Formatting Output in Python v.3.6+

Reference: Input and Output, https://docs.python.org/3/tutorial/inputoutput.html
Reference: str.format(), https://docs.python.org/3/library/string.html#formatstrings
Reference: f-strings, https://docs.python.org/3/tutorial/inputoutput.html

---

The print command has optional parameters:

Parameter	Description	Usage	Result
sep()	Specify the separator between values.	print(m, d, y, sep='/')	m/d/y
end()	Specify the end-of-line character. (Allows you to suppress newline.)	print(var, end='*') print(var2)	var*var2

Commonly used escape characters:

Escape Character	Effect
\n	Generates a new line.
\t	Advance to next tab position.
\v	Vertical tab.
\'	Display a single quote.
\"	Display a double quote.
\\	Display a backslash.

Format-related string functions:

Method	Description	Usage
str()	Converts the argument to a string	str(someValue)
ljust()	Returns argument left-justified in a string of the width specified	str.ljust(width)
rjust()	Returns argument right-justified in a string of the width specified	str.rjust(width)
center()	Returns argument centered in a string of the width specified	str.center(width)
zfill()	Pads a numeric value on the left side with zeros if needed to fill the width	str.zfill(width)
format()	Formats the given value using the format specified.	str.format(value, format_spec)

Using the string formatting options:

The string formatting method can be used in one of three different syntax styles:

1. `str.format()` # substitutes values for placeholders in the string
2. `format(str, specifier)` # returns str formatted according to the specifier
3. `f'str'` # embeds values and formatting directly in placeholders

---

The `str.format()` method allows you to format a string using `{}` as placeholders with an optional format specifier. Values specified as arguments to the `str.format()` method replace the placeholders using optional formatting. Placeholders can be numeric values or keywords. Or if placeholder values are not included, values are substituted in order.

Example	Result
'Number {0}'.format(5)	Number 5
'Cracker {other}'.format(other='Jack')	Cracker Jack
'Multiple of {0} or {1}'.format(1,2)	Multiple of 1 or 2
'{} {} {}'.format(1,2,3)	1 2 3
'The value of Pi is {:.2f}'.format(3.14159}	The value of Pi is 3.14

Use the `format()` function to return a formatted string using this syntax:

format([variable or literal value], [format specifier])

Example:

print(format(x, '7.2f')) # where x is a variable holding a number

Result:

7.20

As of Python version 3.6, `f-strings` combine the features of `str.format()` and `format()` in a more compact syntax. With `f-strings`, variables or literals and optional formatting specifiers are embedded in a string using placeholders as:

{[variable name or literal value]:[format specifier]}

Example:

print(f'The value of Pi is {math.pi:7.2f}')

Result:

The value of Pi is 3.14

A **format specifier** has the following general format:

[width][,][.precision][type]

Where,

width	An optional integer indicating the width of the field.
,	An optional separator character for thousands.
.precision	An optional decimal point followed by an integer number of decimal places.
type	One of: f = floating point, d = integer, e = scientific notation, etc.

Sample formatting options for placeholders:

Option	Description
{:w}	Display in a field with width w
{:<w}, {:>w}, {:^w}	Format left, right, or centered in a field of width w
{:*<w}, {:*>w}, {:*^w}	Format left, right, or centered in a field of width w using fill char *
{!s}	Convert the argument to a string
{:.nf}	Display a float with n decimal places
{:+f}, {: f}, {:-f}	Display float with sign as specified
{:,}	Use a comma as a thousands separator
{:%}, {.n%}	Multiply by 100 and express as percentage; optional n decimal places

Full syntax of format specifications:

format_spec	[[fill]align][sign][#][0][width][,][.precision][type]
fill	<any character>
align	"<" | ">" | "=" | "^" (left, right, padding after sign, center)
sign	"+" | "-" | " " (always, negative only, space or -)
width	integer (minimum width)
precision	integer (digits after decimal for floating point)
type	"b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" | "n" | "o" | "s" | "x" | "X" | "%"

Common data types:

s	string (optional; this is the default for string types)
d	decimal integer
f	fixed point (float with default of 6 decimal places)
g	general (attempts to choose an appropriate format; this is default for numeric)
%	percent (multiplies by 100 and displays as fixed with % sign)

Examples:

Format Specification	Result
'{0:f}'.format(3)	3.000000
'{0:0>4d}'.format(1)	0001
format(1.12345, '*>7.3')	***1.12
f'{1.12345:*<7.3}'	1.12***
f'{.2:%}'	20.000000%

f'--{5:*>3}--{5.0:*>3.3}--'	--**5--5.0--