Python Basic Input and Output

Python Output

In Python, we can simply use the print() function to print output. For example,

```
print('Python is powerful')

# Output: Python is powerful
```

Here, the `print()` function displays the string enclosed inside the single quotation.

Syntax of print()

In the above code, the `print()` function is taking a single parameter.

However, the actual syntax of the print function accepts **5** parameters

```
print(object= separator= end= file= flush=)
```

Here,

- **object** - value(s) to be printed
- **sep** (optional) - allows us to separate multiple **objects** inside `print()`.
- **end** (optional) - allows us to add add specific values like new line `"\n"`, tab `"\t"`
- **file** (optional) - where the values are printed. It's default value is `sys.stdout` (screen)
- **flush** (optional) - boolean specifying if the output is flushed or buffered. Default: `False`

Example 1: Python Print Statement

```
print('Good Morning!')
print('It is rainy today')
```

Output

```
Good Morning!
It is rainy today
```

In the above example, the `print()` statement only includes the **object** to be printed. Here, the value for **end** is not used. Hence, it takes the default value `'\n'`.

So we get the output in two different lines.

Example 2: Python print() with end Parameter

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

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

Output

```
Good Morning! It is rainy today
```

Notice that we have included the `end= ' '` after the end of the first `print()` statement.

Hence, we get the output in a single line separated by space.

Example 3: Python print() with sep parameter

```
print('New Year', 2023, 'See you soon!', sep= '. ')
```

Output

```
New Year. 2023. See you soon!
```

Â

In the above example, the `print()` statement includes multiple **items** separated by a comma.

Notice that we have used the optional parameter `sep= ". "` inside the `print()` statement.

Hence, the output includes items separated by `.` not comma.

Example: Print Python Variables and Literals

We can also use the `print()` function to print Python variables. For example,

```
number = -10.6

name = "Programiz"

# print literals
print(5)

# print variables
print(number)
print(name)
```

Output

```
5
-10.6
Programiz
```

Example: Print Concatenated Strings

We can also join two strings together inside theÂ `print()` statement. For example,

```
print('Programiz is ' + 'awesome.')
```

Output

```
Programiz is awesome.
```

Here,

- the `+` operator joins two strings `'Programiz is '` and `'awesome.'`
- the `print()` function prints the joined string

Output formatting

Sometimes we would like to format our output to make it look attractive. This can be done by using the `str.format()` method. For example,

```
x = 5
y = 10

print('The value of x is {} and y is {}'.format(x,y))
```

Here, the curly braces `{}` are used as placeholders. We can specify the order in which they are printed by using numbers (tuple index).

To learn more about formatting the output, visit Python String format().

Python Input

While programming, we might want to take the input from the user. In Python, we can use the input() function.

Syntax of input()

```
input(prompt)
```

Here, `prompt` is the string we wish to display on the screen. It is optional.

Example: Python User Input

```
# using input() to take user input
num = input('Enter a number: ')

print('You Entered:', num)

print('Data type of num:', type(num))
```

Output

```
Enter a number: 10
You Entered: 10
Data type of num: <class 'str'>
```

In the above example, we have used the `input()` function to take input from the user and stored the user input in the *num* variable.

It is important to note that the entered value **10** is a string, not a number. So, `type(num)` returns `<class 'str'>`.

To convert user input into a number we can use int() or float() functions as:

```
num = int(input('Enter a number: '))
```

Here, the data type of the user input is converted from string to integer .

Table of Contents