Python File I/O Review

Michael Corley

May 19, 2022

Contents

1 Opening Files for Reading Data:

`python` has the ability to open and close files containing data (usually text data) for reading and writing. The key to opening a file (for reading or writing) is the use of the `with` and `open` commands:

```
with open('my_file.txt') as file_object:
```

Above, the `open('my_file.txt')` command tells `python` to access the file `my_file.txt` (Note that it is given as a string: between ' and '). The `with ... as file_object` associates the newly opened file with the `python` object `file_object` on which `python` can operate. The `:` at the end begins a block of code with instructions on what to do with the data in the file just like beginning an `if`-block, a `for` loop, or a `while` loop.

1.1 Reading All Data at Once

After opening a file an associating it with a `python` object, we can do work on the contents of the file. The easiest thing to do is to read the entire contents of the file as one giant string using the following code:

```
with open('my_file.txt') as file_object:
    contents = file_object.read()
```

Calling the `.read()` method on a file object in `python` takes all of the content of the file and stores it in a string (in this example, the string variable with name `contents`.

IMPORTANT: Recall that your data file must be in the same directory (folder) as your `python` script. If it is not, then you must use the absolute filepath to open the file.

1.2 Reading Data Line-by-Line

We can also read data from a file line by line, rather than all in one fell swoop. This is accomplished by using a nested `for` lkoop within the `open` block:

```
with open('my_file.txt') as file_object:
    for line in file_object:
 print(line.rstrip())
```

In the above snippet, the file is opened and associated with a `python` object, then a `for` loop is started. When a `for` loop is started over the contents of a file, `python` looks for the unseen newline character to tell it when to start a new loop iteration. By using newline characters as restart signals for the loop, `python` can store each line of data individually.

Beware, however, when you use `print(line)`, `python` prints the line including newline characters. Calling `print(line.rstrip())` as above will take away that newline character so that the data is displayed in the same format as it was written and viewed at first.

2 Writing Data To a File

Now, we can talk about how to have data written into a file for permanent storage.

2.1 Writing to an Empty File

To write data to a file that is empty (or possibly doesn't exist), we must again call open:

```
file_name = ‘my_file.txt’

with open(filename, ‘w’) as file_object:
    file_object.write(‘‘I love programming!’’)
```

In the above snippet, the `‘w’` argument of `open` tells `python` that we need write access to the data file. If we wanted to write more than one line, we must be aware that python will put all of our write statements on one line unless we explicitly end them with the newline character: `\n`.

2.2 Appending to an EXISTING FILE:

When you call `open` with 'w', `python` will erase whatever is in the file when you write to it. This is fine if you want to erase the data. If you want to keep the data in the file and not change it, you must append it:

```
file_name = my_file

with open(file_name, ’a’) as file_object:
   file_name.append("Tuesdays and Thursdays are good days")
   file_name.append("Mondays and Wednesdays are not so good.")
#Do Somethinge Here
```

That's all you need (and a bit more) to get working and finished on the bonus assignment for Sideways Shooter. There is more we can say about file I/O in `python`, but we'll save that for another time.