12_python_basic_input_output

November 27, 2023

1 Python Basic Input and Output

En python, on peut utiliser la fonction `print()` pour afficher du texte en output

[ ]:
```
from time import sleep
for i in range(10):
    print("le chiffre actuel est :", i, "valeurs", sep="\t", flush=True)
    sleep(1)
```

Ici, `print()` prend un seul paramètre, qui est une chaîne de caractères (string). Cependant, `print()` attend **5** paramètres

`print(object=, sep=, end=, file=, flush=)`

- `object` : la/les variable(s) à afficher
- `sep (optionnel)` : le séparateur entre les `objects` (par défaut, un espace)
- `end (optionnel)` : le caractère à ajouter à la fin de l'affichage (par défaut, un retour à la ligne)
- `file (optionnel)` : le fichier dans lequel écrire (par défaut, la sortie standard `sys.stdout`)
- `flush (optionnel)` : Valeur booléenne indiquant s'il faut vider le buffer après l'affichage (par défaut, `False`)

1.0.1 Exemple 1 : Python Print Statement

Affichage de deux String, ici la valeur de `end` n'est pas renseignée, donc par défaut un retour à la ligne est ajouté à la fin de l'affichage. L'affichage des deux string se fera sur deux ligne différentes.

[ ]:
```
print('Good Morning!', end="\t")
print('It is rainy today')
```

1.0.2 Exemple 2 : Python Print() avec end

Affichage de deux String, ici la valeur de `end` est renseignée, donc un espace est ajouté à la fin de l'affichage. L'affichage des deux string se fera sur une seule ligne.

[ ]:
```
# print with end whitespace
print('Good Morning!', end= ' ')

print('It is rainy today')
```

Exemple 3: Python print() avec sep parameter

Affichage de plusieurs objets, ici la valeur de `sep` est renseignée, donc un séparateur est ajouté entre les objets.

[ ]:
```
def un():
    return 1
print('New Year', 3+13, un(), 'See you soon!', sep= '. ')
```

1.0.3 Example: Print Python Variables and Literals

On peut utiliser `print()` pour afficher des variables et des littéraux (littéraux = valeurs littérales, c'est à dire des valeurs qui ne sont pas stockées dans une variable)

[ ]:
```
number = -10.6

name = "Antonio"

# print literals
print(5)

# print variables
print(number)
print(0xA)
print(number, name, sep="\n")
```

1.0.4 Exemple: Print avec concatenation de string

On peut `join` 2 strings ensemble a l'intérieur de la fonction `print()` avec l'opérateur `+`

[ ]:
```
print('Hello ' + 'World.')
```

2 Output formatting

On peut utiliser la fonction `format()` pour formater l'affichage de variables dans une chaîne de caractères.

[ ]:
```
x = 5
y = 10
z = 10.555555555

print('The value of x is {} and y is {}'.format(x,y))
print('I love {1} and {0}'.format('bread','butter'))

# on peut aussi formatter l'affichage des variables
print('{greeting}, Hello {name} '.format(greeting = 'Good Morning', name =␣
 ↪'John'))


# on peut aussi formatter les nombres a afficher
```

```
print("z is %0.2f"%(z))
print("z is {0:.2f}".format(z))
```

2.1 F-Strings: A New and Improved Way to Format Strings in Python

Depuis Python 3.6, on peut utiliser les `f-strings` pour formater l'affichage de variables dans une chaîne de caractères.

[ ]:
```
# old style formatting
x = 12.3456789
print(f'The value of x is {{{x:.2f}}}')

# new style formatting
print('The value of x is {:.2f}'.format(x))
print(f"The value of x is {x:.2f}")

name = 'John'
greeting = 'Good Morning'

print(f"Hello {name.upper()}, {greeting}")
```

3 Python Input

En python, il est possible de demander à l'utilisateur d'entrer une valeur avec la fonction `input()`

[ ]:
```
hello = input('Enter your name: ')
print(type(hello))
```

Ici, on affichera le string: 'Enter your name' et l'utilisateur pourra entrer une valeur. Cette valeur sera stockée dans la variable `user_input`

[ ]:
```
# using input() to take user input
num = input('Enter a number: ')
print('You Entered:', num)
print('Data type of num:', type(num))

num_int = int(input('Enter a number: '))
print(f'You entered: {num_int}')
print(f'Data type of num_int: {type(num_int)}')
```