Files in Python!

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.

Python Opening a file

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")
```

Consider a file “test.txt” already made and is in the same directory as of our Notebook(Jupyter). Whenever the .txt file is saved in the same location as your notebook’s you don’t need to pass the whole directory to the open( ). Instead we can just do this…

```
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.

```
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…

```
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**!

To write …

```
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()
```