**Python File Handling**

Python is well known for its abilities to handle/manipulate files. It’s one of the reasons why it is used so much by hobbyists and data scientists. Imagine being able to open up every word file in a folder and automatically sign it with your name in an instance or get the average values from a csv file containing yearly sales data from Amazon. We can write Python scripts to do all of the above for us.

File handling is not the same as importing a module. Importing modules involves us taking in a Python file containing Python code which we can use. File handling involves pulling in the data from a different file type.

Important things to understand about File Handling:

1. **Opening** files, working on files and **closing** files correctly.
2. The file handling **Mode**: read, write, append, or a mixture of some.
3. The **Seek** **Position**: where you are in the file that you’re working on.

**Opening and Closing Files**

To open a file, you need to be aware of where that file is. For us we’ll be keeping our Python script in the same directory as the files.

You will often see tutorials using the **open()** and **close()** functions to interact with a file:

**my\_file =  open("fileName.txt")**

***#*** *do something with file*

**my\_file .close()**

While the above code works there is a problem. If an error occurs between opening and closing the file, the file may not be closed off correctly. This can cause issues because the file takes up space in memory when opened. Also, using the above code it can be easy to forget the **close()** function.

Due to this, it’s recommended to open files using the **with** command. We do not need to manually close the file here when we’re finished and if an error occurs while we’re handling it, it automatically closes it.

***#***  *file name read mode variable name storing file*

**with open('fileName.txt', mode='r', encoding='utf-8') as my\_file:**

***#*** *do something with file here.*

***#*** *we are inside the opened file code block.*

***#*** *file closes automatically when we*

***#*** *exit the code block*

**print( 'fileName.txt is closed here' )**

**Modes**

When you open a file, you must specify what you want to do with it. This is called the **mode** and you’ll notice it in the below code highlighted in yellow.

**with open('fileName.txt', mode='r', encoding='utf-8') as my\_file:**

* If you want to just **read** the contents of a file than the mode **‘r’** for ‘read’ will do. You will not be able to write to the file when you open the file in ‘r’ mode.
* If you want to just **write** to a file than the mode **‘w’** for ‘write’ is appropriate. However you won’t be able to read the file. Also, when you open a file in ‘w’ mode that does not exist, it will automatically create the file for you. If you open an existing file in ‘w’ mode it will automatically overwrite everything in the file.
* If you want to just **append**/write contents to the end of a file without overwriting it then **‘a’** mode will work for you. Append doesn’t allow you to read the file but anything you write to it will be added to the end.

You can use a combination of modes by adding the **+** symbol to the end of the modes but take note of the subtle differences between them. **‘r+’** allows you to read and write and so does **‘w+’** however there are differences.

A picture containing text

Description automatically generated

**Performing on Files**

Once you have a file opened and in the correct mode, we can start doing things with it. The functions that specifically work on files are highlighted in yellow. Since we’re often dealing with text here, string functions are very useful.

**with open('fileName.txt', mode='r+', encoding='utf-8') as f:**

***#*** *files are commonly given a short alias like* **f**

**print(f.name) *#*** *prints ‘fileName.txt’*

**allTxt = f.read() *# allTxt*** *contains entire file content*

**lineList = allTxt.split('\n') *# lineList*** *contains each line as items*

**wordList = allTxt.split() *# wordList*** *contains each word as items*

***#*** *writes the string at end of file because seeker is at end after* **f.read()**

**f.write('\nThis line is at END')**

***#*** *moves seeker to start of file*

**f.seek(0)**

***#*** *writes the string at start of file because seeker is has been set to*

***#*** *the start in the previous line* **f.seek(0)**

**f.write('Overwrites at START\n')**

The result of running that code on fileName.txt is shown below. The original file is on the left. Note due to buffering you may have to close and reopen the file to see the changes.

Text

Description automatically generated

**The Seeker**

The seeker keeps track of where we are in the file. Think of it as the cursor in a text file. In ‘r’ read mode the seeker starts at the start of the file. In ‘a’ append mode the seeker starts at the very end. Certain modes allow you to move the seeker around such as above where **f.seek(0)** was used to set it back at the start since the **f.read()** function had moved it to the end. **f.tell()** returns a number indicating where the seeker is currently positioned. The numbers represent each character in the file hence why 0 is used to set the seeker to the start.

To move the seeker to the end of the file you need to use **f.seek(0, 2)**. The second parameter is optional and can be **0** = start of file, **1** = current position in file or **2** = end of file.

**Notes on Files**

The string **'\n'** is a new line character. You don’t see these in files but they are used to indicate that what follows should start on a new line.

The string **'\t'** is a tab character and denotes where a tab should be.

**Common Files Containing Data**

**CSV** (Comma Separated Values)

Text

Description automatically generated

**TSV** (Tab Separated Values)

*Text

Description automatically generated*