Les_fonctions

July 16, 2025

1 Les fonctions

1.1 Définition

0Les fonctions sont une partie importante de notre code. Elles ont plusieurs utilités : - Alléger réutiliser du code - Segmenter notre code

Il existe 2 types de fonctions : les **fonctions sans valeur retour** et les **fonctions avec valeur retour**.

Exemples de fonctions sans valeur retour : 1. La fonction **print( )** - Exécute l'action d'afficher du texte à la console. 2. Ajouter utilisateur à la base de données : **ajouter_utlisateur( )** - Exécute l'action d'ajouter un utilisateur

→ Ces fonctions nous retournent rien (**None**) mais exécutent seulement

Exemples de fontions avec valeur retour : 1. La fonction **type()** : - On souhaite récupérer le type d'une variable, donc **type( )** nous retournera quelque chose. 2. Récupérer l'email d'un utlisateur : - On souhaite récupérer l'adresse mail d'un utilisateur, donc cette fonction nous retournera l'adresse mail de l'utilisateur souhaitée.

Exemple d'une fonction

[1]:
```
# Exemple d'une fonction avec un retour de valeur

def ma_fonction():
    """
    Cette fonction ne fait rien de spécial.
    Elle est juste là pour illustrer
    le retour d'une valeur.
    """
    return 44

# Appel de la fonction
ma_fonction()
```

[1]:
```
44
```

[2]:
```
# On peut stocker le résultat dans une variable
resultat = ma_fonction()
print(resultat)
```

```
44
```

[3]:
```python
# On peut aussi faire un appel direct
print(ma_fonction())
```

```
44
```

1.2 Paramètres et arguments

[4]:
```python
def affiche_un_message(chaine):                                    # ----> On met un␣
 ↪paramètre
    """
    Cette fonction affiche une chaîne de caractères.
    """
    print(chaine)

# Appel de la fonction avec une chaîne de caractères
affiche_un_message("Bonjour, le monde !")                          # ----> On passe␣
 ↪un argument

# Appel de la fonction avec une autre chaîne de caractères
affiche_un_message("Le PSG est en finale de LDC !")                # ----> On passe␣
 ↪un autre argument

affiche_un_message()                                               # ----> On oublie␣
 ↪de passer un argument
```

```
Bonjour, le monde !
Le PSG est en finale de LDC !
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[4], line 13
     10 # Appel de la fonction avec une autre chaîne de caractères
     11 affiche_un_message("Le PSG est en finale de LDC !")           # ----> 0␣
 ↪passe un autre argument
---> 13 affiche_un_message()                                         # ----> 0␣
 ↪oublie de passer un argument

TypeError: affiche_un_message() missing 1 required positional argument: 'chaine
```

[5]:
```python
def ma_phrase(chaine1, chaine2):
    """
    Cette fonction affiche deux chaînes de caractères.
    """
    print(chaine1)
    print(chaine2)
```

```
ma_phrase("Va t-il qualifier l'équipe en CDM ?","On verra.")
```

```
Va t-il qualifier l'équipe en CDM ?
On verra.
```

[6]:
```
def affiche_un_message(chaine="Carlo Ancelotti est sélectionneur du Brésil"):
    # ----> On met un paramètre avec une valeur par défaut
    """
    Cette fonction affiche une chaîne de caractères.
    """
    print(chaine)

# Appel de la fonction sans argument
affiche_un_message()

# Appel de la fonction avec un argument
affiche_un_message(ma_phrase("Va t-il qualifier l'équipe en CDM ?","On verra."))  # ----> On passe un argument

# >>>> Rien nous empêche de faire un appel de fonction dans une autre fonction
```

```
Carlo Ancelotti est sélectionneur du Brésil
Va t-il qualifier l'équipe en CDM ?
On verra.
None
```

[7]:
```
def diviser(a, b):
    """
    Cette fonction divise deux nombres.
    """
    return a / b
# Appel de la fonction diviser
resultat_1 = diviser(10, 5)         # ----> Appel de la fonction avec des arguments
resultat_2 = diviser(b=10, a=5)     # ----> On passe les arguments dans le désordre

print("Résultat de la division sans inversion: {}".format(resultat_1))  # ----> On affiche le résultat
print("Résultat de la division avec inversion: {}".format(resultat_2))  # ----> On affiche le résultat
```

```
Résultat de la division sans inversion: 2.0
Résultat de la division avec inversion: 0.5
```

1.2.1 Résumé

- On peut définir un ou plusieurs paramètres dans une fonction
- Ces paramètres deviennent des varaiables qu'on peut utiliser à l'intérieur de la fonction

- On peut définir des valeurs par défaut pour les paramètres
- On peut spécifier explicitement à quel paramètre on envoie une valeur lors de l'appel de la fonction

1.3 Définir les fonctions au bon endroit

Il est important de définir les fonctions au bon endroit car python lira le script ligne par ligne et si la fonction est appelée avant d'être définie, Python nous affichera une erreur.

[8]:
```
# Définition de test() au mauvais endroit
# On l'appelle avant de la définir

test()

def test():
    print("Great Teacher Onizuka !")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[8], line 1
----> 1 test()
      3 def test():
      4     print("Great Teacher Onizuka !")

NameError: name 'test' is not defined
```

[10]:
```
# Appel de la fonction test au bon endroit
def test():
    print("Great Teacher Onizuka !")
test()
```

```
Great Teacher Onizuka !
```

[ ]:
```
def test():
    suggestion_manga()  # ----> On l'appelle avant de la définir

# -------------------- DEFINTION DE LA FONCTION ----------------------------

def suggestion_manga():
    """
    Cette fonction affiche une suggestion de manga.
    """
    print("Je te conseille de lire le manga 'Dragon Ball' !")

test()
```

```
Je te conseille de lire le manga 'Dragon Ball' !
```

Et oui ! Lorsque l'on définit une fonction, Python la garde dans sa mémoire. C'est pour cela que même si j'ai appelé la fonction suggestion_manga() avant de la définir, du moment que qu'elle est définie plus tard, cela fonctionnera.

1.4 Espace global/local

[ ]:
```
def test_2():
    a = 10
    print(a)  # ----> On affiche la valeur locale de a

a = 5

test_2()
print(a)  # ----> On affiche la valeur globale de a

# Pourquoi a n'a pas été modifié (5 ----> 10) ?
# La variable a est définie dans la portée locale de la fonction
# La variable a est définie dans la portée globale du programme
# La portée locale de la fonction est différente de la portée globale du␣
 ↪programme
```

```
10
5
```

Pourquoi la valeur de a n'a pas été modifié ? (5 —-> 10) ?

- La variable a est définie dans la portée locale de la fonction
- La variable a est définie dans la portée globale du programme
- La portée locale de la fonction est différente de la portée globale du programme

[15]:
```
def test_3():
    b = 10

a = 5
test_3()
print(a)
print(b)  # ----> On veut afficher la valeur de b
```

```
5
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[15], line 7
      5 test_3()
      6 print(a)
----> 7 print(b)  # ----> On affiche la valeur de b
      8 # La variable b n'est pas définie dans la portée globale du programme
```

```
NameError: name 'b' is not defined
```

Pourquoi b n'est pas défini ?

- La variable b n'est pas définie dans la portée globale du programme
- La variable b est définie dans la portée locale de la fonction
- La portée locale de la fonction est différente de la portée globale du programme

[1]:
```
b = 5
def exemple():
    print(b)  # ----> On affiche la valeur de b

exemple()     # ----> Appel de la fonction
```

```
5
```

Pourquoi la fonction exemple( ) a affiché 5 ? - La variable b est définie dans l'espace global du programme - Et donc elle est accessible dans la fonction exemple( ) - La portée locale de la fonction exemple( ) est différente de la portée globale du programme.

[ ]:
```
def fonction():
    a = 10     # ----> Variable locale

a = 5          # ----> Variable globale

# Ces deux variables ont beau avoir le même nom mais elles sont différentes
```

1.4.1 Résumé

- Les variables définies à l'intérieur d'une fonction ne sont pas disponibles en dehors de la fonction
- On peut accéder à une variable globale à l'intérieur d'une fonction
- On a toujours un espace global mais on peut créer plusieurs espaces locaux

2 Autres