String Formatting

String Formatting

Concatenating strings can sometimes be cumbersome and hard to automate. If you need to include variables and/or values in your string, you may be better off using string formatting. We will use this technique more extensively later on.

There are a few ways to format strings. We will cover one of the ways introduced in Python 3. That is using the `string.format()` method.

This method treats everything contained in curly braces`{}` in the string as a replacement field, everything in and including the braces are replaced with the arguments of format in the output string.

[1]:
```
print('Hello {}, how are you?'.format('world'))
```

```
Hello world, how are you?
```

As you can see above, the blank curly braces were replaced with the string argument `'world'`.

Note that the method does not change the string itself but returns a new string.

You can make multiple replacements at a time if you have a string with multiple replacement fields:

[2]:
```
print('{}, {}, {}'.format(1, 2, 3))
```

```
1, 2, 3
```

Sometimes you will want more control over how the arguments of format are placed into the string. There is a specific syntax for formatting which you can read in the documentation. We will cover a few examples.

Specify Arguments by Position

If you want to specify the order in which the arguments of format are placed into the string, you can put numbers in the replacement fields to reference the positional arguments:

[2]:
```
print('{0}, {2}, {1}'.format(1, 2, 3))
```

```
1, 3, 2
```

Note that this also allows you to repeat elements:

[3]:
```
print('{0}, {2}, {1}, {2}'.format(1, 2, 3))
```

```
1, 3, 2, 3
```

Specify Arguments by Name

You can also specify arguments by name, the arguments must then be presented as keyword arguments:

[35]:
```
print('You can find the point at position ({x}, {y}).'.format(x = 2, y = 6))␣
↪#Arguments with names 'x' and 'y'
```

```
You can find the point at position (2, 6).
```

Specifying Numerical Types and Precision

To put it simply, when formatting numerical arguments the format specifier (to be placed in the replacement field) is of the structure: `[argument_reference]:[width][.precision][type]`

Where - `argument_reference` is the position of or name of the argument. - `width` specifies the minimum width that a replacement will take (look to the docs for alignment options) - For floats `precision` can be seen as the number of decimal places. - `type` specifies what type you want to display the number as. Multiple types exist for both integers and floats, but the most commonly used types are `d` for decimal integer and `f` for fixed point number (which you can use for floats)

Each of these parts of the format specifier are optional.

As a first example, lets display an integer:

[27]:
```
print('{:d}'.format(5))
```

```
5
```

Now, lets see how the width affects the output:

[28]:
```
print('{:d}'.format(5)) #minimum width of 0
print('{:1d}'.format(5)) #minimum width of 1
print('{:2d}'.format(5)) #minimum width of 2
print('{:3d}'.format(5)) #minimum width of 3
```

```
5
5
 5
  5
```

As you can see the first 2 outputs are the same. That is because the output is of length 1.

If you want to display a float to 2 decimal places, specify precision:

[29]:
```
print('{:.2f}'.format(1.232435455))
```

```
1.23
```

If you want to specify the position of the argument, include a reference to the argument position:

[32]:
```
print('{1:.3f}'.format(1.232435455, 5.35362)) #argument position of 1
```

```
5.354
```

Alternative Syntax

Instead of using the `.format()` method on a string, you could alternatively use an f-string for formatting (f prefixed before a string literal).

[2]:
```
subject = 'World'
time = 'today'

f'Hello {subject}! How are you doing {time}?'
```

[2]:
```
'Hello World! How are you doing today?'
```

This simplifies things substantially, but has less range of applicability than `.format()`.