Python print() function

The function accepts several arguments as shown below with their default values:

```
print(*objects, sep=' ', end= '\n', file=sys.stdout, flush=False)
```

Parameter Values

Parameter	Description
object(s)	Any object, 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 is ' '
end='*end*'	Optional. Specify what to print at the end. Default is '\n' (line feed)
file	Optional. An object with a write method. Default is sys.stdout
flush	Optional. A Boolean, specifying if the output is flushed (True) or buffered (False). Default is False

The print() function literally prints the result to the screen. To use (or call) this function, you simply type:

```
print()
```

Usually, you need some parameters inside the brackets; print() without any parameters will result in a blank line. You can print the value of a variable or the result of a function. Let's use the variable we've already created to show how the print() function works with a parameter:

```
>>> favourite_food = 'ice cream!'
>>> print(favourite_food)
ice cream!
```

You can also add some text to these values. Combining pieces of text is called 'string concatenation' because string is Python's data type for text. Here's how we concatenate strings inside print():

```
>>> print('My favourite food is ', food)
My favourite food is ice cream!
```

ice cream!

Optional keyword arguments:

- sep: It can be a string which you would like to insert between values, defaults to space.

```
print('python','print','function')

python print function
```

Let's put a list of words inside the print function and separate them with a new line. Remember, by default; the separator adds a blank space between each word.

```
print('python','print','function',sep='\n')   #`\n` will put each word in a new line

python
print
function
```

Similarly, you can separate them with a comma, or add two \n which will add an empty line in between or even add a plus (+) sign.

```
print('python','print','function',sep=',')
python,print,function
```

```
print('python','print','function',sep=',+')
python,+print,+function
```

- end: It is a string appended after the last value, defaults to a newline. It allows the programmer to define a custom ending character for each print call other than the default newline or \n.

```
>>>str1 = 'tutorial on'
>>>str2 = 'python print function'

>>>print(str1)
>>>print(str2)
tutorial on
python print function


>>>print(str1,end=' ')
>>>print(str2)
tutorial on python print function

>>> result=10
>>> print("Result of experiment", result, sep=" is ", end="!")
Result of experiment is 10!
```

- file: A file-like object (stream); defaults to the current sys.stdout. Here you can mention the file in which you would like to write or append the output of the print function.

```
>>> open_file = open('abc.txt', 'w')
>>> string = " print('File Demo')"
   >>> print(string, file=open_file)
   >>> open_file.close()
```

- flush : The flush keyword is a recent addition to the print() function. It accepts a Boolean, either True or False. Normally, the output to a file or the console is buffered. The buffer is "flushed" (or printed) only when it is full or when a newline is reached. Setting this keyword to True forcibly flushes the buffer before it is full or before a newline character is encountered.
- Typically, output to a file or the console is buffered, with text output at least until you print a newline (Source). Buffer means that the output is stored in some kind of a register where the output resides before it is ready to be stored or the file is finally closed. The job of the flush is to make sure that the output, which is being buffered, goes to the destination safely.

```
import time
print('Please enter your email-id : ', end=' ')
#print('Please enter your email-id : ', end=' ', flush=True) #run this to see the difference and comment out the above line of code.
time.sleep(5)
```

If you run the above lines of code in the terminal, you will notice that the prompt string does not show up until the sleep timer ends and the program exits. However, if you add flush=True in the print function, you will see the prompt, and then you will have to wait for 5 seconds for the program to finish.

Let's see some other ways of printing variable values inside the print function.

You can make use of the format argument wherein you can pass as many variables as you want to print. When you pass the variables in the format function, you need to specify the index numbers (order in which they are placed inside the format argument) in the predefined string. So this print function will now act as a template for you.
Another thing to remember is that here the index numbers are represented with a curly bracket {} representing a placeholder.

```
>>>a = 2
>>>b = "ITM"
>>>print("{0} is an integer while {1} is a string.".format(a,b))
2 is an integer while ITM is a string.
```

```
>>>print('We are the {} who say "{}!"'.format('knights', 'Ni'))
We are the knights who say "Ni!"
```

```
>>>print('This {food} is {adjective}.'.format(food='spam', adjective='absolutely horrible'))
This spam is absolutely horrible.
```

```
print('The story of {0}, {1}, and {other}.'.format('Bill', 'Manfred',other='Georg'))
The story of Bill, Manfred, and Georg.
```

- Similar to a format argument where your print function acts as a template, you have a percentage (%) sign that you can use to print the values of the variables.

Like format argument, this also has a concept of placeholders. However, unlike the format function where you pass in just the index numbers, in this, you also need to specify the datatype the placeholder should expect.

  - %d is used as a placeholder for numeric or decimal values.
  - %s is used as a placeholder for strings.

```
>>>a = 2
>>>b = " ITM"
>>>print("%d is an integer while %s is a string."%(a,b))
2 is an integer while ITM is a string.
```

```
>>>pi= 3.14159265359
>>>print('The value of pi is approximately %5.3f.' % )
The value of pi is approximately 3.142.
```