Formatting Output

Programming for Data Science with Python

Overview

The user can format output using either of two methods available in Python.

- The modulo operator method
- The String format method

Both the methods are still popular. However, it is strongly encouraged to use the second method, i.e., the String format, now, and going forward.

---

1. Formatting output with Modulo Operator

1.1 Overview

In Python, the modulo operator “%” is overloaded by the string class to perform string formatting. · Also called string modulo (or sometimes even called modulus) operator · Another term for it is “string interpolation”: it interpolates various class types (int, float, etc.) into a string.

1.2 The Format

- On the left side of the “string modulo operator” is the format string.
- On the right side is a tuple with the contents.
- The values can be literals, variables, or arbitrary arithmetic expressions.

Format String	String Modulo Operator	Tuple with Values
print("Art: %5d, Price per Unit:%8.2f") 	%	453, 59.058

1.3 Format Placeholder

With the print function example: print("Art: %5d, Price per Unit:%8.2f" % (453, 59.058)

Let's first deal with the placeholder for "Art: %5d

- "The first format placeholder: “%5d” is a format description for a decimal.
- It is introduced with the “%” character.

- Then, it is followed by the total number of digits, 5, the string should contain.
- This number does not include any decimal points. The value of the placeholder is a whole number 453 is formatted with 5 characters (2 leading blanks or padding).

Now, let's deal with the placeholder for "Price per Unit: "%8.2f"

- "The second format placeholder: "%8.2f" is a format description for a float number.
- It is introduced with the "%" character.
- Then, it is followed by the total number of digits, 8, the string should contain.
- This number includes the decimal point and all the digits, i.e. before and after the decimal point:

The value of the placeholder is a float number 59.058 is formatted with 8 characters.

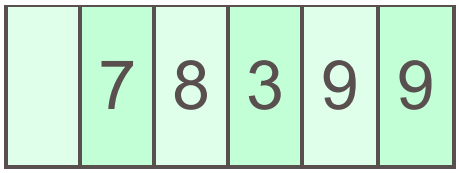
- The decimal part of the number or the precision is set to 2 (59).
- Finally, the last character "f" of our placeholder stands for "float".
- It is noticeable that the 3 decimal digits have been rounded.
- Furthermore, the number has been preceded in the output with 3 leading blanks.

IMPORTANT NOTES: By default, right adjusted is used.

1.4 Another Example

Using the format placeholder %6.2f for 5 different float numbers:

#	#	#	#	#	#
	2	3	.	5	6
		0	.	0	4
1	9	9	.	8	
1	2	5	7	.	8



1.5 List of Data Types

The most frequently used types are: d (Decimal), f (Float), and s (String).

Conversion	Meaning
d	Signed inter decimal
i	Signed inter decimal
o	Unsigned octal
u	Obsolete and equivalent to "d", i.e Signed inter decimal
x	Unsigned hexadecimal (lowercase)
X	Unsigned hexadecimal (uppercase)
e	Floating point exponential format (lowercase)
E	Floating point exponential format (uppercase)
f	Floating point decimal format (lowercase)
F	Floating point decimal format (uppercase)
g	Same as "e", if exponent is greater than -4 or less than precision, "f" otherwise
G	Same as "E", if exponent is greater than -4 or less than precision, "f" otherwise
c	Single character (accepts integer or single character string)
r	String (converts any Python object using repr()
s	String (converts any Python object using str()
%	No argument is converted, results in a "%" character in the result

1.6 Flags in Format Placeholder

The most frequently used flag is ‘ for left adjusted.

Flag	Meaning
#	Used with o, x, or X specifiers. The value is preceded with 0, 0o, 0O, or 0X, respectively

Flag	Meaning
0	The conversion result will be zero padded for numeric values.
-	The converted value is left adjusted.
	If no sign (e.g. minus sign is going to be written, a blank space is inserted before the value.
+	A sign character ("+" or "-") will precede the conversion (overrides a "space" flag).

1.7 Examples of Formatting Output Using Modulo Operator

****Run the following 7 code blocks:****

In [1]:
```
print("%10.3e"%(356.08977))
```
```
 3.561e+02
```

In [12]:
```
print("%3.5e"%(486.7508))
```
```
4.86751e+02
```

In [2]:
```
print("%2.3E"%(356.08977))
```
```
3.561E+02
```

In [16]:
```
print("%7.2E"%(4893631556.089771557))
```
```
4.89E+09
```

In [3]:
```
print("%10o"%(25))
```
```
        31
```

In [23]:
```
print("%1o"%(105))
```
```
151
```

In [4]:
```
print("%10.5o"%(25))
```
```
     00031
```

In [24]:
```
print("%15.7o"%(65))
```
```
        0000101
```

In [5]:
```
print("%5x"%(47))
```
```
   2f
```

In [27]:
```
print("%50x"%(128))
```
```
                                                80
```

In [6]:
```
print("%5.4X"%(47))
```
```
002F
```

In [28]:
```
print("%75x"%(256))
```

```
                                                                        100
```

In [7]:
```
print("Only one percentage sign: %% " %())
```

```
Only one percentage sign: %
```

2. Formatting Output Using String Method “format”

The general form of this method: template.format (p0, p1, ..., k0=v0, k1=v1,...)

2.1 An Example

IMPORTANT NOTES:

- We **DON’T USE** print() method
- We **USE** THE .format() method of the class String to format the output

"Art: {0:5d}, Price per Unit: {1:8.2f}".format (453, 59.058)

Result String

Variable	Value: Argument 0	Variable	Value: Argument 1
"Art:	453	Price per Unit:	59.06

The template (or format string) is a string that contains one or more **format codes** (fields to be replaced):

- The formal codes or “fields to be replaced” are surrounded by **curly braces {}**.
- The **curly braces** and the “code” inside will be substituted with a formatted value from one of the arguments.
- Anything else will be **literally printed**, i.e. without any changes.
- If a brace character has to be printed, it has to be escaped by doubling it: **{{** and **}}**.

There are two kinds of arguments for the .**format**() method:

- The list of arguments starts with 0 or more **positional arguments** (p0, p1, ...).
- It may be followed by 0 or more **keyword arguments** (k0, k1, ...) of the form **name=value**.

A **positional parameter** of the format method can be accessed by placing the index of the parameter after the opening brace, e.g. {0} accesses the first parameter {1} the second one and so on.

The **index** inside of the curly braces can be followed by a colon ':' and a format string (Similar to the notation of the string modulo as discussion in Section 2.)

IMPORTANT NOTES: If the positional parameters are used in the order in which they are written:

- The positional argument specifiers inside of the braces can be omitted.
- So '{} {} {}' corresponds to '{0} {1} {2}'.

(But they are needed, if you want to access them in different orders: '{2} {1} {0}').

2.2 Keyword Arguments

Look at above example in a different way. Pay attention to how the keyword arguments are used to format the output: a=453, b=590958

"Art: {a:5d}, Price per Unit: {p:8.2f}".format (a=453, p=59.058)

Result String

Variable	Keyword Argument 0	Variable	Keyword: Argument 1
"Art:	453	Price per Unit:	59.06

2.3 Options in Format Code

Option	Meaning
<	The field will be left-aligned with the available space. This is usually the default for strings.
>	The field will be right-aligned with the available space. This is usually the default for numbers.
0	If the width field is preceded by a zero (0) character, sign-aware zero padding for the numeric types will be enabled.
,	This option signals the use of a comma for a thousands separator.
=	Forces the padding to be placed after the sign (if any), but before the digits. This is used for printing fields in the form of "+000000120". This alignment option is only valid for numeric types.
^	Forces the field to be centered within the available space.

IMPORTANT NOTES: The most frequently used option is '<' and '>' for "left-justify" and "right-justify", respectively. By default, "left-justify" is used.

Run the following 8 code blocks:

In [31]:
```
x = 378
print ("The value is {:06d}".format (x))
```

```
The value is 000378
```

In [33]:
```
y = 123
print("Bond, the secret PIN is {:05d}".format (y))
```

```
The secret number is 00123
```

In [37]:
```
y = -378
print("The value is {:08d}".format (y))
```

```
The value is -0000378
```

In [36]:
```
sun_age = (4.5 * (10 ** 9))
print("The age of the sun is {:02f}".format (sun_age))
```

```
The combination to the safe is 4500000000.000000
```

In [38]:
```
a = 78962324245
print("The value is {:,}".format (a))
```

```
The value is 78,962,324,245
```

In [46]:
```
earth_radius_km = 6731
print("The combination to the safe is {:,}".format (earth_radius_km))
```

```
The combination to the safe is 6,731
```

In [40]:
```
b = 5897633423
print("The value is {0:6,d}". format(b))
```

```
The value is 5,897,633,423
```

In [48]:
```
earth_pop = 8118835999
print("The mass of the sun is {:5,d}".format (earth_pop))
```

```
The mass of the sun is 8,118,835,999
```

In [44]:
```
c = 5897653423897
print("The value is {0:12,.3f}". format(c))
```

```
The value is 5,897,653,423,897.000
```

In [45]:
```
sun_mass = (1.989 * (10 ** 30))
print("The mass of the sun is {:5,f}".format (sun_mass))
```

```
The mass of the sun is 1,989,000,000,000,000,183,384,231,903,232.000000
```

In [49]:
```
"{0:<20s} {1:6.2f}".format('Spam & Eggs:',6.99)
```

Out[49]:
```
'Spam & Eggs:           6.99'
```

In [54]:
```
"{0:<10s} {1:3.2f}".format("Dragon Dogma 2:", 69.99)
```

Out[54]:
```
'Dragon Dogma 2: 69.99'
```

In [55]:
```
"{0:>20s} {1:6.2f}".format('Spam & Eggs:',6.99)
```

Out[55]:
```
'         Spam & Eggs:   6.99'
```

In [59]:
```
"{0:>65s} {1:2.3f}".format('Horizon:Forbidden West :', 59.99)
```

Out[59]:
```
'                                                Horizon:Forbidden West : 59.990'
```

In [60]:
```
"{0:<0s} {1:6.2f}".format('Spam & Eggs:',6.99)
```

Out[60]:
```
'Spam & Eggs:   6.99'
```

In [67]:
```
"{0:>115} {1:5.2f}".format('Ghost of Tsushima:', 59.99)
```

Out[67]:
```
'
Ghost of Tsushima: 59.99'
```

2.3.1 Options: Signs

IMPORTANT NOTES: The sign options are used only for numeric values.

Option	Meaning
+	Indicates that a sign should be used for both positive as well as negative numbers.
-	Indicates that a sign should be used only for negative numbers, which is the default behavior.
	If no sign, this indicates that a leading space should be used on positive numbers, and a minus sign on negative numbers.

2.4 Examples of Formatting Output Using .format() Method of Class String

Run the following 6 code blocks:

In [68]:
```
"First argument :{0}, second one: {1}".format(47,11)
```

Out[68]:
```
'First argument :47, second one: 11'
```

In [69]:
```
"First argument :{0}, Last argument: {1}".format(1,100)
```

Out[69]:
```
'First argument :1, Last argument: 100'
```

In [ ]:
```
"Second argument :{1}, first one: {0}".format(47,11)
```

In [70]:
```
"Last argument :{1}, first argument: {0}".format(1,100)
```

Out[70]:
```
'Last argument :100, first argument: 1'
```

In [ ]:
```
"Second argument :{1:3d}, first one: {0:7.2f}".format(47.42,11)
```

In [74]:
```
"Last argument :{1:5,d}, first argument: {0:3.6f}".format(13579.2468, 975318642)
```

Out[74]:
```
'Last argument :975,318,642, first argument: 13579.246800'
```

In [75]:
```
"First argument :{}, second one: {}".format(47,11)
```

Out[75]:
```
'First argument :47, second one: 11'
```

In [76]:
```
"First argument :{}, second one: {}".format(50,150)
```

Out[76]:
```
'First argument :50, second one: 150'
```

In [79]:
```
"various precisions :{0:6.2f} or {0:6.3f}".format(1.4148)
```

Out[79]:
```
'various precisions :  1.41 or  1.415'
```

In [82]:
```
"various precisions :{0:5.3f} or {0:6.6f}".format(3.14159265)
```

Out[82]:
```
'various precisions :3.142 or 3.141593'
```

In [78]:
```
"first argument :{}, second one: {}".format(1.4148, 2.1678)
```

Out[78]:
```
'first argument :1.4148, second one: 2.1678'
```

In [83]:
```
"first argument :{}, second one: {}".format(3.14159265, 4.669201609102990671853203820
```

Out[83]:
```
'first argument :3.14159265, second one: 4.66920160910299'
```

In [ ]:
```

```