Python Formatted output: Old and New

K. S. Ooi
Foundation in Science
Faculty of Health and Life Sciences
INTI International University
Persiaran Perdana BBN, Putra Nilai,
71800 Nilai, Negeri Sembilan, Malaysia
E-mail: kuansan.ooi@newinti.edu.my
dr.k.s.ooi@gmail.com

Abstract

The new string formatter of Python 3 has more to offer than the old formatted output absorbed from C. The adjustment can be specified by either :> (move to right), :< (move to left), or :^ (move to center), which is quite intuitive. One can select character to pad spaces and no longer to have only whitespaces as padding characters. With or without precision specification, the new formatter is predictable and intuitive.

Keywords: Python 3, formatted output.
Date: Dec 25, 2020

1. The Old

In my previous article [1], I show by experimentation that Python absorbs the formatted output from C, *almost* having the identical conversion specifications (cc) of the format string. The following two figures summarize the characteristics of the cc, one without precision and one with precision.

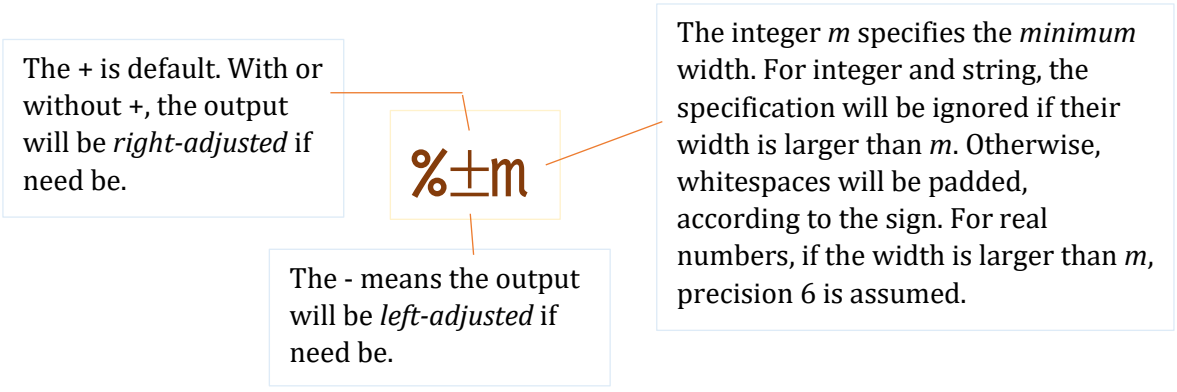

Figure 1: cc without precision.

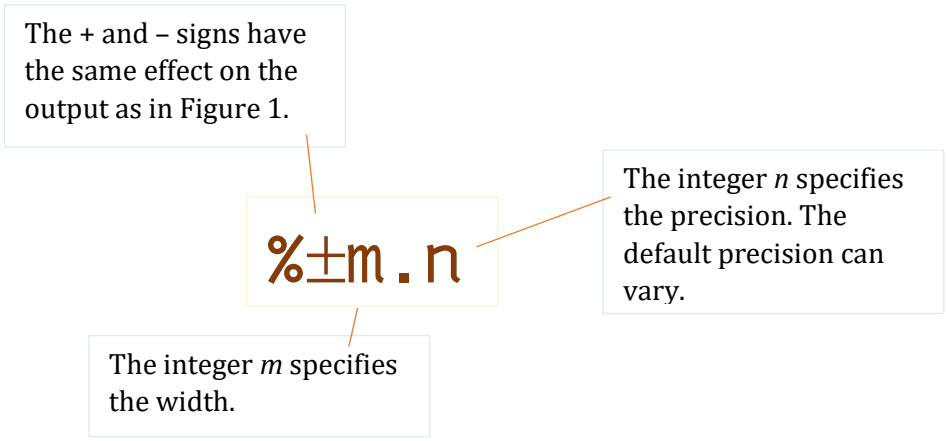

Figure 2: cc with precision

From the results presented in [1], you had better specify the width for outputting numbers in exponential format, and do not let the specification defaulting. The "old" format is shared between C and Python, and, of course, other languages, too.

2. The New

Since this article is written specifically for scientific computing, I will only focus of string and numbers. One interesting site you might want to visit is [2].

If the width *m* is larger than the string or number, we summarize the results of experimentation in the following table.

Table 1: In the cases that the width *m* is larger than the string or the numbers. The square brackets are included in the print statements to observe the whitespaces, if they appear.

Print statement	Output	Comment
print("[{:12}]".format(456))	[456]	The output is *right-adjusted*. This is the default.
print("[{:>12}]".format(456))	[456]	The same result as the previous print statement. The > is intuitive. It says move the number to the right.
print("[{:<12}]".format(456))	[456]	The output is *left-adjusted*. This is the default. The < directs the number to the left. Fairly intuitive.
print("[{:^12}]".format(4567811))	[4567811]	The ^ centralizes the output. If the centralization is not

		perfect, the output will be more toward to the left.
print("[{:>15}]".format("Mambo Jumbo"))	[Mambo Jumbo]	Integers and string have the same formatter.
print("[{:@>15}]".format("Mombo"))	[@@@@@@@@@@Mombo]	You can pad the spaces with characters of your choosing.
print("[{:@<15}]".format("Mombo"))	[Mombo@@@@@@@@@@]	The character must be in between : and <. The previous experiment, between : and >.

If you do not specify the width, integers and string will be printed in full. The results are summarized in Table 2.

Table 2: In the cases that the width *is not specified*.

Print statement	**Output**	**Comment**
print("[{:}]".format("My dog eats it!")) # or print("[{}]".format("My dog eats it!"))	[My dog eats it!]	The : is optional.
print("[{:>12}]".format(456))	[456]	The same result as the previous print statement. The > is intuitive. It says move the number to the right.
print("[{:d}]".format(23456789))	[23456789]	The whole number is printed.
print("[{:f}]".format(132.3456789094))	[132.345679]	The precision set for you is 6.
print("[{:e}]".format(1326.3456789094))	[1.326346e+03]	The exponential format, either e or E, has the default precision of 6.
print("[{:G}]".format(1326.3456789094))	[1326.35]	The formatter decides on the general format. The shorter one will be printed.
print("[{:g}]".format(1326345678.9094))	[1.32635e+09]	The general format. This time it prints the exponential format.

The cases where precision is specified are summarized in Table 3.

Table 3: In the cases that the precision *is specified*.

Print statement	Output	Comment
print("[{:.5}]".format("Her dog eats my work!"))	[Her d]	Select the first five characters of the string.
print("[{:15.5}]".format("Her dog eats my work!")) print("[{:<15.5}]".format("Her dog eats my work!")) print("[{:>15.5}]".format("Her dog eats my work!")) print("[{:^15.5}]".format("Her dog eats my work!")) print("[{:8<15.5}]".format("Her dog eats my work!")) print("[{:1>15.5}]".format("Her dog eats my work!")) print("[{:7^15.5}]".format("Her dog eats my work!"))	[Her d] [Her d] [Her d] [Her d] [Her d8888888888] [1111111111Her d] [77777Her d77777]	The formatter become very predictable.

References

1. K. S. Ooi, Formatted Output: The Forgotten C Facility, ePrint: https://vixra.org/abs/2012.0195 (2020)

2. PyFormat, Using % and .format() for great good!, at https://pyformat.info/ (accessed Dec 26, 2020)

Abbreviations used

cc - conversion specification