Python Tutorials and Notes

Python String format() with EXAMPLES

SETScholars & WACAMLDS

What is Python Format?

Python format() function helps us to replace, substitute, or convert the string with placeholders with valid values in the final string. The placeholders inside the string are defined in curly brackets, e.g., "Welcome to Guru99 {}".format('value here'). It is built-in function of the string class.

In this tutorial, you will learn:

Syntax:

```
templatestring.format(val1, val2...)
```

Parameters

val1, val2 ... : The values that need to replace in the given template string that has placeholders in the form of curly brackets {}. The placeholders can be a string, key/value pair, integers, floating-point numbers, characters, etc.

Return value:

It will return the final string, with valid values replaced in place of the placeholders given in curly brackets.

Placeholders

The placeholders in the template string are represented using curly brackets, e.g. {}. The placeholder can be empty {}, or it can have a variable for e.g {name} , or it can have a number index e.g {0} , {1} etc.

How string format() works?

The string format() will scan the original strings for placeholders. The placeholders can be empty curly brackets {}, positional arguments i.e the string can have placeholders with index 0, 1 for e.g {0}, {1} etc.

For keyword arguments the variable name will be present inside the curly brackets for e.g {name}, {age}. In the case of empty curly brackets, the values from the format will be replaced inside the curly brackets in sequence.

The first value will be replaced with the first empty curly bracket, followed by the next one. For positional arguments, the index will start from 0 and so on. The values will be available in format separated with commas, and the 0^{th} value will point to the first value inside format and so on.

For Keyword arguments, i.e., when you use a variable inside your placeholders, you can have the order of values inside the format as you need.

The order does not matter here as the values will be replaced based on the variable name present in the format(). Here are a few examples of how to use placeholders inside a string as empty, positional, and using keywords or variables.

Example: Empty Placeholder replaced with a string value

In the example below, the string has empty curly brackets{}. The value given to the format will get replaced inside the curly brackets{}.

The value that we want to be replaced is a string.

Example:

Using format, we want the curly brackets {} to be replaced with a string value. The value is given to format("Guru99"). On execution, the curly brackets {} is replaced with Guru99, and you will get the final string as Welcome to Guru99 tutorials.

```
print ("Welcome to {} tutorials".format("Guru99"))
```

Output:

```
Welcome to Guru99 tutorials
```

Example: Empty Placeholder replaced with a numeric value

In the example below, we want the numeric value to be replaced inside the original string. The curly brackets{} are added to the place where you need the numeric value. When it executes, the empty curly brackets {} is replaced with the numeric value.

Example:

You can also make use of format() to insert numbers inside your string. The example will show how to replace the empty Placeholder {} with number 99 present inside format().

```
print ("Welcome to Guru{} Tutorials".format("99"))
```

Output:

```
Welcome to Guru99 Tutorials
```

Example: Using variable or keyword arguments inside the Placeholder

It is also possible to make use of variables inside the curly brackets, as shown in the example below. The variables are defined inside format(). Therefore, when it executes, the value assigned to the variable is replaced inside the original string.

Example:

You can use variables inside curly brackets for example {name} {num}.The values for name and num variable are available inside format i.e. format(name="Guru", num="99"). The values given to name and num will be replaced inside the {name} and {num}.

```
print ("Welcome to {name}{num} Tutorials".format(name="Guru", num="99"))
```

Output:

```
Welcome to Guru99 Tutorials
```

Example: Using index or positional arguments inside the Placeholder

The values can be replaced using indexes like 0, 1, 2 inside the placeholders. The value will be picked in order from the format(), as shown in the example below.

Example:

```
print ("Welcome to {0}{1} Tutorials".format("Guru","99"))
```

Output:

```
Welcome to Guru99 Tutorials
```

Example: Using multiple placeholders inside a string

In this example, we are going to use multiple empty placeholders.

Example:

The string used in the example has multiple empty placeholder and each placeholder will refer to a value inside format() .The first value will be replaced for the first placeholder and so on.

```
print ("{} is {} new kind of {} experience!".format("Guru99", "totally","learning"))
```

Output:

```
Guru99 is a new kind of learning experience
```

Formatting inside Placeholders

You have seen that it is possible to have Placeholder as empty, with a variable or an index. It is also possible that you can apply some formatting inside the Placeholder.

Here is the list of formats

Format	Description	Example
:d	It will give the output in decimal format when used inside the placeholder	print("The binary to decimal value is : {:d}".format(0b0011)) **Output:** The binary to decimal value is : 3
:b	It will give the output in binary format when used inside the placeholder	print("The binary value is : {:b}".format(500)) **Output:** The binary value is : 111110100
:e	It will give the output in scientific format when used inside the placeholder, the exponent e in the output will be lowercase.	print("The scientific value is : {:e}".format(40)) **Output:** The scientific format value is : 4.000000e+01
:E	It will give the output in scientific format when used inside the placeholder, the exponent E in the output will be uppercase	print("The scientific value is : {:E}".format(40)) **Output:** The scientific value is : 4.000000E+01

Format	Description	Example
:f	This will output a fixed-point number format. By default, you will get the output of any number with six decimal places. In case you need up to 2 decimal places, use it as. 2f i.e.. a period (.) in front of 2f	print("The value is : {:f}".format(40)) **Output:** The value is : 40.000000 Example: Showing output upto 2 decimal places. print("The value is : {:.2f}".format(40)) **Output:** The value is: 40.00
:o	This will output octal format	print("The value is : {:o}".format(500)) **Output:** The value is : 764
:x	This will output hex format in lowercase	print("The value is : {:x}".format(500)) **Output:** The value is : 1f4
:X	This will output hex format in uppercase.	print("The value is : {:X}".format(500)) **Output:** The value is : 1F4

Format	Description	Example
:n	This will output number format.	print("The value is : {:n}".format(500.00)) **Output:** The value is : 500
:%	This will give the output in a percentage format. By default it will give 6 decimal places for the percentage output, in case you don't want any decimal value you can use period with 0 i.e {:.0%}.	print("The value is : {:%}".format(0.80)) **Output:** The value is : 80.000000% This example shows how to skip the decimal places by using {:.0%} inside the placeholder. print("The value is : {:.0%}".format(0.80)) **Output:** The value is: 80%
:_	This will output an underscore as a thousand separator. It is available from python 3.6+.	print("The value is {:_}".format(1000000)) **Output:** The value is : 1_000_000

Format	Description	Example
:,	This will output comma as a thousands separator	print("The value is : {:,}".format(1000000)) **Output:** The value is : 1,000,000 The comma(,) is added , as a thousand separator as shown in the output.
:	This will add a space before any positive numbers	This example shows how to add space or padding before the given number. The number 5 indicates the space count you want before the number. print("The value is: {:5}".format(40)) **Output:** The value is: 40
:-	This will add a minus sign before negative numbers	The example shows how to get the output with a minus (-) sign before the number using {:-}. print("The value is: {:-}".format(-40)) Output: The value is: -40

Format	Description	Example

:+	You can use plus sign to indicate the number is positive	The example shows how to get the output with a plus (+) sign before the number using {:+}.
		`print("The value is: {:+}".format(40))`
		Output:
		`The value is: +40`

:=	The equal to is used to place the +/- sign on the left side.	The example shows how to get the output with a plus (+/-) sign before equal to sign using {:=}.
		`print("The value is {:=}".format(-40))`
		Output:
		`The value is -40`

:^	This will center align the final result	The example shows to use {:^} to center align the text. The number 10 is used to add 10 spaces to show the center-aligned when the value is replaced.
		`print("The value {:^10} is positive value".format(40))`
		Output:
		`The value  40  is a positive value`
		Here, you can use 10 that will add 10 spaces in the final text, and the value to be replaced will be center-aligned between the 10 spaces. The spaces of 10 are added just to show the center alignment of the replaced value.

Format	Description	Example

:>	This will right-align the final result	The space of 10 is added using (:>10), and the value replaced is right-aligned.
		`print("The value {:>10} is positive value".format(40))`
		Output:
		`The value 40 is positive value`

:<	This will left align the final result	The space of 10 is added using (:<10), and the value replaces is left aligned.
		`print("The value {:<10} is positive value".format(40))`
		Output:
		`The value 40 is positive value`

Using class with format()

In this example, we are creating a class and use the object of the class inside the format. The placeholders will refer to class properties or members using the class object.

Example:

The class is called inside the format(c=MyClass()).The object c will have the reference to the properties and methods inside class MyClass().

```
class MyClass:
    msg1="Guru"
    msg2="Tutorials"

print("Welcome to {c.msg1}99 {c.msg2}!".format(c=MyClass()))
```

Output:

```
Welcome to Guru99 Tutorials!
```

Using dictionary with format()

It is also possible to make use of dictionary inside format() as shown in the example below:

```
my_dict = {'msg1': "Welcome", 'msg2': 'Guru99'}
print("{m[msg1]} to {m[msg2]} Tutorials!".format(m=my_dict))
```

Output:

```
Welcome to Guru99 Tutorials!
```

Padding Variable Substitutions

Using string.format() method, you can add padding, space by using placeholders inside your string.

Example:

In below example will add space inside the Placeholder using the format(). To add space, you have to specify the number of spaces inside curly brackets after the colon(:). So the Placeholder will look like {:5}.

```
print("I have {:5} dogs and {:5} cat".format(2,1))
```

Output:

```
I have     2 dogs and     1 cat
```

You can also give the index inside the placeholder for example: {0:5} where 0 will refer to the first value inside format.

```
print("I have {0:5} dogs and {1:5} cat".format(2,1))
```

Output:

```
I have     2 dogs and     1 cat
```

Summary

- Python string class gives us an important built-in command called format() that helps us to replace, substitute, or convert the string with placeholders with valid values in the final string.
- The placeholders inside the string are defined in curly brackets, e.g., "Welcome to Guru99 {}".format('value here').
- The placeholder can be empty {}, or it can have a variable for e.g {name} , or it can have a number index e.g {0} , {1} etc.
- You can make use of the formatters inside placeholders that can help to add padding, center align, and also help with number formatting.