Opening Data Files

Introduction to Python Programming

# File Paths: Continued

We learned about *absolute* file paths in the last module. In this module, we will learn about *relative* file paths. Absolute file paths always begin at the root directory of the computer, and relative paths can begin at any location and are relative to the location that they start from. For example, consider the file path from the last module:

/Users/williamhenry/Documents/notes.txt

If we started the path from the ‘williamhenry’ directory, the path would be:

Documents/notes.txt

Now, let’s say we started from the ‘Applications’ directory, which is inside the ‘williamhenry’ directory. The path to the Applications directory is:

/Users/williamhenry/Applications/

Note that the ‘Applications’ directory is not part of the file path to the ‘notes.txt’ file. So, how do we get from ‘Applications’ to ‘notes.txt’? The path first needs to move us up one directory to the ‘williamhenry’ directory, and then we can go from there to ‘notes.txt’. The relative path that does this is:

../Documents/notes.txt

This path begins with ‘..’ which means ‘go up one directory.’ The ‘..’ takes us from the ‘Applications’ directory to the ‘williamhenry’ directory. From there, the path takes us to the ‘Documents’ directory and then to the ‘notes.txt’ file.

# Opening Files in Python

Opening files in Python is fairly simple. We open the file, and then we can loop through the lines of data in the file just like it is a list. Here is some example code for opening a file:

|  |
| --- |
| path\_to\_file = '/Users/williamhenry/Documents/my\_data.txt' my\_file = open(path\_to\_file) **for** line **in** my\_file:  print(line) my\_file.close() |

In this code, we first create a variable, ‘path\_to\_file’, that is assigned the file path to the file we want to open. We then open the file with the ‘open’ function and we store the output in the variable ‘my\_file’. The output of the open function is a file object. This file object simply represents the file in our Python code. For example, we can treat the file as a list of lines. You can see this in the example above. In the code, we use a *for* loop to loop through the lines of text in the file. Inside the loop, we are simply printing the lines of data. Finally, we close the file object. This is important! We always need to close a file that we open as this will free up resources and also release any locks the system may have put on the file while open.

# Using Context Managers

In the section above, we learned how to open a file in Python. This is a perfectly valid way to open a file; however, there is another way that is even more ‘Pythonic’ (in other words, it is the preferred method of most Python programmers). This second method uses what is called a *context manager*. Context managers allow blocks of code to contain temporary variables, or objects, that are automatically ‘cleaned up’ when the block of code has completed. Let’s look at a specific example:

|  |
| --- |
| path\_to\_file = '/Users/williamhenry/Documents/my\_data.txt' **with** open(path\_to\_file) **as** my\_file:  **for** line **in** my\_file:  print(line) |

In this example, we open the file in a very different way. We start with the line ‘**with** open(path\_to\_file) **as** my\_file:’. This line essentially means, “Run the following indented code with the variable ‘my\_file’ being equal to the output of ‘open(path\_to\_file)’.” So, within the block of code that is indented after the ‘with’ line, the variable ‘my\_file’ represents the file that has been opened with ‘open(path\_to\_file)’.

Once the indented block completes, the file is automatically closed. Also, if any error occurs within the indented code, the file will automatically close before the code stops due to the error. This is the magic of context managers. There are many applications of content managers, but the most common example, by far, is opening files.

The actual context manager is the file object returned by the open function. The ‘**with**’ keyword is a special keyword that allows you to use context managers, and it can be used with all context managers, not just when opening files.

Overall, the topic of ‘context managers’ is considered a more advanced topic, so we don’t fully cover them here. For now, all you need to know is that when you open a file in Python, you should use the ‘with’ statement. This way, you can be confident that your file will be closed correctly when the code indented in the ‘with’ block is complete.