Files

Python uses file objects to interact with external files on your computer. These file objects can be any sort of file you have on your computer, whether it be an audio file, a text file, emails, Excel documents, etc. Note: You will probably need to install certain libraries or modules to interact with those various file types, but they are easily available. (We will cover downloading modules later on in the course).

Python has a built-in open function that allows us to open and play with basic file types. First we will need a file though. We're going to use some IPython magic to create a text file!

IPython Writing a File

This function is specific to jupyter notebooks! Alternatively, quickly create a simple .txt file with sublime text editor.

In [1]:
```
%%writefile test.txt
Hello, this is a quick test file.
```

```
Overwriting test.txt
```

Python Opening a file

Let's being by opening the file test.txt that is located in the same directory as this notebook. For now we will work with files located in the same directory as the notebook or .py script you are using.

It is very easy to get an error on this step:

In [1]:
```
myfile = open('whoops.txt')
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-1-dafe28ee473f> in <module>()
----> 1 myfile = open('whoops.txt')

FileNotFoundError: [Errno 2] No such file or directory: 'whoops.txt'
```

To avoid this error,make sure your .txt file is saved in the same location as your notebook, to check your notebook location, use **pwd**:

In [2]:
```
pwd
```

Out[2]:
```
'C:\\Users\\Marcial\\Pierian-Data-Courses\\Complete-Python-3-Bootcamp\\00-Python Object and Data Structure Basics'
```

Alternatively, to grab files from any location on your computer, simply pass in the entire file path.

For Windows you need to use double \ so python doesn't treat the second \ as an escape character, a file path is in the form:

```
myfile = open("C:\\Users\\YourUserName\\Home\\Folder\\myfile.txt")
```

For MacOS and Linux you use slashes in the opposite direction:

```
myfile = open("/Users/YouUserName/Folder/myfile.txt")
```

In [2]:
```
# Open the text.txt we made earlier
my_file = open('test.txt')
```

In [3]:
```
# We can now read the file
my_file.read()
```

Out[3]:
```
'Hello, this is a quick test file.'
```

In [4]:
```
# But what happens if we try to read it again?
my_file.read()
```

Out[4]:
```
''
```

This happens because you can imagine the reading "cursor" is at the end of the file after having read it. So there is nothing left to read. We can reset the "cursor" like this:

In [5]:
```
# Seek to the start of file (index 0)
my_file.seek(0)
```

Out[5]:
```
0
```

In [6]:
```
# Now read again
my_file.read()
```

Out[6]:
```
'Hello, this is a quick test file.'
```

You can read a file line by line using the readlines method. Use caution with large files, since everything will be held in memory. We will learn how to iterate over large files later in the course.

In [7]:
```
# Readlines returns a list of the lines in the file
my_file.seek(0)
my_file.readlines()
```

Out[7]:
```
['Hello, this is a quick test file.']
```

When you have finished using a file, it is always good practice to close it.

In [8]:
```
my_file.close()
```

Writing to a File

By default, the `open()` function will only allow us to read the file. We need to pass the argument `'w'` to write over the file. For example:

In [9]:
```
# Add a second argument to the function, 'w' which stands for write.
# Passing 'w+' lets us read and write to the file

my_file = open('test.txt','w+')
```

Use caution!

Opening a file with `'w'` or `'w+'` truncates the original, meaning that anything that was in the original file **is deleted**!

In [10]:
```
# Write to the file
my_file.write('This is a new line')
```

Out[10]: 18

In [11]:
```
# Read the file
my_file.seek(0)
my_file.read()
```

Out[11]: 'This is a new line'

In [12]:
```
my_file.close()  # always do this when you're done with a file
```

Appending to a File

Passing the argument `'a'` opens the file and puts the pointer at the end, so anything written is appended. Like `'w+'`, `'a+'` lets us read and write to a file. If the file does not exist, one will be created.

In [13]:
```
my_file = open('test.txt','a+')
my_file.write('\nThis is text being appended to test.txt')
my_file.write('\nAnd another line here.')
```

Out[13]: 23

In [14]:
```
my_file.seek(0)
print(my_file.read())
```

```
This is a new line
This is text being appended to test.txt
And another line here.
```

In [15]:
```
my_file.close()
```

Appending with `%%writefile`

We can do the same thing using IPython cell magic:

In [16]:
```
%%writefile -a test.txt

This is text being appended to test.txt
And another line here.
```

```
Appending to test.txt
```

Add a blank space if you want the first line to begin on its own line, as Jupyter won't recognize escape sequences like `\n`

Iterating through a File

Lets get a quick preview of a for loop by iterating over a text file. First let's make a new text file with some IPython Magic:

In [17]:
```
%%writefile test.txt
First Line
Second Line
```

```
Overwriting test.txt
```

Now we can use a little bit of flow to tell the program to for through every line of the file and do something:

In [18]:
```
for line in open('test.txt'):
    print(line)
```

```
First Line

Second Line
```

Don't worry about fully understanding this yet, for loops are coming up soon. But we'll break down what we did above. We said that for every line in this text file, go ahead and print that line. It's important to note a few things here:

1. We could have called the "line" object anything (see example below).
2. By not calling `.read()` on the file, the whole text file was not stored in memory.
3. Notice the indent on the second line for print. This whitespace is required in Python.

In [19]:
```
# Pertaining to the first point above
for asdf in open('test.txt'):
    print(asdf)
```

```
First Line

Second Line
```

We'll learn a lot more about this later, but up next: Sets and Booleans!