Notes

* The general format for opening a file is, file\_variable = open(filename, mode).
* 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.
* 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:
* **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 the open() and close() functions. We'll print the file variable, election\_data, to the screen.
* 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:

**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.
* 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\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', '\_get\_sep', '\_joinrealpath', '\_varprog', '\_varprogb', 'abspath', 'altsep', 'basename', 'commonpath', 'commonprefix', 'curdir', 'defpath', 'devnull', 'dirname', 'exists', 'expanduser', 'expandvars', 'extsep', 'genericpath', 'getatime', 'getctime', 'getmtime', 'getsize', 'isabs', 'isdir', 'isfile', 'islink', 'ismount', 'join', 'lexists', 'normcase', 'normpath', 'os', 'pardir', 'pathsep', 'realpath', 'relpath', 'samefile', 'sameopenfile', 'samestat', 'sep', 'split', 'splitdrive', 'splitext', 'stat', 'supports\_unicode\_filenames', 'sys']
* 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.

o 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")

To write a file to a directory on your computer, perform steps similar to those we followed when we read a file:

1. Create a filename variable to a direct or indirect path where the file is to be located.
2. Use the open() function in the "w" mode to open a file and write data to the file.

Here's how we would write this in Python.

# Create a filename variable to a direct or indirect path to the file.

file\_to\_save = os.path.join("analysis", "election\_analysis.txt")

# Using the open() function with the "w" mode we will write data to the file.

open(file\_to\_save, "w")

When you add this code to the PyPoll.py file and run it in VS Code terminal, we'll see that this results in an error.

Traceback (most recent call last):

File "PyPoll.py", line 24, in <module>

open(file\_to\_save, "w")

IOError: [Errno 2] No such file or directory: 'analysis/election\_analysis.txt'

The error is an IOError, which is an "Input/Output" error, meaning that we used an output directory, 'analysis/election\_analysis.txt', that doesn't exist with the given file path.

The error states that there is no file or directory 'analysis/election\_analysis.txt'. This error occurs because we don't have a folder named "analysis" where the election\_analysis.txt file should be saved.

To correct this error, create an empty folder in the Election\_Analysis folder and name it "analysis." When we execute the file again, we can open the "analysis" folder and see the election\_analysis.txt file in the VS Code sidebar.

Open election\_analysis.txt and you'll see that it's empty. As we perform the election analysis, we'll write data to this file. For now, let's practice adding some simple data to this file and saving it in the "analysis" folder.

In election\_analysis.txt add "Hello World" to the first line by adding the following code to PyPoll.py and running the file in VS Code.

# Create a filename variable to a direct or indirect path to the file.

file\_to\_save = os.path.join("analysis", "election\_analysis.txt")

# Using the with statement open the file as a text file.

outfile = open(file\_to\_save, "w")

# Write some data to the file.

outfile.write("Hello World")

# Close the file

outfile.close()

Here's what's happening in this code:

1. After we create the file\_to\_save variable, we set the open(file\_to\_save, "w") to a filename variable, outfile.
2. Then, we use the filename variable to write "Hello World" to the file using the write() function from the os module.
3. Lastly, we use outfile.close() to close the file.

When we execute this file and open election\_analysis.txt, we see the string Hello World in the first line.

Now that we know how to write data to a file, let's make the code cleaner and more concise. We'll do this by using the with statement instead of using the open() and close() functions.

Your code for writing to a file should look like this:

# Create a filename variable to a direct or indirect path to the file.

file\_to\_save = os.path.join("analysis", "election\_analysis.txt")

# Using the with statement open the file as a text file.

with open(file\_to\_save, "w") as txt\_file:

# Write some data to the file.

txt\_file.write("Hello World")

We can continue to add data to this file using the write() function.

In place of "Hello World,"  add the following counties to the file: "Arapahoe," "Denver," and "Jefferson." This can be done in two ways.

The first way is to add each county with its own write() function on a separate line, like this:

# Write three counties to the file.

txt\_file.write("Arapahoe")

txt\_file.write("Denver")

txt\_file.write("Jefferson")

When we execute the file and open election\_analysis.txt we'll see the names of the three counties, but there is no space between each county name.

To separate "Arapahoe" and "Denver" by a comma and space, we need to add them to the end of the county name in the write() function as follows:

# Write three counties to the file.

txt\_file.write("Arapahoe, ")

txt\_file.write("Denver, ")

txt\_file.write("Jefferson")

The second method is adding all three counties to one line, like this:

# Write three counties to the file.

txt\_file.write("Arapahoe, Denver, Jefferson")

After editing the code and executing one of these two options, open election\_analysis.txt. You'll see that the counties are separated by a comma and a space.

If we want to write each county on a separate line, we need to add the newline escape sequence to the end of each county name. The **newline escape sequence** is the letter "n" preceded by the backward slash like this: \n.

**NOTE**

The newline escape escape sequence will create a newline, like pressing "return" when it is read. Everything after the \n will be on the next line.

Add the newline escape sequence to the end of the first two county names so that the code looks like this:

# Write three counties to the file.

txt\_file.write("Arapahoe\nDenver\nJefferson")

Edit the code and execute the PyPoll.py file. Then open election\_analysis.txt to see that the counties on separate lines.

**SKILL DRILL**

Modify your code so that the output file looks like this:

Congratulations—you now know how to read a CSV file and write to a text file using Python! This is a huge step because accessing data on a file and writing data to a file are common tasks that programmers and analysts perform.

**Tom** tells you that you have been given the green light to read the election results. You know how to open the file, so you can use that code to start. Then, Tom will guide you in reading the data in the election\_results.csv file.

Now it's time to read the election\_results.csv file. As a reminder, the code in our PyPoll.py file should look similar to this code:

# Add our dependencies.

import csv

import os

# Assign a variable to load a file from a path.

file\_to\_load = os.path.join("Resources", "election\_results.csv")

# Assign a variable to save the file to a path.

file\_to\_save = os.path.join("analysis", "election\_analysis.txt")

# Open the election results and read the file.

with open(file\_to\_load) as election\_data:

# To do: read and analyze the data here.

Next, we'll use the reader function to read election\_data.csv.

**IMPORTANT**

Don't forget to add a colon after the with statement and indent on the next line. Forgetting to add a colon will result in a SyntaxError, and not indenting on the next line will result in an IndentationError. In both cases, your code will not run.

**REWIND**

Remember that we found the reader function within the csv module that will read the CSV file.

To PyPoll.py, add the following code where it says # To do: read and analyze the data here. This will allow us to read the CSV file using the csv module with the reader function.

# Read the file object with the reader function.

file\_reader = csv.reader(election\_data)

The variable, file\_reader, is referencing the file object, which is stored in memory. To "pull" the data out of the file object, we can iterate through the file\_reader variable and print each row, including the headers, or column names.

Add the following code to PyPoll.py and run the file in the VS Code terminal.

# Print each row in the CSV file.

for row in file\_reader:

print(row)

Wow—the output was printed quickly! If you blinked, then you might have missed it. So, run it again. If your output for the last 10 lines looks like this, then your code is correct.

While this output was being generated, two things happened:

1. We did not see the headers or columns printed because the output was generated very quickly.
2. Each row in the CSV file was printed out as a list.

In Python, we can retrieve just the headers from the CSV file using a specific method, which we will use a bit later. We can use list indexing to get each element, or ballot ID, county, and candidate, in each row or list.

For our analysis of the election data, we do not need the column headers. Therefore, when we retrieve the data from the CSV file, we will have to skip the first row—the header row.

To skip the header row of the CSV file, use the next() method. This method will skip the first row and return the next item in the list.

Inside the parentheses of the next() method, add the variable file\_reader that is referencing the file object assigned to the CSV file: next(file\_reader).

Running this code will skip the first instance of what is being read. In our case, it is the first row, or header row, of the CSV file. Then we can start iterating through the data starting with the second row.

Just to make sure we are skipping the header row, let's print out the headers for the CSV file. Modify the code as follows and run the file.

# Read the file object with the reader function.

file\_reader = csv.reader(election\_data)

# Print the header row.

headers = next(file\_reader)

print(headers)

At this point, PyPoll.py should look like this:

# Add our dependencies.

import csv

import os

# Assign a variable to load a file from a path.

file\_to\_load = os.path.join("Resources", "election\_results.csv")

# Assign a variable to save the file to a path.

file\_to\_save = os.path.join("analysis", "election\_analysis.txt")

# Open the election results and read the file.

with open(file\_to\_load) as election\_data:

file\_reader = csv.reader(election\_data)

# Read and print the header row.

headers = next(file\_reader)

print(headers)

When you run this code in your VS Code terminal, the output should only show the headers from the CSV file.

['Ballot ID', 'County', 'Candidate']

Now that we have confirmed that we skipped the header row, we can iterate through each row and gather data for our analysis.

**NOTE**

For more information, see the [documentation on using the next() method with the csv module](https://docs.python.org/3.6/library/csv.html#reader-objects)