



Programmation avec Python

Module 03 : Les fonctions et paramètres



OBJECTIFS

Python : les fonctions et paramètres
OBJECTIFS

Structurer son application

Connaître la syntaxe d'une
fonction en Python

Savoir utiliser une fonction

Comprendre les spécificités
des valeurs de retour



SOMMAIRE

Python : les fonctions et paramètres
SOMMAIRE



Procédure et fonction



Passage de paramètres



Valeurs de retour



Expressions lambda

- Syntaxe des procédures :

```
def my_procedure():  
    pass  
    # bloc d'instructions  
    # fin de la procédure
```

- Syntaxe des fonctions :

```
def my_function():  
    # bloc d'instructions  
    return a_value # fin de la méthode  
    # fin de la fonction
```

- Exemple d'appel :

- `my_procedure()` # appel de procédure traitée
- `my_fonction()` # appel de fonction traitée
- `not_treated()` # `NameError: name 'not_treated' is not defined`



La fonction appelée doit avoir été traitée préalablement par l'interpréteur !

- Description par des exemples :

- `def my_function_1(parametre_1, param_2, prm_3, param_N)`
- `def my_function_2(argument_1, arg_2, limite=100):` *# avec une valeur par défaut*
- `def my_function_3(a=15, b=24, c=36, d=47):` *# avec que des valeurs par défaut*

- Exemples d'appel :

- `my_function_1(1, 2)` *# TypeError: ... missing N required positional arguments: ...*
- `my_function_1(123, 456.78, "Paramètre", 'N')` *# avec tous les arguments renseignés*
- `my_function_2(8, 5.94)` *# argument_1 = 8, arg_2 = 5.94, limite = 100*
- `my_function_2(6, 4.67, 52)` *# argument_1 = 6, arg_2 = 4.67, limite = 52*
- `my_function_3(c=6, b=4, a=5, d=7)` *# a = 5, b = 4, c = 6, d = 7*
- `my_function_3(d=58, a=26.8)` *# a = 26.8, b = 24, c = 36, d = 58*

- Syntaxe :

```
def my_function():  
    # bloc d'instructions  
    return a_value, other_value # au moins une valeur par return  
# fin de la méthode
```

- Exemples d'Appel :

- `my_function()`
- `a_result, other_result = my_function()` # récupération unitaire des valeurs
- `a_result = my_function()` # récupération d'un ensemble de valeurs (n-uplet)



Cette spécificité est appelée **retour de valeurs multiples**

- Syntaxe :
 - `def my_function(*args)`

```
def make_sum(*integers):  
    total = 0  
    for integer in integers:  
        total += integer  
    return total  
  
print("La somme de 1, 2, 3 et 4 est", make_sum(1, 2, 3, 4))
```

La somme de 1, 2, 3 et 4 est 10

- Syntaxe :
 - `def my_function(**kwargs)`

```
def describe_user(**attributes):  
    result = "La personne a pour "  
    for key, value in attributes.items():  
        result += str(key) + " " + str(value) + " et "  
    return result[:-4] # suppression des 4 derniers caractères  
  
print(describe_user(last_name="DUPONT", first_name="Jean"))
```

La personne a pour last_name DUPONT et first_name Jean

- Utilisation de la fonction `type()` :
 - Chaque variable est un *objet*
 - Retourne le *type* de l'*objet* passé en paramètre
 - Un *objet* est une *instance* issue d'une *classe*
 - Le type de l'*instance* correspondant au nom de sa *classe*
- Syntaxe :
 - `type(a_variable)` # renvoi un objet de type 'type'
- Exemple :
 - `type(True)` # renvoi un objet de type 'type': <class 'bool'>
 - `print(type(24))` # affiche : "<class 'int'> »
 - `print(type(3.14))` # affiche : "<class 'float'> »
 - `print(type("Chaîne de caractères"))` # affiche : "<class 'str'>"

- Définition :
 - Instruction qui se comporte comme une petite fonction
 - Peut prendre n'importe quel nombre d'arguments, mais renvoie qu'une seule expression.
- Syntaxe :
 - `my_lambda = lambda arg1, arg2, ... : result`
- Exemple :
 - `power = lambda nb, exp : nb ** exp # nombre puissance exposant`
 - `what_else = lambda arg_condition, arg_else : arg_condition or arg_else`
 - `type(what_else) # renvoi : <class 'function'>`

DÉMONSTRATION

Création et utilisation de fonctions



**TRAVAUX
PRATIQUES**

Création et utilisation de fonctions