
Fonctions

Cours

Contenus	Capacités attendues
Constructions élémentaires	Être capable de : <ul style="list-style-type: none">▪ Définir une nouvelle fonction, avec ou sans paramètres▪ Appeler une fonction, avec ou sans paramètres

C'est quoi une fonction en programmation ?

Une fonction en programmation est un bloc de code autonome qui effectue une tâche spécifique.

Voici ses principales caractéristiques :

1. **Nom** : Chaque fonction a un nom qui permet de l'identifier et de l'appeler dans le programme.
2. **Paramètres** : Les fonctions peuvent prendre des arguments (ou paramètres) en entrée, permettant de leur passer des données pour qu'elles les traitent.
3. **Corps** : C'est le code qui définit ce que fait la fonction. Ce code est exécuté chaque fois que la fonction est appelée.
4. **Valeur de retour** : Une fonction peut renvoyer une valeur (ou plusieurs) en utilisant l'instruction `return`. Cela permet de récupérer le résultat de l'opération effectuée par la fonction.
5. **Portée** : Les variables définies à l'intérieur d'une fonction ont généralement une portée locale, ce qui signifie qu'elles ne sont pas accessibles en dehors de cette fonction.

En résumé, une fonction est un moyen de structurer le code, de le rendre réutilisable, et d'encapsuler des comportements spécifiques.

1. Définir une fonction

Une fonction associe une séquence d'instruction à un nom. Voici un exemple simple d'une fonction :

```
def fonction_calcul() :  
    return 3 * 5 + 4
```

La définition commence par le mot-clé **def** suivie par **le nom de la fonction**, puis une **paire de parenthèses** et **deux points**.

Ici nous définissons la fonction mathématique qui au nombre x associe le nombre $3 * 5 + 4$.

En Python, le mot-clé **return** permet d'indiquer le résultat de la fonction.

Il est suivi d'une expression et indique que la valeur de cette expression est **renvoyée** comme résultat de la fonction.

Le corps de la fonction est écrit en retrait comme on le ferait pour un bloc à l'intérieur d'une instruction **if** ou **for**. Pour exécuter les instructions de la fonction, on appelle la fonction avec la syntaxe suivante :

```
1  
2 def fonction_calcul() :    Définition de la fonction  
3     return 3 * 5 + 4  
4  
5 resultat = fonction_calcul()    Appel de la fonction  
6  
7 print(resultat)  
8  
9
```

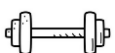
Console ×

```
>>> %Run -c $EDITOR_CONTENT
```

```
19
```

Affichage du résultat de la fonction

```
>>>
```

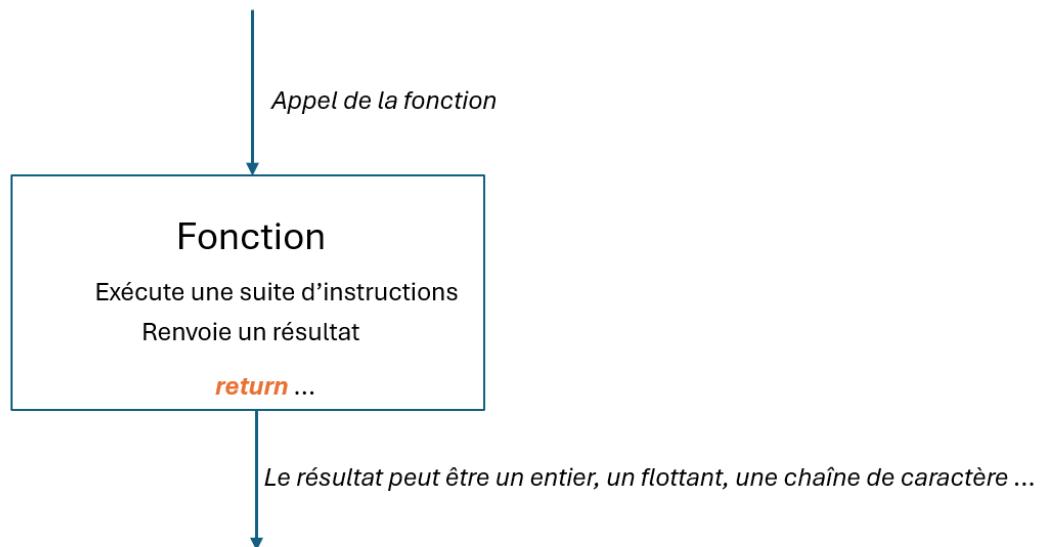


Activité : Exécuter le code ci-dessus avec Thonny.

Une fonction peut donc être vue comme une boîte, exécutant du code.

Elle peut être appelée à tout moment.

Nous pouvons donc **factoriser du code** afin d'éviter de répéter à plusieurs endroits les mêmes instructions (mêmes buts).



Erreurs

Comme dans les chapitres précédents avec for et if c'est l'indentation qui définit le bloc d'instruction formant le corps d'une fonction. Un retrait inattendu comme ci-dessous :

```
1 def truc():
2     valeur_1 = 2
3     valeur_2 = 5
4     total = 0
5     total = valeur_1 + (valeur_1 * valeur_2)
6     print("Total:", total)
7
```

Console ×

```
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<string>", line 6, in <module>
NameError: name 'total' is not defined
>>>
```

...produit une erreur immédiate, tandis qu'un retour au niveau de base marque la fin de la définition de la fonction.

Dans le cas précédent, il était plutôt attendu le code suivant :

```
1 def truc():
2     valeur_1 = 2
3     valeur_2 = 5
4     total = 0
5     total = valeur_1 + (valeur_1 * valeur_2)
6     print("Total:", total)
7
8 print("Démarrage du programme...")
9 truc() # Appel de la fonction truc()
10
```

Console ×

```
>>> %Run -c $EDITOR_CONTENT
Démarrage du programme...
Total: 12
>>>
```

2. Fonction avec paramètre

Dans notre fonction précédente, les valeurs `valeur_1` et `valeur_2` sont fixées respectivement à 2 et 5. Pour effectuer le calcul avec des valeurs quelconques on peut ajouter des paramètres à notre fonction :

```
1 def truc(param_1, param_2):
2     valeur_1 = param_1
3     valeur_2 = param_2
4     total = 0
5     total = valeur_1 + (valeur_1 * valeur_2)
6     print("Total:", total)
7
8 print("Démarrage du programme...")
9 truc(2, 5) # Appel de la fonction truc() avec les paramètres 2 et 5
```

Les paramètres sont désignés par : `param_1` et `param_2`, ajoutés entre les parenthèses. Ils sont ensuite utilisés dans le corps de la fonction à la manière d'une variable. Chaque paramètre désigne une valeur qui n'est pas connue à l'avance et qui pourra être différente à chaque appel de la fonction.

Une version totalement équivalente pourrait être :

```
1 def truc(param_1, param_2):
2     total = 0
3     total = param_1 + (param_1 * param_2)
4     print("Total:", total)
5
6 print("Démarrage du programme...")
7 truc(2, 5) # Appel de la fonction truc() avec les paramètres 2 et 5
```

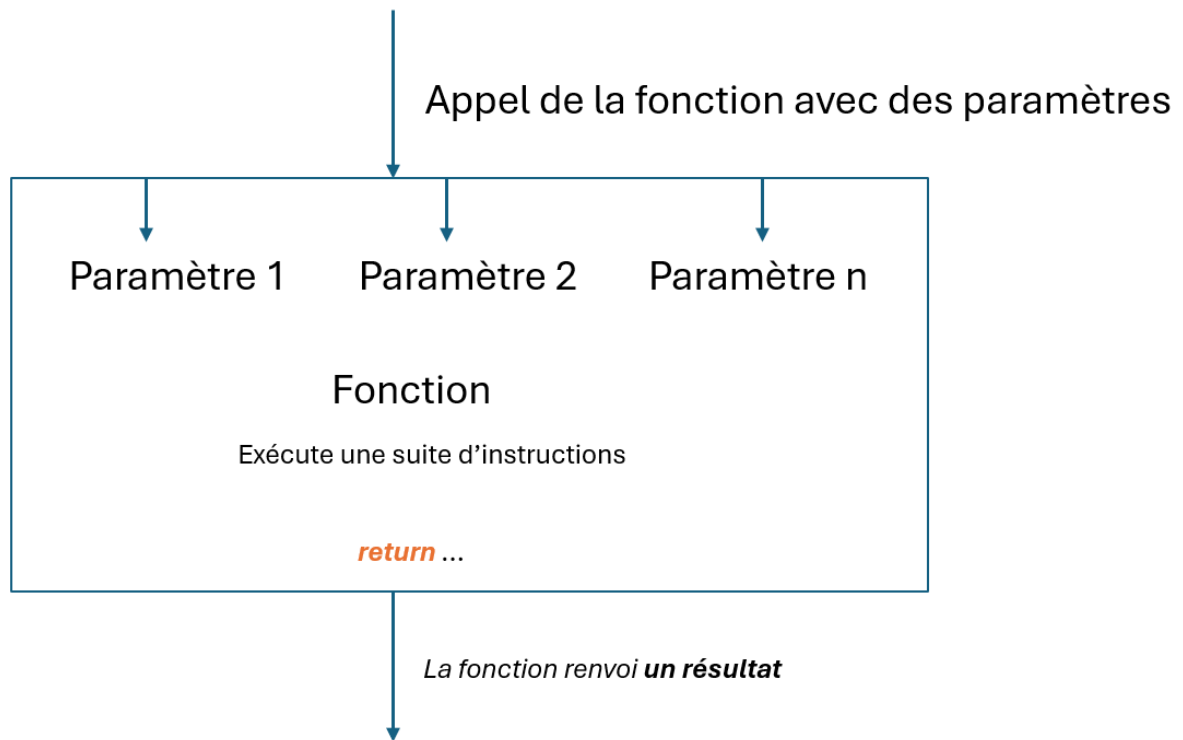
Console ×

```
>>> %Run -c $EDITOR_CONTENT
```

```
Démarrage du programme...
```

```
Total: 12
```

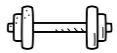
```
>>>
```



3. Une fonction sans le mot clé return

Il est tout à fait possible de définir une fonction sans le mot clé return.

```
1
2 def fonction_sans_return() :
3     x = 2
4     y = 22
5
6
7 resultat = fonction_sans_return()
8
9 print(resultat)
```



Activité : Exécuter le code ci-dessus avec Thonny. Que constatez-vous ?

None est défini comme la représentation de l'absence de valeur.

None est utilisé pour signifier l'achèvement d'une fonction **sans retour**.

Vous remarquerez que même si nous n'utilisons pas l'instruction **return**, la fonction retourne par défaut la valeur **None**.

