Python String Format Cookbook

October 10, 2012

Python v2.7 introduced a new string fomatting method, that is now the default in Python3. I started this string formatting cookbook as a quick reference to help me format numbers and strings. Thanks to other contributors I've expanded the examples over time.

Python 3.6 introduced, formatted string literals, often referred to as f-strings as another method to help format strings. Jump to the new F-strings section below.

Number Formatting

The following table shows various ways to format numbers using Python's str.format(), including examples for both float formatting and integer formatting.

To run examples use: `print("FORMAT".format(NUMBER));`

To get the output of the first example, formatting a float to two decimal places, you would run:

```
print("{:.2f}".format(3.1415926));
```

Number	Format	Output	Description
3.1415926	{:.2f}	3.14	Format float 2 decimal places
3.1415926	{:+.2f}	+3.14	Format float 2 decimal places with sign
-1	{:+.2f}	-1.00	Format float 2 decimal places with sign
2.71828	{:.0f}	3	Format float with no decimal places
5	{:0>2d}	05	Pad number with zeros (left padding, width 2)
5	{:x<4d}	5xxx	Pad number with x's (right padding, width 4)
10	{:x<4d}	10xx	Pad number with x's (right padding, width 4)

1000000	{:,}	1,000,000	Number format with comma separator
0.25	{:.2%}	25.00%	Format percentage
1000000000	{:.2e}	1.00e+09	Exponent notation
13	{:10d}	13	Right aligned (default, width 10)
13	{:<10d}	13	Left aligned (width 10)
13	{:^10d}	13	Center aligned (width 10)

string.format() basics

Here are a couple of examples of basic string substitution, the `{}` is the placeholder for substituted variables. If no format is specified, it will insert and format as a string.

```
s1 = "show me the {}".format("money")
s2 = "hmmm, this is a {} {}".format("tasty", "burger")
```

You can also use the numeric position of the variables and change them in the strings, this gives some flexibility when doing the formatting, if you make a mistake in the order you can easily correct without shuffling all the variables around.

```
s1 = " {0} is better than {1} ".format("emacs", "vim")
s2 = " {1} is better than {0} ".format("emacs", "vim")
```

💡 Tip: You can use `{}` as a variable inside the formatting brackets (h/t Peter Beens for tip). This example uses a precision variable to control how many decimal places to show:

```
pi = 3.1415926
precision = 4
print( "{:.{}f}".format( pi, precision ) )
~~ 3.1415
```

Older % string formatter

Prior to python 2.6, the way to format strings tended to be a bit simpler, though limited by the number of arguments it can receive. These methods still work as of Python 3.3, but there are veiled threats of deprecating them completely though no time table. [PEP-3101]

Formatting a floating point number:

An example comparing the older `%` with `format()` for formatting a float number:

```
pi = 3.14159
print(" pi = %1.2f " % pi)         # old
print(" pi = {:.2f}".format( pi )) # new
```

Multiple Substitution Values

An example comparing variable substitution:

```
s1 = "cats"
s2 = "dogs"
s3 = " %s and %s living together" % (s1, s2)
s4 = " {} and {} living together ".format(s1, s2)
```

Not Enough Arguments

Using the older format method, I would often get the errors:

```
TypeError: not enough arguments for format string
```

or

```
TypeError: not all arguments converted during string formatting
```

because I miscounted my substitution variables, doing something like the following made it easy to miss a variable.

The new Python string formatter you can use numbered parameters so you don't have to count how many you have, at least on half of it.

```
set = " ({0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}) ".format(a,b,c,d,e,f,g)
```

More String Formatting with .format()

The format() function offers a fair amount of additional features and capabilities, here are a few useful tips and tricks using .format()

Named Arguments

You can use the string format as a template engine using named arguments, instead of requiring a strict order.

```
madlib = " The {object} is against the {place}".format(object="chair", place="wall")
~~ The chair is against the wall
```

💡 If you are using Python 3.6 or greater, see the f-strings formatting below for an easier way to create template–they also compute faster interpolations!

Reuse Same Variable Multiple Times

Using % to format requires a strict ordering of variables, the `.format()` method allows you to put them in any order as well as repeating for reuse.

```
str = "Oh {0}, {0}! wherefore art thou {0}?".format("Romeo")
~~ Oh Romeo, Romeo! wherefore art thou Romeo?
```

Convert Values to Different Bases

A surprising use, you can use the string format command to convert numbers to different bases. Use the letter in the formatter to indicate which number base: **d**ecimal, he**x**, **o**ctal, or **b**inary.

This example formats the number `21` in each base:

```
print("{0:d} - {0:x} - {0:o} - {0:b} ".format(21))
~~ 21 - 15 - 25 - 10101
```

Use Format as a Function

You can use `.format` as a function to separate text and formatting from code. For example, at the beginning of your program include all your formats for later use.

```
## defining formats
email_f = "Your email address was {email}".format

## use elsewhere
print(email_f(email="bob@example.com"))
```

Hat tip to earthboundkids who provided this on reddit.

Using format as a function can be used to adjust formating by user preference.

```
## set user preferred format
num_format = "{:,}".format

## use elsewhere
print(num_format(1000000))
```

Internationalization

To use locale specific formatting for numbers, you need to first set the locale, and then use the formating code `n` instead of `d`. For example, using commas or periods to separate thousands in numbers based on the user's locale.

Here is an example, setting locale and formatting a number to display the proper separator:

```
import locale
locale.setlocale(locale.LC_ALL, '')

print("{:n}".format(1000000))
```

Escaping Braces

If you need to use braces when using `str.format()` just double them up:

```
print(" The {} set is often represented as {% raw %}{{0}}{% endraw %}".format("empty"))
~~ The empty set is often represented as {0}
```

Table Formatting Data

Use the width and the left and right justification to align your data into a nice table format. Here's an example to show how to format:

```
# data
starters = [
    [ 'Andre Iguodala', 4, 3, 7 ],
    [ 'Klay Thompson', 5, 0, 21 ],
    [ 'Stephen Curry', 5, 8, 36 ],
    [ 'Draymon Green', 9, 4, 11 ],
    [ 'Andrew Bogut', 3, 0, 2 ],
]

# define format row
```

```
row = "| {player:<16s} | {reb:2d} | {ast:2d} | {pts:2d} |".format

for p in starters:
    print(row(player=p[0], reb=p[1], ast=p[2], pts=p[3]))
```

This would output:

```
| Andre Iguodala   |  4 |  3 |  7 |
| Klay Thompson    |  5 |  0 | 21 |
| Stephen Curry    |  5 |  8 | 36 |
| Draymon Green    |  9 |  4 | 11 |
| Andrew Bogut     |  3 |  0 |  2 |
```

F-Strings

Python 3.6 introduced formatted string literals–yet another method of formatting strings–referred to as f-strings because you start the string specifying an `f` on the outside of the quotes, like so `f"string"`. F strings use a shorter syntax making it easier and more template-like.

```
book = "Lord of the Rings"
author = "J.R.R. Tolkien"
print( f"The {book} was written by {author}" )

# outputs: The Lord of the Rings was written by J.R.R. Tolkien
```

Formatted string literals support running functions inside of the brackets `{ }` this allows you to:

1. Do math with f-strings:

```
print( f"Do math: 3 * 6 = {3 * 6}" )
# outputs: Do math: 3 * 6 = 18
```

2. Call functions with f-strings;

```
verb = "runs"
print( f"The girl {verb.upper()} quickly." )
# outputs: The girl RUNS quickly.
```

You can use f-strings using the three different type of quotation marks in Python, single, double, or triple quotes. The following will all output the same:

```
name = "Fred"
print( f'{name}' )
print( f"{name}" )
print( f"""{name}""" )
```

The one thing you'll want to be careful is mixing the two formats, if you try to use `{}` inside of an f-string, you will get the error:

```
SyntaxError: f-string: empty expression not allowed
```

Each set of brackets used in an f-string requires a value or variable.

Resources

- Python String Library – Standard Library Documentation
- My Python Argparse Cookbook – examples parsing command-line arguments
- My Python Date Formatting – examples working with Python dates.
-

Published on:	October 10, 2012
Category:	Code
Tags:	#python

xkcd graph style in d3 **Python Date Formatting**