# **Python | Output using print() function**

**Python print() function** prints the message to the screen or any other standard output device.

**Syntax:** print(value(s), sep= ‘ ‘, end = ‘\n’, file=file, flush=flush)

**Parameters:**

* **value(s) :** Any value, and as many as you like. Will be converted to string before printed
* **sep=’separator’ :** (Optional) Specify how to separate the objects, if there is more than one.Default :’ ‘
* **end=’end’:** (Optional) Specify what to print at the end.Default : ‘\n’
* **file :** (Optional) An object with a write method. Default :sys.stdout
* **flush :** (Optional) A Boolean, specifying if the output is flushed (True) or buffered (False). Default: False

**Returns:** It returns output to the screen.

Though it is not necessary to pass arguments in the print() function, it requires an empty parenthesis at the end that tells python to execute the function rather calling it by name. Now, let’s explore the optional arguments that can be used with the print() function.

## **String Literals**

String literals in python’s print statement are primarily used to format or design how a specific string appears when printed using the print() function.

* **\n :** This string literal is used to add a new blank line while printing a statement.
* **“” :** An empty quote (“”) is used to print an empty line.

**Example:**

|  |
| --- |
| print("GeeksforGeeks \n is best for DSA Content.") |

**Output:**

GeeksforGeeks

is best for DSA Content.

## **end= ” ” statement**

The end keyword is used to specify the content that is to be printed at the end of the execution of the print() function. By default, it is set to “\n”, which leads to the change of line after the execution of print() statement.

### **Example: Python print() without new line.**

|  |
| --- |
| # This line will automatically add a new line before the  # next print statement  print("GeeksForGeeks is the best platform for DSA content")    # This print() function ends with "\*\*" as set in the end argument.  print("GeeksForGeeks is the best platform for DSA content", end="\*\*")  print("Welcome to GFG") |

**Output:**

GeeksForGeeks is the best platform for DSA content

GeeksForGeeks is the best platform for DSA content\*\*Welcome to GFG

## **flush Argument**

The I/Os in python are generally buffered, meaning they are used in chunks. This is where flush comes in as it helps users to decide if they need the written content to be buffered or not. By default, it is set to false. If it is set to true, the output will be written as a sequence of characters one after the other. This process is slow simply because it is easier to write in chunks rather than writing one character at a time. To understand the use case of the flush argument in the print() function, let’s take an example.

**Example:**

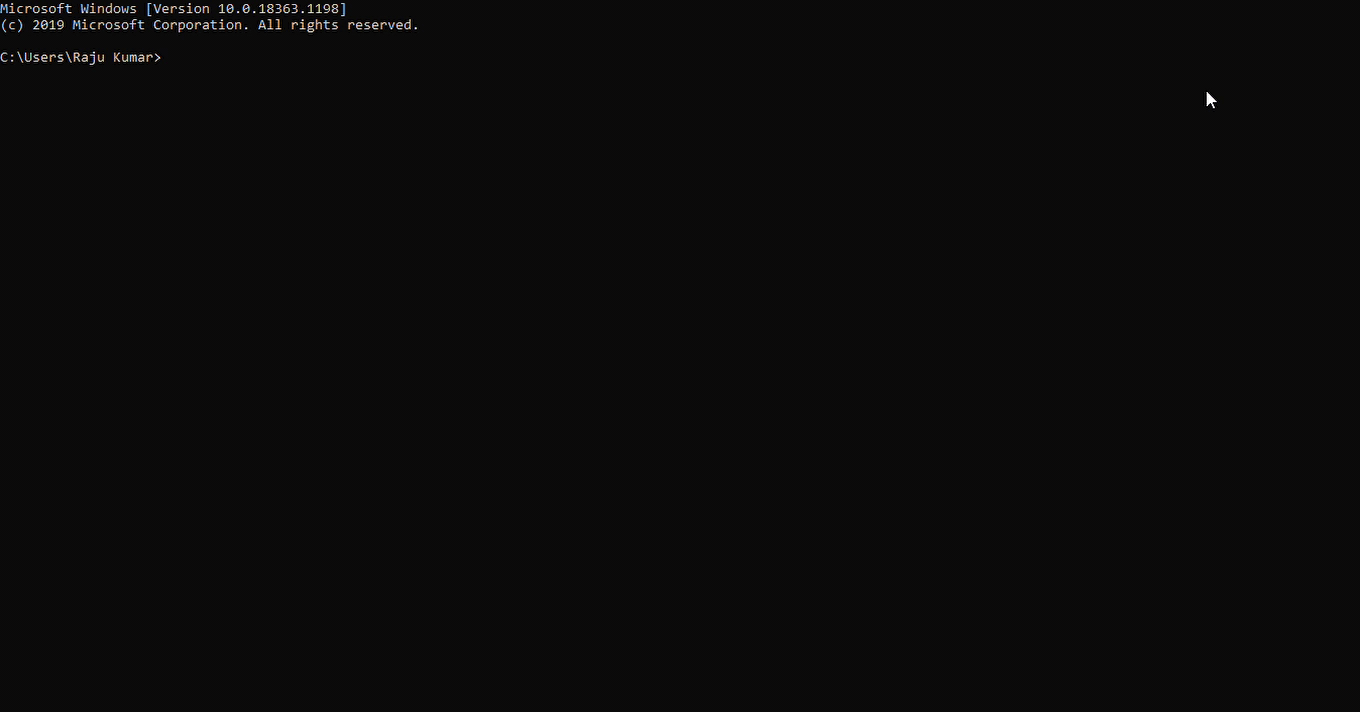
Imagine you are building a countdown timer, which appends the remaining time to the same line every second. It would look something like below:

3>>>2>>>1>>>Start

The initial code for this would look something like below;

|  |
| --- |
| importtime    count\_seconds =3  fori inreversed(range(count\_seconds +1)):      ifi > 0:          print(i, end='>>>')          time.sleep(1)      else:          print('Start') |

So, the above code adds text without a trailing newline and then sleeps for one second after each text addition. At the end of the countdown, it prints Start and terminates the line. If you run the code as it is, it waits for 3 seconds and abruptly prints the entire text at once. This is a waste of 3 seconds caused due to buffering of the text chunk as shown below:



Though buffering serves a purpose, it can result in undesired effects as shown above. To counter the same issue, the flush argument is used with the print() function. Now, set the flush argument as true and again see the results.

|  |
| --- |
| importtime    count\_seconds =3  fori inreversed(range(count\_seconds +1)):      ifi > 0:          print(i, end='>>>', flush =True)          time.sleep(1)      else:          print('Start') |

**Output:**

Video Player

00:00

00:10

## **Separator**

The print() function can accept any number of positional arguments. These arguments can be separated from each other using a **“,” separator**. These are primarily used for formatting multiple statements in a single print() function.

**Example:**

|  |
| --- |
| b ="for"    print("Geeks", b , "Geeks") |

**Output:**

Geeks for Geeks

## **file Argument**

Contrary to popular belief, the print() function doesn’t convert the messages into text on the screen. These are done by lower-level layers of code, that can read data(message) in bytes. The print() function is an interface over these layers, that delegates the actual printing to a stream or **file-like object**. By default, the print() function is bound to *sys.stdout* through the file argument.

### **Example: Python print() to file**

|  |
| --- |
| importio    # declare a dummy file  dummy\_file =io.StringIO()    # add message to the dummy file  print('Hello Geeks!!', file=dummy\_file)    # get the value from dummy file  dummy\_file.getvalue() |

**Output:**

'Hello Geeks!!\n'

### **Example : Using print() function in Python**

|  |
| --- |
| # Python 3.x program showing  # how to print data on  # a screen    # One object is passed  print("GeeksForGeeks")    x =5  # Two objects are passed  print("x =", x)    # code for disabling the softspace feature  print('G', 'F', 'G', sep='')    # using end argument  print("Python", end='@')  print("GeeksforGeeks") |

**Output:**

GeeksForGeeks

x = 5

GFG

Python@GeeksforGeeks