COURS

Fonctions

Définir une fonction en Python, lire sa signature et l'appeler avec des arguments.

I. Définitions

> **Définition - Fonction**
>
> **Séquence d'instructions réutilisable associée à un nom.**
>
> Une fonction est une séquence d'instructions réutilisable. Elle associe une séquence d'instructions à un nom.

Nous la distinguons des variables grâce aux parenthèses qui suivent son nom lors d'un appel. Exemple : `ma_fonction()`.

Nous avons déjà rencontré la fonction `type()` dans le chapitre sur les types. Cette fonction permet d'obtenir le type de la valeur passée en paramètre, c'est-à-dire entre parenthèses.

Une fonction peut être décrite à l'aide de sa signature.

> **Définition - Signature**
>
> **Nom de la fonction, noms et types des paramètres, et type du résultat renvoyé.**
>
> La signature d'une fonction est composée du nom de la fonction, du nom et du type des paramètres et du type de la valeur du résultat renvoyé par la fonction.

Exemple :

Soit la fonction `somme()` permettant de faire la somme de deux entiers. Sa signature peut s'écrire de la façon suivante :

PYTHON
```
somme(a: int, b: int) -> int
```

- Le nom de la fonction est `somme`.
- Les paramètres sont `a` et `b`, et leurs valeurs sont de type `int`.
- La valeur renvoyée par cette fonction est de type `int`.

> **Définition - Paramètre**
>
> **Variable qui peut être utilisée à l'intérieur d'une fonction.**
>
> Un paramètre est une variable qui pourra être utilisée à l'intérieur de la fonction.

Note

Une fonction peut avoir un ou plusieurs paramètres.

- ☐ Je sais expliquer ce qu'est une fonction et lire sa signature.

II. Écriture d'une fonction en Python

Une fonction en Python est composée de sa signature et d'un corps.

Pour définir une fonction, nous utilisons le mot-clé `def` suivi de sa signature et de deux-points `:`.

> **Attention**
>
> Le corps d'une fonction est la séquence d'instructions à l'intérieur de la fonction. Il est indenté, c'est-à-dire décalé vers la droite, souvent grâce à la touche de tabulation.

On dit qu'une fonction **renvoie** un résultat. Le mot-clé `return` permet de renvoyer ce résultat.

> **Exemple**

```python
def somme(a: int, b: int) -> int:
    resultat = a + b
    return resultat
```

- ☐ Je sais définir une fonction avec def, indenter son corps et renvoyer un résultat avec return.

III. Type d'une fonction

Le type d'une fonction est `function`.

> **Exemple**

```python
>>> type(somme)
<class 'function'>
```

Cependant, lorsque l'on applique `type()` à un appel de fonction avec ses arguments, on obtient le type du résultat renvoyé par la fonction.

> **Exemple**
>
> PYTHON
> ```
> >>> type(somme(3, 2))
> <class 'int'>
> ```

- [ ] Je sais distinguer le type d'une fonction du type de son résultat.

IV. Appels de fonction

A. Appeler une fonction

Un appel de fonction revient à utiliser le code de cette fonction avec des arguments.

> **Définition - Argument**
>
> **Valeur prise par un paramètre lors d'un appel de fonction.**
>
> Un argument est la valeur que prend un paramètre lors d'un appel de fonction.

> **Attention**
>
> Paramètre et argument ne désignent pas la même chose.

Pour appeler une fonction :

1. Exécuter votre programme.
2. Écrire le nom de votre fonction avec les arguments souhaités entre les parenthèses.

PYTHON
```
>>> somme(10, 5)
15
```

Les arguments sont `10` et `5` :

- le paramètre `a` prend la valeur `10` ;
- le paramètre `b` prend la valeur `5`.

Le type de l'argument doit être **le même** que celui du paramètre.

- [ ] Je sais appeler une fonction et distinguer ses paramètres de ses arguments.

B. Utiliser le résultat d'une fonction

On peut stocker le résultat dans une variable.

Exemple

```
PYTHON
>>> a = somme(10, 5)
>>> a
15
```

On peut aussi réutiliser le résultat de la fonction dans une opération.

Exemple

```
PYTHON
>>> 2 * somme(10, 5)
30
```

- ☐ Je sais stocker le résultat d'une fonction dans une variable et le réutiliser dans une opération.

Applications

Application 1

Donner, pour chaque signature suivante, le nom de la fonction, son ou ses paramètres ainsi que leur type et le type de la valeur du résultat.

```
PYTHON
produit(a: int, b: int) -> int
est_superieur(a: int, b: int) -> int
pythagore(a: float, b: float, c: float) -> float
nombre_caracteres(mot: str) -> int
```

- ☐ J'ai terminé l'application 1.

Application 2

Écrire, en Python, la fonction `produit(a: int, b: int) -> int` qui prend en paramètres deux entiers `a` et `b` et renvoie comme résultat `a × b`.

- [ ] J'ai terminé l'application 2.

Application 3

Écrire, en Python, la fonction `carre(a: int) -> int` qui prend en paramètre un entier `a` et renvoie comme résultat a^2.

- [ ] J'ai terminé l'application 3.

Application 4

Vérifier les réponses aux applications 2 et 3 en appelant les fonctions dans la console Python.

- [ ] J'ai terminé l'application 4.

Application 5

Sans utiliser l'ordinateur, proposer le résultat de chacune des suites d'instructions suivantes.

PYTHON
```
>>> type(produit(5, 5))
...
```

PYTHON
```
>>> produit(5, 5)
...
```

PYTHON
```
>>> a = somme(4, 2)
>>> b = produit(7, 8)
>>> c = a + b
>>> c
...
```

PYTHON
```
>>> produit(2, produit(2, 2))
...
```

- [ ] J'ai terminé l'application 5.

À retenir

> **À retenir**
>
> Une fonction associe un nom à une séquence d'instructions réutilisable. Sa signature précise son nom, ses paramètres et le type du résultat. En Python, on utilise `def` pour la définir et `return` pour renvoyer un résultat. Lors d'un appel, les arguments donnent des valeurs aux paramètres.