leaves are large and detailed, while others are smaller and more stylized. The overall effect is a tropical, summery aesthetic.

SOLUTIONS

CORRECTION

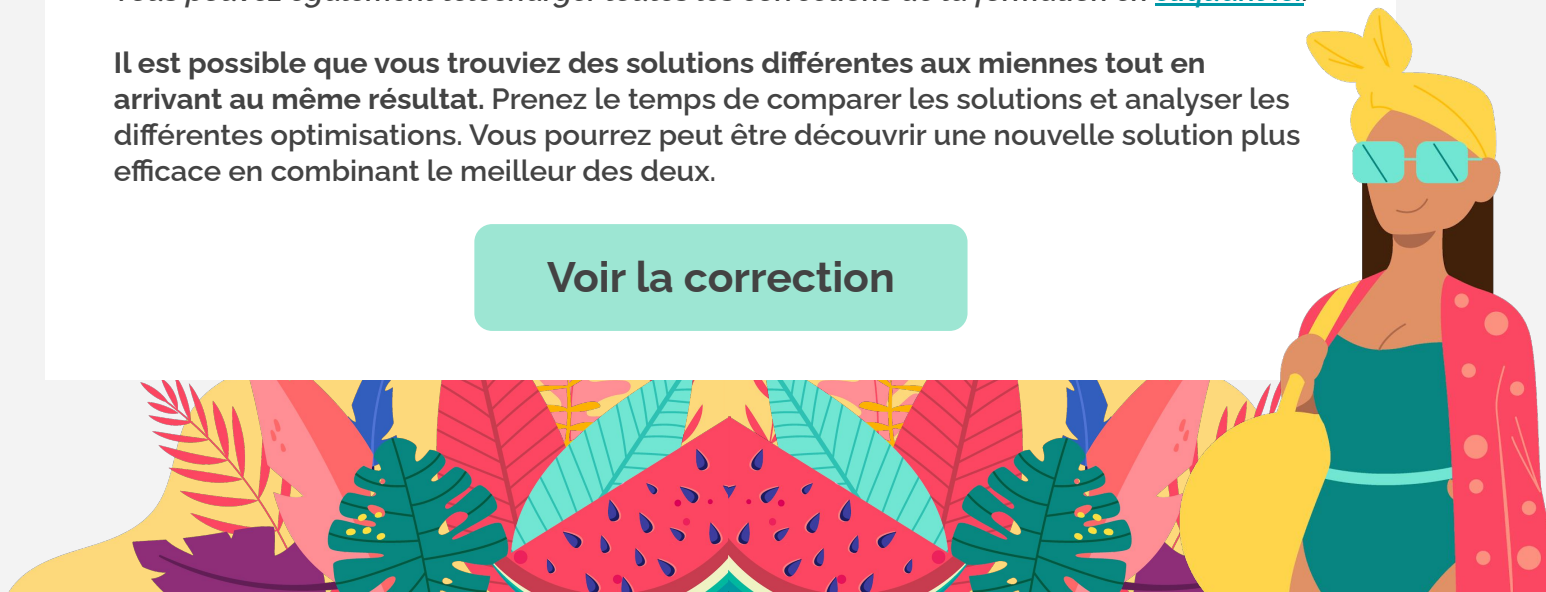
Pour visualiser la correction du chapitre cliquer sur le bouton ci-dessous.

*Le fichier de la correction s'ouvrira dans un nouvel onglet de votre navigateur préféré.
Pour cela vous devez avoir accès à une connexion Internet.*

Vous pouvez également télécharger toutes les corrections de la formation en [cliquant ici](#).

Il est possible que vous trouviez des solutions différentes aux miennes tout en arrivant au même résultat. Prenez le temps de comparer les solutions et analyser les différentes optimisations. Vous pourrez peut être découvrir une nouvelle solution plus efficace en combinant le meilleur des deux.

Voir la correction





**Félicitation vous avez terminé
le dernier chapitre sur les
fonctions avec Python!**

A decorative border of various tropical leaves in vibrant colors like red, orange, yellow, green, and purple surrounds the central white area.

CRÉDITS

- Modèle de la présentation par [Slidesgo](#)
- Icônes par [Flaticon](#)
- Images et infographies par [Freepik](#)