
MPSI - ITC

Cours 3 - Fonctions

I Utiliser des fonctions

- 1 - Fonctions déjà vues** On a déjà utilisé certaines fonctions Python, comme par exemple :
- La fonction `len` qui prend 1 argument (liste ou chaîne de caractères) et renvoie la longueur de son argument.
 - La fonction `print` qui prend 1 ou plusieurs arguments et les affiche
 - La fonction `int` qui prend 1 argument et le transforme en entier

- 2 - Syntaxe pour l'appel de fonctions** Pour appeler une fonction, la syntaxe est

`nom_de_la_fonction(argument1, argument2, ...)`

Par exemple, `len([0, 5, 7])` est un appel à la fonction `len` (cet appel s'évalue vers 3).

- 3 - Modules Python** Pour nous, un **module Python** est un ensemble de fonctions mises à notre disposition. Par exemple :

- Le module `math` propose des fonctions mathématiques, dont celles du tableau ci-dessous

Nom de la fonction	Argument-s	Résultat
<code>sqrt</code>	Un nombre flottant	La racine carrée de ce nombre flottant
<code>comb</code>	Deux entiers n et k	Le coefficient binomial « k parmi n »

- Le module `cmath` propose des fonctions sur les nombres complexes, dont celles du tableau ci-dessous

Nom de la fonction	Argument-s	Résultat
<code>isclose</code>	Deux nombres complexes	Un booléen indiquant s'ils sont proches
<code>sqrt</code>	Un nombre complexe	L'une de ses racines carrées

- 4 - Importer des fonctions depuis un module Python** Pour pouvoir utiliser une fonction fournie par un module, il faut l'avoir **importée** au préalable, en utilisant la syntaxe

`from nom_du_module import nom_de_la_fonction`

Voici un exemple de code qui importe puis utilise la fonction `sqrt` du module `math` :

```
from math import sqrt
print("La racine carrée de 2 vaut :", sqrt(2))
```

II Définir ses propres fonctions

- 5 - Syntaxe** La syntaxe pour définir une fonction en python est

`def nom_de_fonction (paramètre1, paramètre2, ...) :`
`bloc_de_code`

La ligne `def nom_de_fonction (paramètre1, paramètre2, ...) :` est appelé **l'entête** de la fonction, tandis que le bloc de code indenté après l'entête est appelé **le corps** de fonction.

Chaque paramètre doit être un nom de variable (idéalement pas encore utilisé dans le programme).

Quand une fonction est appelée, il se passe plusieurs choses :

- On affecte à chaque paramètre la valeur de l'argument correspondant, le temps d'évaluer le corps de la fonction
- On exécute le corps de la fonction
- Si, pendant l'exécution du corps de la fonction, on croise une ligne de la forme

`return expression`

on évalue *expression* (soit v la valeur vers laquelle s'évalue *expression*) puis on arrête l'exécution du corps de la fonction : l'appel de fonction en cours s'évalue lui aussi vers v .

- 6 - Exemple** Considérons le code suivant

```
1 def double(x):
2     return 2*x
3
4 print( double(2) )
```

Les lignes 1 et 2 définissent une fonction `double` avec un seul paramètre `x`.

À la ligne 4, quand on évalue l'appel `double(2)`, il se passe les choses suivantes :

- Le paramètre `x` reçoit la valeur 2 (car c'est la valeur de l'argument de `double` dans l'appel `double(2)`).
- On exécute le corps de la fonction `double`, à savoir `return 2*x`
- Puisqu'il s'agit d'un `return`, l'appel `double(2)` va s'évaluer vers la même valeur vers laquelle s'évalue `2*x`, à savoir 4.
- On a fini d'évaluer l'appel de fonction, donc désormais, ce n'est plus le cas que x soit associée à la valeur 2.

Puisque `double(2)` s'évalue vers 4, ce code affiche 4.

- 7 - Exercice** Quel est l'effet du code suivant ? (Vous devrez être capable d'expliquer en détail à l'oral pourquoi)

```
def max_2(x, y):
    if y > x:
        return y
    else:
        return x

print( max_2(2, 3) )
print( max_2(5, 1) )
```

8 - Exercice Qu'affiche le code suivant ?

```
def double(x):  
    return 2*x  
  
x = 3  
print(double(2) + x)
```

9 - Autre exemple Voici un code qui définit une fonction sans paramètres, qui renvoie 1 à chaque fois qu'elle est appelée :

```
def constante_1():  
    return 1
```

10 - Exercice de lecture Que font les fonctions `f` et `g` définies dans le code suivant ?

```
from math import sqrt  
  
def f(a, b):  
    return sqrt(a**2 + b**2)  
  
def g(n):  
    for d in range(2, n+1):  
        if n % d == 0:  
            return (d, n//d)
```

11 - Exercice d'écriture Écrivez une fonction `max_3` qui prend trois nombres en entrée et renvoie le plus grand d'entre eux.

Astuce : votre fonction `max_3` peut appeler la fonction `max_2` définie au paragraphe 7.

III Portée des variables

12 - Remarque et définition (variables locales) Affecter une valeur à une variable dans le corps d'une fonction n'a d'effet que jusqu'à la fin de l'appel en cours d'évaluation.

Par exemple, si on tente d'exécuter le code

```
1 def quadruple(x):  
2     y = 2*x  
3     return 2*y  
4  
5 print(double(3))  
6 print(y)
```

le code commence par afficher 12 (jusqu'à la ligne 5, aucun problème), puis déclenche une erreur

`NameError` : name 'y' is not defined

à cause de la ligne 6.

On dit de la variable `y` qu'elle est **locale** à la fonction `double`.

13 - Définition (variables globales) Par opposition aux variables locales, une variable qui est définie à l'extérieur de toute fonction est dite **globale**.

Par exemple, dans le code suivant

```
x = 3  
  
def affiche_2x():  
    print(2*x)  
  
affiche_2x()  
x = 4  
affiche_2x()
```

la variable `x` est globale. (Ce code affiche 6, puis 8).

14 - En cas d'ambiguïté, la variable locale prévaut... Une variable locale et une variable globale peuvent avoir le même nom (c'est maladroit, évitez de faire ça). Dans ce cas, à l'intérieur de la fonction qui définit la variable locale, c'est la variable locale qui va être utilisée **et non** la variable globale.

Par exemple, le code suivant

```
y = 7  
  
def double(x):  
    y = 2*x  
    return y  
  
print(double(2))
```

a pour effet d'afficher 4.

15 - ... et ce même si elle n'est pas encore définie ! Si on tente d'exécuter le code suivant

```
1 y = 7  
2  
3 def double(x):  
4     print(y)  
5     y = 2*x  
6     return y  
7  
8 print(double(2))
```

on obtient l'erreur suivante

`UnboundLocalError`: cannot access local variable 'y' where it is not associated with a value

Car en exécutant la ligne 5 (lors de l'évaluation de l'appel de la ligne 8), Python a besoin de la valeur de la variable **locale** `y`, qui n'a pas encore été définie.