String Formatting

String Declarations

There are a lot of different ways to declare a string in Python, you can use a set of double quotes (“) or a set of single quotes (‘). You can also create a string be enclosing a number of strings in round brackets (generators). To declare a sting over multiple lines all you have to do is enclose the strings in three double quotes (“) or three single quotes (‘).

`a = "Hello"`	String	`Hello`
`b = 'World'`	String	`World`
`c = ("Three " "Strings "` `"Together")`	String	`Three Strings Together`
`d = '''a multiple` `line string'''`	String	`a multiple` `line string`
`e = """More` `Multiple Strings"""`	String	`More` `Multiple Strings`

Counting and Searching for a Character

Python strings have a number of built-in methods, including `count` which counts how often a particular chatacter or substring appears in a string. The `find` method locates the first occurance of a particular chatacter or substring (starting at location zero). The `rfind` (reverse find) method locates the last occurance of a character or substring.

`s = "Hello World"`	
`s.count('o')` # How often does ‘o’ appear in s? `s.count('l')` # How often does ‘l’ appear in s?	2 (‘o’ appears twice) 3 (‘l’ appears three times)
`s.find('o')` # What position is the first ‘o’ at? `s.find('l')` # What position is the first ‘l’ at?	4 (starting at location zero) 2 (starting at location zero)
`s.rfind('o')` # What position is the last ‘o’ at? `s.rfind('l')` # What position is the last ‘l’ at?	7 (starting at location zero) 9 (starting at location zero)

String Manipulation

Other built-in methods include the `split` method to split a string based on a specificed parameter, the `join` method join substrings based on a specificed parameter, the `replace` method replaces characters in a string with others, the `partition` method divides the string into three parts based on the first occurance of a specificed parameter, and finally the `rpartition` method splits on the last occurance of a parameter.

`s = "Hello World, how are you?"`	
`s2 = s.split(' ')`	['Hello', 'World,', 'how', 'are', 'you?']
`s3 = '#'.join(s2)`	Hello#World,#how#are#you?
`s4 = s.replace(' ', '**')`	Hello**World,**how**are**you?
`s5 = s.partition(' ')`	('Hello', ' ', 'World, how are you?')
`s6 = s.rpartition(' ')`	('Hello World, how are', ' ', 'you?')

Python has many other built-in methods, including `center()`, `endswith()`, `isalpha()`, `isdigit()`, `isspace()`, `lower()`, and `upper()`.

String Formatting

The `format` Method

Another built-in method that Python 3 provides is the `format` method. This allows you to add text into a string using curly braces. There are a number of different ways of persenting the arguments to the string.

Unindexed Arguments

If you put empty curly braces into the string with and have an equivalent number of strings in the `format` command, Python will use positional substitution to repleace the first set of braces with the first string in the `format` command, the second braces with with the second string in the `format` command, etc.

```
MyText = "{}, you are currently {}"
print(MyText.format('Damian', 'teaching'))
```

Gives you the following output:

```
Damian, you are currently teaching
```

Indexed Arguments

If you put numbers in the curly braces and have a number of strings in the `format` command, the Python intrepreter will substitute the arguments on the basis of the numbers (starting at zero).

```
MyText = "{0}, you are currently {1}, thanks {0}"
print(MyText.format('Damian', 'teaching'))
```

Gives you the following output:

```
Damian, you are currently teaching, thanks Damian
```

Keywords Arguments

If you put labels in the curly braces and have a number of strings, each associated with one of the labels, in the `format` command, the Python intrepreter will substitute the arguments on the basis of the labels.

```
MyText = "{name}, you are currently {activity}"
print(MyText.format(name="Damian", activity="teaching"))
```

Gives you the following output:

```
Damian, you are currently teaching
```

Mixing Argument Types

You are also allowed to mix together different types of arguments, so in the example below the two unindex arguments are mixed with a keyword arguement. The two unindexed arguments match with the strings "x" and "x+1" in the `format` command which are also unlabelled. And the argument labelled as "Label" matches the string "=":

```
print("{} {Label} {}".format("x", "x + 1", Label = "="))
```

Gives you the following output:

```
x = x + 1
```

This is a sampling of the range of ways you can use string arguments.