Formatting Output in Python v.3

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

---

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.

You can use the format() function to display numeric values:

format([number], [format specifier])

Example:

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

The format specifier has the following format*:

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

Where,

width	An optional integer indicating the width of the field.
,	An optional comma separator 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

* For a full description of the possible syntax, see the `str.format()` variation below.

An alternative method is to use 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	string.ljust(width)
rjust()	Returns argument right-justified in a string of the width specified	string.rjust(width)
center()	Returns argument centered in a string of the width specified	string.center(width)
zfill()	Pads a numeric value on the left side with zeros if needed to fill the width	string.zfill(width)

The `str.format()` method allows you to format a string using placeholders. Values specified as arguments to the `str.format()` method replace the placeholders using the formatting indicated. 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) 'Cracker {other}'.format(other='Jack') 'Multiple of {0} or {1}'.format(1,2) '{} {} {}'.format(1,2,3)	Number 5 Cracker Jack Multiple of 1 or 2 1 2 3

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

Store format in a variable: `my_fmt = 'Your email addess is {email}'.format`
Then use it elsewhere: `print my_fmt(email='joe@gmail.com')`

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
'{value:*>7.3}'.format(value=1.12345)	***1.12
'{value:*>7.3f}'.format(value=1.12345)	*1.1235
'{0:%}'.format(.2)	20.000000%
'{0:.2%}'.format(.2)	20.00%