Python | Output using print() function

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

Example

In this example, we have created three variables integer, string and float and we are printing all the variables with print() function in Python.

- Python3

```
name = "John"

age = 30


print("Name:", name)

print("Age:", age)
```

Output

```
Name: John
Age: 30
```

Python print() Function Syntax

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 a 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*

Return Type: *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 than calling it by name. Now, let's explore the optional arguments that can be used with the print() function.

How print() works in Python?

You can pass variables, strings, numbers, or other data types as one or more parameters when using the print() function. Then, these parameters are represented as strings by their respective str() functions. To create a single output string, the transformed strings are concatenated with spaces between them.

In this code, we are passing two parameters name and age to the print function.

- Python3

```python
name = "Alice"

age = 25


print("Hello, my name is", name, "and I am", age, "years old.")
```

Output :

```
Hello, my name is Alice and I am 25 years old.
```

Python print() Function with Examples

Python 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

This code uses \n to print the data to the new line.

- Python3

```python
print("GeeksforGeeks \n is best for DSA Content.")
```

Output

```
GeeksforGeeks
 is best for DSA Content.
```

Python "end" parameter in print()

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

In this example, we are using print() with end and without end parameters.

- Python3

```python
# 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 parameter in Python with print() function

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 as follows:

- Python3

```python
import time


count_seconds = 3

for i in reversed(range(count_seconds + 1)):

    if i > 0:

        print(i, end='>>>')

        time.sleep(1)

    else:

        print('Start')
```

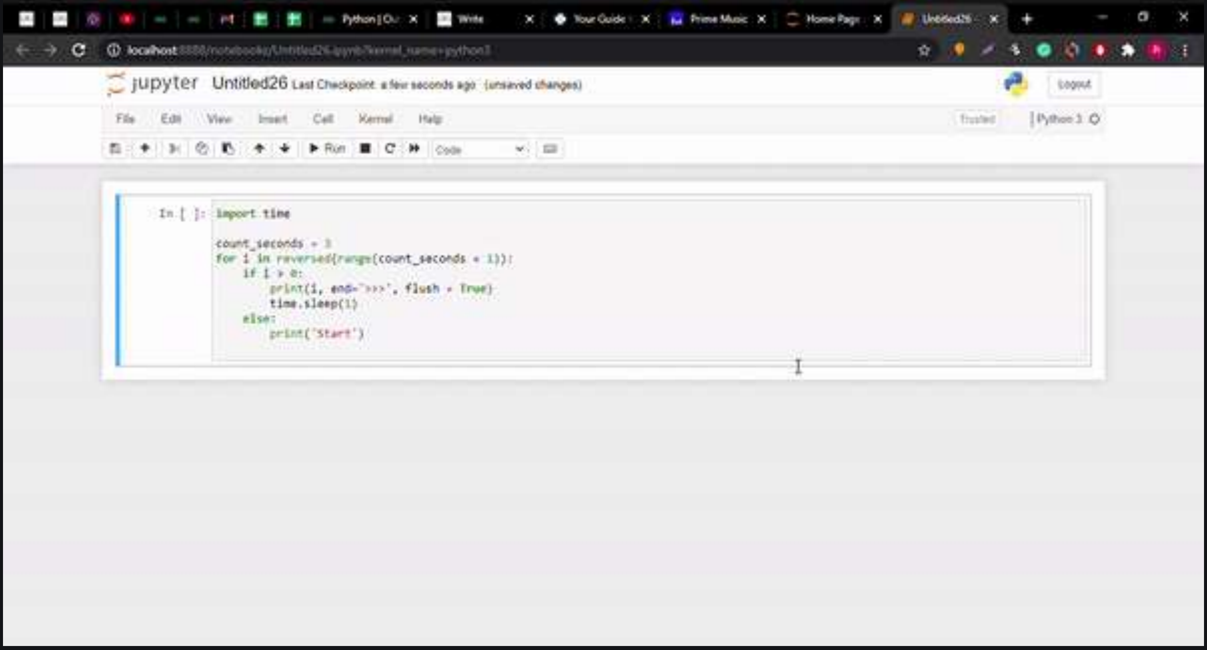
So, the above code adds text without a trailing newline and then sleeps for one second after each text addition. At the end of the countdown, it prints Start and terminates the line. If you run the code as it is, it waits for 3 seconds and abruptly prints the entire text at once.

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.

- Python3

```python
import time


count_seconds = 3

for i in reversed(range(count_seconds + 1)):

    if i > 0:
        print(i, end='>>>', flush = True)

        time.sleep(1)

    else:

        print('Start')
```

Output



```
In [ ]: import time

        count_seconds = 3
        for i in reversed(range(count_seconds + 1)):
            if i > 0:
                print(i, end='>>>', flush = True)
                time.sleep(1)
            else:
                print('Start')
```

Python print() flush argument

Python “sep” parameter in print()

The print() function can accept any number of positional arguments. To separate these positional arguments, the keyword argument “sep” is used.

Note: As sep, end, flush, and file are keyword arguments their position does not change the result of the code.

Example

This code is showing that how can we use the sep argument for multiple variables.

- Python3

```
a=12

b=12

c=2022

print(a,b,c,sep="-")
```

Output :

```
12-12-2022
```

Example

Positional arguments cannot appear after keyword arguments. In the below example **10**, **20** and **30** are positional arguments where **sep=’ – ‘** is a keyword argument.

- Python3

```
print(10, 20, sep=' - ', 30)
```

Output :

```
  File "0b97e8c5-bacf-4e89-9ea3-c5510b916cdb.py", line 1
    print(10, 20, sep=' - ', 30)
                            ^
SyntaxError: positional argument follows keyword argument
```

File Argument in Python print()

Contrary to popular belief, the print() function doesn't convert 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.

With IO Module

This code creates a dummy file using the io module in Python. It then adds a message "**Hello Geeks!!**" to the file using the print() function and specifies the file parameter as the dummy file.

- Python3

```
import io


# 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

print(dummy_file.getvalue())
```

Output

```
Hello Geeks!!
```

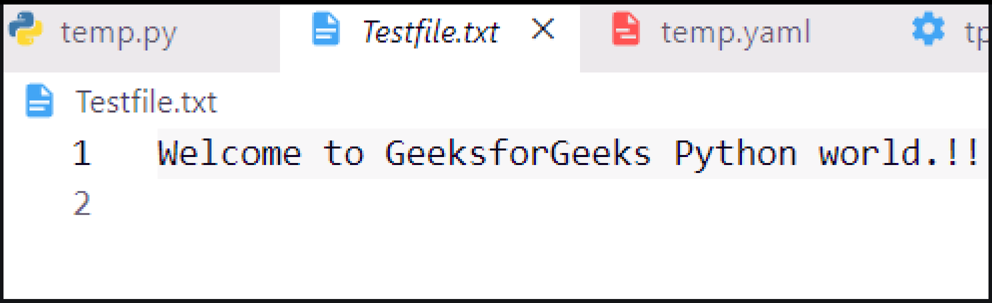
Writing to a File with Python's print() Function

This code is writing the data in the **print()** function to the text file.

- Python3

```
print('Welcome to GeeksforGeeks Python world.!!', file=open('Testfile.txt', 'w'))
```

Output


Testfile.txt

```
Welcome to GeeksforGeeks Python world.!!
```

Python Print()

How to print without newline in Python?

Generally, people switching from C/C++ to Python wonder how to print two or more variables or statements without going into a new line in python. Since the Python print() function by default ends with a newline. Python has a predefined format if you use print(a_variable) then it will go to the next line automatically.

Example

```
Input: [geeks,geeksforgeeks]
Output: geeks geeksforgeeks

Input: a = [1, 2, 3, 4]
Output: 1 2 3 4
```

- Python3

```
print("geeks")

print("geeksforgeeks")
```

Output

```
geeks
geeksforgeeks
```

But sometimes it may happen that we don't want to go to the next line but want to print on the same line. So what we can do? The solution discussed here is totally dependent on the Python version you are using.

Print without newline in Python 2.x

In Python 2.x, the `print` statement does not have the `end` parameter like in Python 3.x. To achieve the same behavior of printing without a newline in Python 2. x, you can use a comma at the end of the `print` statement, just like in the given code.

- Python

```
# Python 2 code for printing

# on the same line printing

# geeks and geeksforgeeks

# in the same line



# Without newline
```

```python
print("geeks"),

print("geeksforgeeks")


# Array

a = [1, 2, 3, 4]


# Printing each element on the same line

for i in xrange(4):

    print(a[i]),
```

Output

```
geeks geeksforgeeks

1 2 3 4
```

Print without newline in Python 3.x

In Python 3.x, the `print()` function behaves slightly differently from Python 2.x. To print without a newline in Python 3. x, you can use the `end` parameter of the `print()` function.

- python3

```python
# Python 3 code for printing

# on the same line printing

# geeks and geeksforgeeks

# in the same line


print("geeks", end =" ")

print("geeksforgeeks")


# array

a = [1, 2, 3, 4]


# printing a element in same

# line

for i in range(4):

    print(a[i], end =" ")
```

Output

```
geeks geeksforgeeks
1 2 3 4
```

Print without newline in Python 3.x without using For Loop

In Python 3. x, you can print without a newline without using a `for` loop by using the `sep` parameter of the `print()` function. The `sep` parameter specifies the separator to be used between multiple items when they are printed.

- Python3

```python
# Print without newline in Python 3.x without using for loop


l = [1, 2, 3, 4, 5, 6]


# using * symbol prints the list

# elements in a single line

print(*l)
```

Output

```
1 2 3 4 5 6
```

Print without newline Using Python sys module

To use the sys module, first, import the module sys using the import keyword. Then, make use of the stdout.write() method available inside the sys module, to print your strings. It only works with string If you pass a number or a list, you will get a TypeError.

- Python3

```python
import sys


sys.stdout.write("GeeksforGeeks ")

sys.stdout.write("is best website for coding!")
```

Output

```
GeeksforGeeks is best website for coding!
```

Python end parameter in print()

By default Python's print() function ends with a newline. A programmer with C/C++ background may wonder how to print without a newline. Python's print() function comes with a parameter called 'end'. By default, the value of this parameter is '\n', i.e. the new line character.

Example 1:

Here, we can end a print statement with any character/string using this parameter.

- Python3

```
# ends the output with a space

print("Welcome to", end = ' ')

print("GeeksforGeeks", end= ' ')
```

Output:

```
Welcome to GeeksforGeeks
```

Example 2:

One more program to demonstrate the working of the end parameter.

- Python3

```
# ends the output with '@'

print("Python", end='@')

print("GeeksforGeeks")
```

Output:

```
Python@GeeksforGeeks
```

Example 3:

The print() function uses the sep parameter to separate the arguments and ends after the last argument.

- Python3

```
print('G','F', sep='', end='')

print('G')

#\n provides new line after printing the year

print('09','12','2016', sep='-', end='\n')
```

```python
print('Red','Green','Blue', sep=',', end='@')

print('geeksforgeeks')
```

Output

```
GFG

09-12-2016

Red,Green,Blue@geeksforgeeks
```

Using end to concatenate strings:

In this example, we use the end parameter to concatenate the two print() statements into a single line of output. The end parameter is set to a space character " " for the first print() statement, so the second print() statement will start on the same line, separated by a space character.

The end parameter is a useful feature of the print() function in Python that can be used to control the formatting of output in various ways.

- Python3

```python
name = "Alice"

age = 30

print("My name is", name, "and I am", age, "years old.", end=" ")

print("Nice to meet you!")
```

Output

```
My name is Alice and I am 30 years old. Nice to meet you!
```

Python | sep parameter in print()

The separator between the arguments to print() function in Python is space by default (softspace feature) , which can be modified and can be made to any character, integer or string as per our choice. The 'sep' parameter is used to achieve the same, it is found only in python 3.x or later. It is also used for formatting the output strings.

Examples:

- Python3

```
#code for disabling the softspace feature

print('G','F','G', sep='')


#for formatting a date

print('09','12','2016', sep='-')


#another example

print('pratik','geeksforgeeks', sep='@')
```

Output:

```
GFG

09-12-2016

pratik@geeksforgeeks
```

The sep parameter when used with the end parameter it produces awesome results. Some examples by combining the sep and end parameters.

- Python3

```
print('G','F', sep='', end='')

print('G')

#\n provides new line after printing the year

print('09','12','2016', sep='-', end='\n')


print('prtk','agarwal', sep='', end='@')

print('geeksforgeeks')
```

Output:

```
GFG

09-12-2016
```

```
prtkagarwal@geeksforgeeks
```

Note: Please change the language from Python to Python 3 in the online ide.
Go to your interactive python ide by typing python in your cmd (windows) or terminal (linux)

- Python3

```python
#import the below module and see what happens

import antigravity

#NOTE - it wont work on online ide
```

If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.
using the sep parameter in the print() function:
By default, the sep parameter is set to a space character, so if you don't specify it explicitly, the values will be separated by a space.

Approach:
The code is using the print() function to print out strings with different separators. The sep parameter of the print() function is used to specify the separator between the strings. In the first example, a comma is used as the separator, in the second example, a semicolon is used, and in the third example, an emoji is used.

Time Complexity:
The time complexity of the print() function is O(n), where n is the total number of characters to be printed. However, the time complexity of specifying a separator is O(1), as it is a constant time operation.

Space Complexity:
The space complexity of the code is also O(n), where n is the total number of characters to be printed. This is because the print() function needs to allocate memory to store the strings and separators before printing them out.

Overall, the code has a constant time complexity for specifying the separator, and a linear time and space complexity for printing out the strings and separators.

- Python3

```python
# using a comma separator

print('apples', 'oranges', 'bananas', sep=', ')

# output: apples, oranges, bananas



# using a semicolon separator
```

```python
print('one', 'two', 'three', sep=';')

# output: one;two;three


# using an emoji separator

print('????', '????', '????', sep='????')

# output: ????????????????????
```

Output

```
apples, oranges, bananas

one;two;three

????????????????????
```

Python | Output Formatting

In Python, there are several ways to present the output of a program. Data can be printed in a human-readable form, or written to a file for future use, or even in some other specified form. Users often want more control over the formatting of output than simply printing space-separated values.

Output Formatting in Python

There are several ways to format output using String Method in Python.

- Using String Modulo Operator(%)
- Using Format Method
- Using The String Method
- Python's Format Conversion Rule

Formatting Output using String Modulo Operator(%)

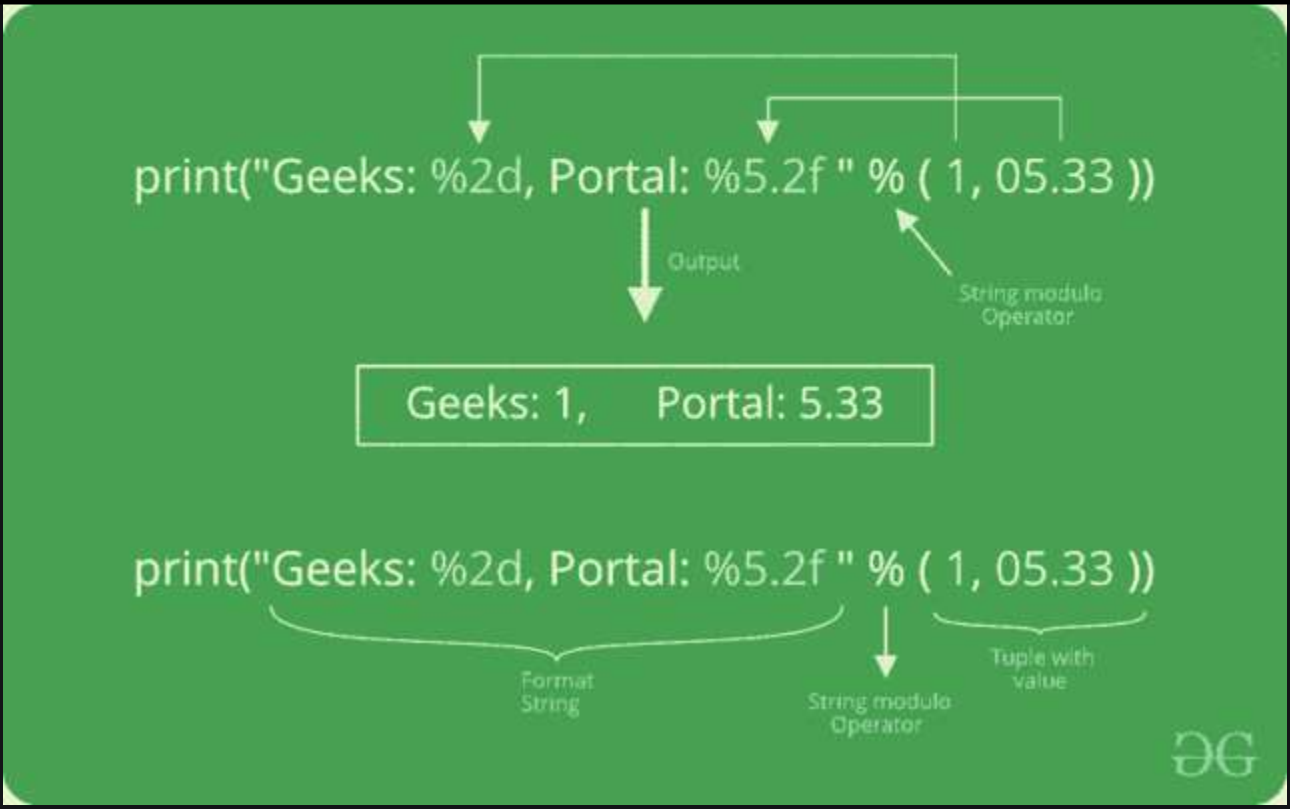
The Modulo % operator can also be used for string formatting. It interprets the left argument much like a printf()-style format as in C language strings to be applied to the right argument. In Python, there is no printf() function but the functionality of the ancient printf is contained in Python. To this purpose, the modulo operator % is overloaded by the string class to perform string formatting. Therefore, it is often called a string modulo (or sometimes even called modulus) operator. The string modulo operator (%) is still available in Python(3.x) and is widely used. But nowadays the old style of formatting is removed from the language.

- Python3

```
# Python program showing how to use string modulo operator(%)


print("Geeks : %2d, Portal : %5.2f" % (1, 05.333))


print("Total students : %3d, Boys : %2d" % (240, 120))   # print integer value


print("%7.3o" % (25))   # print octal value


print("%10.3E" % (356.08977))   # print exponential value
```

Output

```
Geeks :  1, Portal : 5.33
Total students : 240, Boys : 120
    031
3.561E+02
```


print("Geeks: %2d, Portal: %5.2f " % (1, 05.33))
Output
String modulo Operator
Geeks: 1,   Portal: 5.33
print("Geeks: %2d, Portal: %5.2f " % (1, 05.33))
Format String
String modulo Operator
Tuple with value

Output Formatting using Modulo Operator

There are two of those in our example: "%2d" and "%5.2f". The general syntax for a format placeholder is:

```
%[flags][width][.precision]type
```

Let's take a look at the placeholders in our example.

- The first placeholder '%2d' is used for the first component of our tuple, i.e. the integer 1. It will be printed with 2 characters, and as 1 consists of only one digit, the output is padded with 1 leading blank.
- The second placeholder '%5.2f' is for a float number. Like other placeholders, it's introduced with the % character. It specifies the total number of digits the string should contain, including the decimal point and all the digits, both before and after the decimal point.
- Our float number 05.333 is formatted with 5 characters and a precision of 2, denoted by the number following the '.' in the placeholder. The last character 'f' indicates that the placeholder represents a float value.

Formatting Output using The Format Method

The format() method was added in Python(2.6). The format method of strings requires more manual effort. Users use {} to mark where a variable will be substituted and can provide detailed formatting directives, but the user also needs to provide the information to be formatted. This method lets us concatenate elements within an output through positional formatting. For Example –

Example 1: The code explain various Python string formatting techniques.The values are either explicitly supplied or referred to by the order in which they appear in the format() procedure.f-Strings enable the use of curly braces and the f prefix to embed expressions inside string literals. The f-Strings' expressions are assessed and their appropriate values are substituted for them.

- Python3

```
print('I love {} for "{}!"'.format('Geeks', 'Geeks'))


# using format() method and referring a position of the object

print('{0} and {1}'.format('Geeks', 'Portal'))


print('{1} and {0}'.format('Geeks', 'Portal'))


print(f"I love {'Geeks'} for \"{'Geeks'}!\"")


# using format() method and referring a position of the object

print(f"{'Geeks'} and {'Portal'}")
```

Output

```
I love Geeks for "Geeks!"
Geeks and Portal
Portal and Geeks
I love Geeks for "Geeks!"
Geeks and Portal
```

The brackets and characters within them (called **format fields**) are replaced with the objects passed into the format() method. A number in the brackets can be used to refer to the position of the object passed into the format() method.

Example 2: With the help of positional parameters and a named argument ('other') in the first line, the values 'Geeks', 'For', and 'Geeks' are added to the string template.'Geeks:12, Portal: 0.55′ is printed, with the first value appearing as a 2-digit integer and the second number having 2 decimal places and an 8-bit width. The format() method's named arguments, denoted by specific labels ('a' and 'p') for the numbers '453' and '59.058',

- Python3

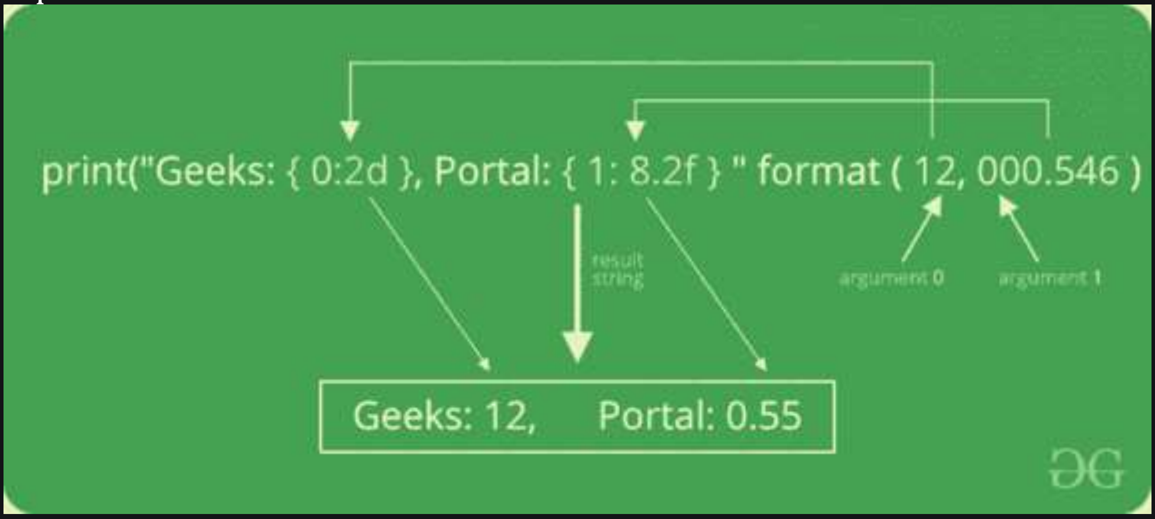
```
# combining positional and keyword arguments

print('Number one portal is {0}, {1}, and {other}.'

     .format('Geeks', 'For', other ='Geeks'))


# using format() method with number

print("Geeks :{0:2d}, Portal :{1:8.2f}".

      format(12, 00.546))


# Changing positional argument
```

```
print("Second argument: {1:3d}, first one: {0:7.2f}".

      format(47.42, 11))


print("Geeks: {a:5d},  Portal: {p:8.2f}".

     format(a = 453, p = 59.058))
```

Output

```
Number one portal is Geeks, For, and Geeks.
Geeks :12, Portal :    0.55
Second argument:  11, first one:   47.42
Geeks:   453, Portal:    59.06
```

The following diagram with an example usage depicts how the format method works for positional parameters:


print("Geeks: { 0:2d }, Portal: { 1: 8.2f } " format (12, 000.546)
result string
argument 0
argument 1
Geeks: 12,   Portal: 0.55

Output Formatting using Format method

Example 3: The code shows how to use dictionaries with Python's format() method. The dictionary's 'tab' in the first example has keys and associated values. The format() method uses indexing to put the values into the string template. In the second example, named keys in a dictionary are used as "data.

- Python3

```
tab = {'geeks': 4127, 'for': 4098, 'geek': 8637678}


# using format() in dictionary

print('Geeks: {0[geeks]:d}; For: {0[for]:d}; '

    'Geeks: {0[geek]:d}'.format(tab))


data = dict(fun ="GeeksForGeeks", adj ="Portal")
```

```python
print("I love {fun} computer {adj}".format(**data))
```

Output

```
Geeks: 4127; For: 4098; Geeks: 8637678
I love GeeksForGeeks computer Portal
```

Formatting Output using The String Method

This output is formatted by using ***string method i.e.*** slicing and concatenation operations. The string type has some methods that help in formatting output in a fancier way. Some methods which help in formatting an output are str.ljust(), str.rjust(), and str.centre()

- Python3

```python
cstr = "I love geeksforgeeks"


# Printing the center aligned string with fillchr

print("Center aligned string with fillchr: ")

print(cstr.center(40, '#'))


# Printing the left aligned string with "-" padding

print("The left aligned string is : ")

print(cstr.ljust(40, '-'))


# Printing the right aligned string with "-" padding

print("The right aligned string is : ")

print(cstr.rjust(40, '-'))
```

Output

```
Center aligned string with fillchr: 
##########I love geeksforgeeks##########
The left aligned string is : 
I love geeksforgeeks--------------------
The right aligned string is : 
--------------------I love geeksforgeeks
```

Python's Format Conversion Rule

This table lists the standard format conversion guidelines used by Python's format() function.

Conversion	**Meaning**
d	Decimal integer
b	Binary format

Conversion	**Meaning**
o	octal format
u	Obsolete and equivalent to ‘d’
x or X	Hexadecimal format
e or E	Exponential notation
f or F	Floating-point decimal
g or G	General format
c	Single Character
r	String format(using repr())
s	String Format(using str())
%	Percentage