3.4.2 Open and Read Files Using Python

Now that you know how to bring in the programming tools, or import dependencies, you're ready to practice opening and reading CSV files. There are two ways to read a file in using a programming language: supply a direct path to the file or use an indirect path. Being familiar with both methods will help you avoid setbacks later on during other projects, Tom will review both with you.

During the file handling process, i.e., opening and reading a file, we may come across two different file types: a text file and a binary file. Before we walk through how to open a file let's go over these two file types.

File Types

A **text file** can be opened in a text editor like VS Code, TextEdit on Mac, and Notepad on Windows. A CSV file is a text file. The data in a text file is encoded as text using ASCII (pronounced "askee") or Unicode. A **binary file** contains data that has not been converted to text. Binary files cannot be opened with a text editor.

REWIND

Binary numbers, or base-2 numbers, are numbers that are written with only ones and zeros. In binary, we use the digits 0 and 1 to represent how many times a power of 2 is included in a number.

NOTE

> For more information, go to the **ASCII website (https://ascii.cl/)** and this **website on Unicode characters (https://www.rapidtables.com/code/text/unicode-characters.html)**.

Open a File

You can access a file in a folder on your computer if you know the direct file path. If you do not know the direct file path, but know the folder and filename, you can access the file indirectly.

After providing the file path in our Python script, we will be able to open and read the file. When the program reads the file, it creates a **file object** in the computer's memory, which provides a way for the program to work with that file. In our script, we can use a variable to reference the file object.

The general format for opening a file is, `file_variable = open(filename, mode)`.

Let's break down what each component is doing in the general format.

- `file_variable` is the name of the variable that will reference the file object.

- `filename` is a string specifying the name of the file.
- `mode` is a string specifying the mode for reading or writing the file object. The possible modes are:
  - `"r"`: Open a file to be read.
  - `"w"`: Open a file to write to it. This will overwrite an existing file and create a file if one does not already exist.
  - `"x"`: Open a file for exclusive creation. If the file does not exist, it will not create one.
  - `"a"`: Open a file to append data to an existing file. If a file does not exist, it creates one, if a file has been created the data will be added to the file.
  - `"+"`: Open a file for reading and writing.

Now that we know how to open a file, we need to open our `election_results.csv` file and read the data in the file.

Read Data from a File

Let's go over how to read data from a file using both a direct path and an indirect path.

Direct Path to the File

First, open the CSV file using the direct path method. The direct path to our `election_results.csv` file will be `Resources/election_results.csv`.

Using VS Code, type the following in the `PyPoll.py` file to assign a variable to our file path.

```
# Assign a variable for the file to load and the path.
file_to_load = 'Resources/election_results.csv'
```

When we type `Resources/election_results.csv`, we are telling the computer to get the `election_results.csv` file that is located in the "Resources" folder.

Next, we will open the file, `file_to_load`, with the `open()` function, using the `"r"` mode to read the file. Then, we'll print the filename object. After reading the file, close the file with the `close()` function. In between the opening and closing of the file is where we will read the data and perform our analysis.

Below our file assignment variable, `file_to_load`, add the following code:

```
# Open the election results and read the file.
election_data = open(file_to_load, 'r')

# To do: perform analysis.

# Close the file.
election_data.close()
```

IMPORTANT

> Closing a file disconnects the program from the file. It's important that you close the file after you read a file and write data to a file.
>
> When you read data from a file and it is not closed at the end of the operation, you can lose some of the data. When you write data to a file, the data is not stored in the file at first. It is written to a "buffer" in the computer memory and may be overwritten later if the file is not closed. Once you close the file, the data is stored in the file.

Python has a way to read and write to a file without needing to use the `open()` and `close()` functions every time. We simply replace the `open()` function with the `with` statement.

The `with` statement opens the file and ensures proper acquisition or release of any data without having to close the file, to ensure that the data isn't lost or corrupted.

The format for the `with` statement is the following:

`with open(filename) as file_variable:`

The `file_variable` is used to reference the file object throughout the script.

Let's modify this code, using the `with` statement instead of the `open()` and `close()` functions. We'll print the file variable, `election_data`, to the screen.

Edit your code below the file assignment variable, `file_to_load`, to look like this:

```
# Open the election results and read the file
with open(file_to_load) as election_data:

    # To do: perform analysis.
    print(election_data)
```

The `with` statement ends with a colon, which means we need to indent on the next line, as we did with `if-else` statements and `for` loops.

Save the `PyPoll.py` file and run the file in the VS Code terminal. The output in VS Code will look something like this:

```
<_io.TextIOWrapper name='Resources/election_results.csv' mode='r' encoding='
```

In this output, the `_io.TextIOWrapper` is a Python class that will allow us to read or write data to and from the file when we used the appropriate

methods and attributes. The `name` represents the path of the file object, and the computer tells us that the file is open in "read" mode with `UTF-8` encoding.

Indirect Path to the File

Sometimes we won't know the direct path to the file on our computer, only that it's in a specific folder. Usually, you will know the direct path, but in a real-world setting, you may be given the indirect path to the file by a fellow coworker or your manager.

To access and open a file for which the direct path is unknown, we use the `os` module.

The `os` module allows us to interact with our operating system. We can see all the different attributes and methods that the `os` module uses by importing the module and typing `print(dir(os))` in the Python interpreter.

```
>>> import os
>>> dir(os)
```

The list is quite extensive, as you can see.

Python provides a submodule `os.path` that allows us to access files on different operating systems, like macOS and Windows.

The `os.path` submodule contains several useful functions to make it easier to join a path, as shown by typing `dir(os.path)` in the Python interpreter.

```
['__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__
```

In this output, we can see there is a function called `join`. The `join()` function joins our file path components together when they are provided as separate strings; then, it returns a direct path with the appropriate operating system separator, forward slash for macOS or backward slash for Windows.

REWIND

Different operating systems use different path separators to separate files and folders:

- macOS uses the forward-slash: `/`
- Windows uses the backslash: `\`

To declare a variable for the file to load, connect the `os.path` submodule with the `join()` function, like this: `os.path.join()`. This is called chaining.

Chaining is a programmatic style that is used for making multiple method calls on the same object. This is a common practice that makes code look clean and concise.

Inside the parentheses of the `join()` function, we will add the folder and file to join together. In this case, we'll add the Resources folder and `election_results.csv` separated by a comma, like this:

```
os.path.join("Resources", "election_results.csv")
```

Then, we use a filename variable to reference the path to `election_data.csv`, like this:

```
file_to_load = os.path.join("Resources", "election_results.csv")
```

Let's put all of this to practical use! In the VS Code `PyPoll.py` file, complete the following steps:

1. Import the `csv` and `os` modules.
2. Add the filename variable that references the path to `election_results.csv`.
3. Open the `election_results.csv` using the `with` statement as the filename object, `election_data`.
4. Print the filename object.

Your `PyPoll.py` file should look like this:

```
import csv
import os
# Assign a variable for the file to load and the path.
file_to_load = os.path.join("Resources", "election_results.csv")
# Open the election results and read the file.
with open(file_to_load) as election_data:

    # Print the file object.
     print(election_data)
```

When we run this file in the VS Code terminal, the output is similar to when we used the direct path for the file variable:

```
<open file 'Resources/election_results.csv', mode 'r' at 0x10479c780>
```

IMPORTANT

> You'll notice that we made comments before the code to explain what we were doing. This is a good practice in coding. Not only will this help others reading your code, but it will also help you refresh your memory if you have to revisit your code a few months down the line.

NOTE

> For more information, see the **documentation on file and directory access (https://docs.python.org/3.7/library/os.path.html)**.