PALs CSC-152

Section 02, Professor Kim, M-W-F

Tony Stone

November 12, 2025

Lesson VI

1 File Handling

1.1 Motivation

My personal experience with file handling has always felt so empty. It is traditional in introductory computer science courses to include sections on file handling; but why? It usually feels like a weird/niche veer to take in the middle of a course. Even worse, the explanation into the existence of the unit is always "so you can handle files" which a. obviously, and b. that is not a good reason! Why on Earth would we *want* to handle files?

Let's consider some applications of file handling! **Data Science** and file handling practically go hand in hand. Essentially all of the data you plan to use in a project will be interacted with through external files. If you are not so interested in data science—perhaps you are more into software development—the ability to process files is key to resolving user requests, and safety concerns. **Web applications** and **web development** are prominent use cases of file handling. Though admittedly, the choice of Python for such a task is debatable, so consider this as a broader conceptual endeavor into file handling, with practices that transcend any specific programming language.

1.2 The Flow of File Handling

The first step to file handling is opening the file. We'll course with a file called 'basicFile.txt':

```
My name is Tony Stone
```

As a general practice, we'll be opening the file, doing some sort of existence check, and closing the file. To open, we use the **open()** function, and store it in a variable. next, we'll use a print statement to check that it's worked, and then we will call the **close()** method to close our file.

```
file = open('basicFile.txt')
print(file)
file.close()
```

Output:

```
<_io.TextIOWrapper name='basicFile.txt' mode='r' encoding='cp1252'>
```

The argument for the open() function is a string literal, of the file path to the file. When the file is in the same folder as the one you are working in, the file is implied. But feel free to further organize your files, and be more explicit with file paths, separating each 'level' with a '/'. That will look something like:

```
'folderName/fileName.txt'
```

Now back to our example, what is all that junk at the bottom? what we've done is opened the file, and stuck it in a variable creating a file **Object**. Objects have defined methods that upon printing them, it displays something useful. hold on, what even is a method? A **method** is a function that is specific to an object. So, our file object has a method that makes printing it meaningful. As previously stated, we use another method, .close() in order to close the file. If you tried to call .close() on a plain string, or other type of object, you would get an error. since .close() is specific to file objects, it is called a method.

A good way to distinguish a method and a function is the preceding '.' for methods. This period is a pointer to the method, think of it like determining the job of a method by 'attaching' it to the object. Function definitions are not object-specific, and thus do not need specific pointers from objects. Please do not get too bogged down in this, just observe the nuances, and be a little more mindful as you place these tools in your toolbox.

1.3 Reading Files

So now we can access, open, and close files, what are we going to do with it? Why don't we read the file. I'll create a folder called that contains our basic file, with a file that has some substance to it, so we can practice file paths along the way!

This is basically the same, but rather than printing the file object, lets use the **.read()** method to get all the stuff in the file, and then print that.

```
file = open('basicFolder/substantFile.txt')
fileContent = file.read()
print(fileContent)
file.close()
```

Output:

```
1 2 3 4 5
Howdy
:D
```

file.read() returns a string, so feel free to do all of your string manipulations to it! I quite enjoy that smiley face, lets just print that. fileContent is our string from file.read(), well go ahead and use the .split() method specifying were splitting at the '**n**' character. This returns a list which is our string divvied up at each line break. We'll index the last list element; our smiley face.

```
file = open('basicFolder/substantFile.txt')
fileContent = file.read()
print(fileContent.split('\n')[-1])
file.close()
```

Output:

```
:D
```

1.4 Writing to Files

Reading files is a very important tool that is integral to all kinds of tasks and workflows. But its only half the story! We want to have a way to document certain things about our code. Perhaps we are collecting data for a data science project, we need a place to store it all. Maybe we are running experiments and want a place to record results in a way more permanent than the console. File writing is the initiatory system for performing such tasks, and worth our exploring.

'afile.txt' will be our target, where we'll be applying some file writing techniques. Note that our file is located in the 'basicFolder' folder.

```
file = open('basicFolder/afile.txt', 'w')
file.write("Hello!")
file.close
```

Output:

```
Hello!
```

The 'w' argument in the function, open(), is one of several arguments that defines the way Python opens the function. 'w' specifically means we are opening the file in write mode. This will override any exist content, and even make a file if it doesn't exist! Go ahead, in the companion materials, feel free to delete 'afile.txt' and re-run the code. It's back!

We can also append to files. Have you ever written anything in one fell swoop? Of course not, any sort of record-keeping is a piecewise task, so we ought to have a way to do this. The 'a' argument allows us to simply append a text string to our document.

```
file = open('basicFolder/afile.txt', 'a')
file.write("Hello again!")
file.close
```

Output:

```
Hello!Hello again!
```

Yikes. let's be careful, adding a space at the start of our append string. We can overwrite all of this with the initial string with the 'w' argument, and then we can append our more precise string.

```
file = open('basicFolder/afile.txt', 'w')
file.write("Hello!")
file.close
```

```

file = open('basicFolder/afile.txt', 'a')
file.write(" Hello again!")
file.close
```

Output:

```
Hello! Hello again!
```

As we work on building our intuition and perception of code, this should feel somewhat off; and you'd be right! This is an extremely inefficient way to write files, we might as well just write them.

We can instead pass a number of strings as a list, so we don't have to keep opening and closing to keep in line with our intentions with the file. When doing this, well call the .writelines() method, and pass our list of strings as an argument. This may be slightly deceiving, as this method doesn't write 'lines' in the way we think of them, i.e., hitting the 'enter' key. It just mean's it will write with the multiple strings contained in the list. We'll use the handy-dandy '\n' to get some nice structure.

```
content = ["Hello! \n",
           "Hello again! \n"
           "Sh-boom"]
file = open('basicFolder/afile.txt', 'w')
file.writelines(content)
file.close()
```

Output:

```
Hello!
Hello again!
Sh-boom.
```

We now have a guaranteed water-tight system for interacting with files!

...

2 Key Terms

- **Data Science**: The field of computer science related to data collection, processing, and analysis.
- **Web Development**: The field of computer science related to the developing, maintaining, and distributing of internet applications.
- **Method**: A function whose definition is specific to the object it is being used on.

3 Availability

- In Lecture:
  - Wednesday: 10:00am - 11:00am
- PAL Sessions:
  - Monday: 2:00pm - 3:00pm
  - Wednesday: 9:00am - 10:00am
  - Wednesday: 2:00pm - 3:00pm

References

[1] Tony Gaddis. *Starting Out with Python*. Pearson, 5th edition, 2021.

[2] GeeksforGeeks. Python Tutorial — Learn Python Programming Language - GeeksforGeeks — geeksforgeeks.org. https://www.geeksforgeeks.org/python/python-programming-language-tutorial/. [Accessed 29-10-2025].

[3] Bradley N. Miller and David L. Ranum. Problem solving with algorithms and data structures using python. https://runestone.academy/ns/books/published/pythonds/index.html, 2014. Interactive online edition — accessed: 2024.

[4] Python Software Foundation. *Python: A dynamic, open source programming language*, 2025. Version 3.x documentation.

[5] W3Schools. W3Schools.com — w3schools.com. https://www.w3schools.com/python/default.asp. [Accessed 29-10-2025].