String Methods

Let's learn about methods provided by the Python's string object

A string is an object in Python. In fact, everything in Python is an object. However, you're not really ready for that. If you want to know more about how Python is an object oriented programming language, then you'll need to skip to that chapter. In the meantime, it's enough to know that strings have their very own methods built into them. For example, let's say you have the following string:

```
my_string = "This is a string!"
```

Now you want to cause this string to be entirely in uppercase. To do that, all you need to do is call its **upper()** method, like this:

```
my_string = "This is a string!"
print(my_string.upper())

# Or more concisely
print("This is a string!".upper())
```

There are many other string methods. For example, if you wanted everything to be lowercase, you would use the **lower()** method. If you wanted to remove all the leading and trailing white space, you would use **strip()**. To get a list of all the string methods, you can use Python's built-in function **dir**. Let's look at this in action

```
my_string = "This is a string!"
print(dir(my_string))

# [
#   '__add__', '__class__', '__contains__', '__delattr__',
#   ...
#   ...
```

```
#   ...
#  'translate', 'upper', 'zfill'
# ]
```

You can safely ignore the methods that begin and end with double-underscores, such as **__add__**. They are not used in every day Python coding. Focus on the other ones instead. If you'd like to know what one of them does, just ask for **help**. For example, say you want to learn what **capitalize** is for. To find out, you would type

```
my_string = "This is a string!"
print(help(my_string.capitalize))

# Help on built-in function capitalize:
# capitalize(...) method of builtins.str instance
# S.capitalize() -> str
# Return a capitalized version of S, i.e. make the first character
# have upper case and the rest lower case.
```

You have just learned a little bit about a topic called **introspection**. Python allows easy introspection of all its objects, which makes it very easy to use. Basically, introspection allows you to ask Python about itself. In an earlier section, you learned about casting. You may have wondered how to tell what type the variable was (i.e. an int or a string). You can ask Python to tell you that!

```
my_string = "This is a string!"
print(type(my_string)) # <class 'str'>
```

As you can see, the my_string variable is of type str!