Introduction

Python has several data types, each type is designed to handle data in a specific way. In this lesson we're going to look at strings. In Python a string is a sequence of characters which are contained within single ('...') or double ("...") quotes.

In [ ]:
```
# Python strings
print("Hello World!")
print('Hello World!')
print("ABCdef123@£%&")
```

As you can see, no matter whether single or doube quotes are used the output is the same. This flexibility allows us to write strings that use quotes and apostrophes in them. Let's take a look.

In [ ]:
```
# Python string with quotes
print("I told my boss 'I will be late to work tomorrow'.")

# Python string with apostrophes
print("My friend's car broke down on the way to work.")

# Python string with single quotes
print('I dont like my job.')
```

title() method

In [ ]:
```
# Python title() method
name = "Tony Staunton"
print(name)
```

In [ ]:
```
# Python title() method
name = "tony staunton"
print(name.title())
```

In the example above we have used Python's **title()** method. A method is an action that Python can perform on a piece of data. In this example, the **title()** method has performed the action of changing the first lowercase character of each word in the variable **name** to uppercase.

As you can see, the title() method comes after the variable in the print statement. The dot(.) joining the variable and method together tells Python to make the title() method perform its action on the variable **name**.

In Python every method is followed by a set of parentheses. To perform their actions on a variable, methods sometimes need additional information and this information is placed inside the parentheses. The title() method doesn't need any additional information to do its job so the parentheses are empty.

upper() method

The title() method is not the only tool available to the Python programmer for dealing with case in strings. There is also the **upper()** method. Can you guess what it does?

In [ ]:
```
# Python upper() method
name = "tony staunton"
print(name.upper())
```

lower() method

Imagine this sceniro, you have a website that accepts customer names and displays them back on screen when the customer logs in. When your customers are entering their names, their writing things like:

- John Smith
- JOHN SMITH
- John smith

You don't want to keep the capitalization that your customers enter, so before storing customer names in the database you'll convert the strings to lowercase before saving them, in other words, you'll cleanse them.

In [15]:
```
# Python lower() method
name1 = "John Smith"
name2 = "JANE JONES"
name3 = "Tony staunton"

print(name1.lower())
print(name2.lower())
print(name3.lower())
```

```
john smith
jane jones
tony staunton
```

Now when your customers log back in to your website, you can present their names on screen in a clean way.

In [16]:
```
# Python title() method
print(name1.title())
print(name2.title())
print(name3.title())
```

```
John Smith
Jane Jones
Tony Staunton
```

How to use variables in strings

There will be times, lots of them, when you will want to use a variable's value inside of a string. Staying with the customer application example above, you may want to collect a customers name as a first and last name. That way, when you send out your amazing email updates you can say something like "Hey Tony" instead of "Hey Tony Staunton". Later on, if you wanted, you could combine the first name, last name values to display the customers their full name.

In [17]:
```
# Python variables in strings
first_name = "tony"
last_name = "staunton"
full_name = f"{first_name} {last_name}"
print(full_name)
```

```
tony staunton
```

As you can see in this code example, to insert a variable's value into a string, you place the letter **f** immediately before the opening quotation mark. You then place curly braces {} around the name of any variable you want to use in the string. In this example that's, **first_name** and **last_name**. When you run your code, Python will replace each variable with its value and then display it.

These strings are called **f-strings**. The **f** stands for **format**, as Python formats the string by replacing the name of any variable in curly braces with its value.

f-strings were introduced in Python 3.6, so if you are using a version of Python lower than that you will need to use the **format()** method to display strings. For more information check out the Python documentation.

In [18]:
```
# Using the title() method with f-strings
print(f"Welcome back, {first_name.title()}!")
```

```
Welcome back, Tony!
```

In this example, we have combined the variable, **first_name** with the **title()** method. We've also used this combination in a full sentence welcoming our customer back. We can even assign this welcome back message to a variable.

In [19]:
```
# Assigning f-strings to a variable
welcome_message = f"Welcome back, {first_name.title()}!"
print(welcome_message)
```

```
Welcome back, Tony!
```

Imagine, for a moment, the power of this little piece of code. Your application could have hundres of messages, and all you have to use to display them is a print statement with the variable name.

How to add whitespace to strings

In Python, whitespace is any nonprinting character, these include, spaces, tabs, and end of line symbols.

In [20]:
```
# Python tab character
print("Hello")
print("\tWorld!")
```

```
Hello
        World!
```

As you can see in this example, each print statement is displayed on a new line and the character combination of **\t** produces a tab output.

To add a new line without using additional print statements we can do the following:

In [21]:
```
# Python new line character
print("Hello \nWorld!")
```

```
Hello
World!
```

As you can see from this output, the character combination of **\n** used directly before a word forces that word and everything that follows it onto a new line.

We can also combine the **\t** and **\n** characters in a single string:

In [22]:
```
print("Hello \n\tWorld")
```

```
Hello
        World
```

How to strip whitespace

We've seen now to add white space but how about removing it out or stripping it? Firstly, why would you want to remove whitespace, it doesnt do anything to anyone, right?

rstrip() method

Let me ask you this, is the string "Tony Staunton" sames as " Tony Staunton "? In Python this additional white space is significant. Not only will these two strings be saved as different from one another, the additional whitespace takes up extra memory in a database.

Let's go back to the example of a customer application. When a customer is logging in, in the background the application is going to compare the user name entered by the customer to the one saved on the database, this means that "tonystaunton" is not the same as " tonystaunton" and so the log in will fail. Python can help us solve this problem:

In [23]:
```
# Pythons rstrip() method
username = "tonystaunton    "
print(username)
print(username.rstrip())
```

```
tonystaunton
tonystaunton
```

In this example, the removal of whitespace is only temproary. If we print out the **username** variable again it's back:

In [24]:
```
print(username)
```

```
tonystaunton
```

To permanently remove whitespace you have to assign the stripped value to a variable:

In [25]:
```
username = "tony staunton    "
clean_username = username.rstrip()
print(clean_username)
```

```
tony staunton
```

In this example we have created a new variable, **username**, at line 1 and assigned it the value tony staunton with additional whitespace on the right-hand side.

We then created a second new variable, **clean_username**, and assgined it the value of the first variable but before this assignment happens the **rstrip()** method acts on **username** and strips out whitespace on the right-hand side. We are now free to use **clean_username** throughout the rest of our application.

lstrip() method

Just as we can remove whitespace from the right-hand side of a string, we can do the same thing on the left-hand side by using the method **lstrip()**:

In [26]:
```
# Pythons lstrip() method
username = "    tonystaunton"
print(username)
print(username.lstrip())
```

```
    tonystaunton
tonystaunton
```

Again, the removal of this whitespace is only temporary and you will need to assign the modified to string to a new variable to make this change permanent.

strip() method

If you have a string with whitepsace on each side of it then you can use Python's **strip()** method to remove it:

In [27]:
```
username = "    tonystaunton    "
print(username)
print(username.strip())
```

```
    tonystaunton
tonystaunton
```

Syntax errors

As you start to use strings more and more and your program begin to get bigger a common error will be syntax errors. These errors occur when Python doesn't recognize a part of your code. Here's an example"

In [28]:
```
# Python sytnax error
username = "    tonystaunton    '
```

```
  Input In [28]
    username = "    tonystaunton    '
                                    ^
SyntaxError: EOL while scanning string literal
```

This error has occurred becuase the string was opened with double quotes and closed with a single quote. Python was expecting it to close with a double quote.

Like the error we looked at in the previous lesson the error output from Python tries to help us. You can see from the output that the error has occured on line 2, right after the single quote. If you are using a different code editor from Jupyter Notebook then this might help to more easily spot syntax errors.

Exercises

1. Creata a variable that contains a message and print it out to the screen.
2. Create a variable that can contain a customer's name and print a message to that customer.
3. With the customer name variable created above apply the case methods that we used in this lesson, title(), upper(), lower()
4. Print to the screen a string that contains single and double quotes.
5. Create a variable with additional whitespace and using Python's lstrip(), rstrip(), and strip() methods to remove it.