Python String Formatting

% operator

```
>>> name = 'Pete'
>>> 'Hello %s' % name
# "Hello Pete"
```

We can use the `%x` format specifier to convert an int value to a string:

```
>>> num = 5
>>> 'I have %x apples' % num
# "I have 5 apples"
```

str.format

Python 3 introduced a new way to do string formatting that was later back-ported to Python 2.7. This makes the syntax for string formatting more regular.

```
>>> name = 'John'
>>> age = 20'

>>> "Hello I'm {}, my age is {}".format(name, age)
# "Hello I'm John, my age is 20"

>>> "Hello I'm {0}, my age is {1}".format(name, age)
# "Hello I'm John, my age is 20"
```

Formatted String Literals or f-strings

If your are using Python 3.6+, string `f-strings` are the recommended way to format strings.

```
>>> name = 'Elizabeth'
>>> f'Hello {name}!'
# 'Hello Elizabeth!'
```

It is even possible to do inline arithmetic with it:

```
>>> a = 5
>>> b = 10
>>> f'Five plus ten is {a + b} and not {2 * (a + b)}.'
# 'Five plus ten is 15 and not 30.'
```

Formatting Digits

Adding thousands separator

```
>>> a = 10000000
>>> f"{a:,}"
# '10,000,000'
```

Rounding

```
>>> a = 3.1415926
>>> f"{a:.2f}"
# '3.14'
```

Showing as Percentage

```
>>> a = 0.816562
>>> f"{a:.2%}"
# '81.66%'
```

Number formatting table

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
4	{:0>2d}	04	Pad number with zeros (left padding, width 2)
4	{:x<4d}	4xxx	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
0.35	{:.2%}	35.00%	Format percentage
1000000000	{:.2e}	1.00e+09	Exponent notation
11	{:11d}	11	Right-aligned (default, width 10)
11	{:<11d}	11	Left-aligned (width 10)
11	{:^11d}	11	Center aligned (width 10)

Template Strings

A simpler and less powerful mechanism, but it is recommended when handling strings generated by users. Due to their reduced complexity, template strings are a safer choice.

```
>>> from string import Template
>>> name = 'Elizabeth'
>>> t = Template('Hey $name!')
>>> t.substitute(name=name)
# 'Hey Elizabeth!'
```