W05 Learning Activity (1 of 2): Text Files

Most computers permanently store lots of data on devices such as hard drives, solid state drives, and thumb drives. The data that is stored on these devices is organized into files. Just as a human can write words on a paper, a computer can store words and other data in a file. During this lesson, you will learn how to write Python code that reads text from text files.

Concepts

Broadly speaking, there are two types of files: text files and binary files. A *text file* stores words and numbers as human readable text. A *binary file* stores pictures, diagrams, sounds, music, movies, and other media as numbers in a format that is not directly readable by humans.

Text Files

In order to read data from a text file, the file must exist on one of the computer's drives, and your program must do these three things:

1. Open the file for reading text
2. Read from the file, usually one line of text at a time
3. Close the file

The built-in **open** function opens a file for reading or writing. Here is an excerpt from the official documentation for the **open** function:

```
open(filename, mode="rt")
```

> Open a file and return a corresponding file object.
>
> *filename* is the name of the file to be opened.
>
> *mode* is an optional string that specifies the mode in which the file will be opened. It defaults to **"rt"** which means open for reading in text mode. Other common values are **"wt"** for writing a text file (truncating the file if it already exists), and **"at"** for appending to the end of a text file.

Example 1 contains a program that opens a text file named **plants.txt** for reading at line 20. At line 23 there is a **for** loop that reads the text in the file one line at a time and repeats the body of the for loop once **for** each line of text in the file. In the body of the **for** loop at lines 24–29, the code removes surrounding white space, if there is any, from each line of text and then stores each line of text in a list.

```
1 # Example 1
```

```
 2 def main():
 3     # Read the contents of a text file
 4     # named plants.txt into a list.
 5     text_list = read_list("plants.txt")
 6     # Print the entire list.
 7     print(text_list)
 8 def read_list(filename):
 9     """Read the contents of a text file into a list and
10     return the list. Each element in the list will contain
11     one line of text from the text file.
12     Parameter filename: the name of the text file to read
13     Return: a list of strings
14     """
15     # Create an empty list that will store
16     # the lines of text from the text file.
17     text_list = []
18     # Open the text file for reading and store a reference
19     # to the opened file in a variable named text_file.
20     with open(filename, "rt") as text_file:
21         # Read the contents of the text
22         # file one line at a time.
23         for line in text_file:
24             # Remove white space, if there is any,
25             # from the beginning and end of the line.
26             clean_line = line.strip()
27             # Append the clean line of text
28             # onto the end of the list.
29             text_list.append(clean_line)
30     # Return the list that contains the lines of text.
31     return text_list
32 # Call main to start this program.
33 if __name__ == "__main__":
34     main()
```

```
> python example_1.py
['baobab', 'kangaroo paw', 'eucalyptus', 'heliconia', 'tulip',
'chupasangre cactus', 'prickly pear cactus', 'ginkgo biloba']
```

After the body of a **`for`** loop that reads from a file, we can write a call to the **`file.close`** method. However, when calling the **`open`** function, most programmers use a **`with`** block as shown in example 1 at line 20 and nest the **`for`** loop inside the **`with`** block as shown at lines 23–29. When the **`with`** block ends, the computer will automatically close the file, so that the programmer doesn't have to write a call to the **`file.close`** method.

CSV Files

Many computer systems import and export data in CSV files. CSV is an acronym for comma separated values. A *CSV file* is a text file that contains tabular data with each row on a separate line of the file and each cell (column) separated by a comma. The following example shows the contents of a CSV file named **`hymns.csv`** that stores data about religious songs. Notice that the first row of the

file contains column headings, the next four rows contain data about four hymns, and each row contains three columns separated by commas.

```
Title,Author,Composer
O Holy Night,John Dwight,Adolphe Adam
Away in a Manger,Anonymous,William Kirkpatrick
Joy to the World,Isaac Watts,George Handel
With Wondering Awe,Anonymous,Anonymous
```

Python has a standard module named **csv** that includes functionality to read from and write to CSV files. The program in example 2 shows how to open a CSV file and use the **csv** module to read the data and print it to a terminal window. In example 2 at line 6, there is a call to the Python built-in **open** function, which opens the **hymns.csv** file for reading. At line 9, the program creates a **csv.reader** object that will read from the **hymns.csv** file. Within the **for** loop at lines 12 and 13 the **csv.reader** reads and prints each row from the CSV file.

```
 1 # Example 2
 2 import csv
 3 def main():
 4     # Open the CSV file for reading and store a reference
 5     # to the opened file in a variable named csv_file.
 6     with open("hymns.csv", "rt") as csv_file:
 7         # Use the csv module to create a reader object
 8         # that will read from the opened CSV file.
 9         reader = csv.reader(csv_file)
10         # Read the rows in the CSV file one row at a time.
11         # The reader object returns each row as a list.
12         for row_list in reader:
13             print(row_list)
14 # Call main to start this program.
15 if __name__ == "__main__":
16     main()
```

```
> python example_2.py
['Title', 'Author', 'Composer']
['O Holy Night', 'John Dwight', 'Adolphe Adam']
['Away in a Manger', 'Anonymous', 'William Kirkpatrick']
['Joy to the World', 'Isaac Watts', 'George Handel']
['With Wondering Awe', 'Anonymous', 'Anonymous']
```

When a **csv.reader** reads a row from a CSV file, the reader returns the row as a list of strings. The output from example 2 shows that a **csv.reader** returns a list of strings. In the output, notice the five lists of strings, (strings surrounded by square brackets [...]) that were printed by the print statement at line 17. Notice also that the reader reads all the rows from a CSV file, including the first row, which contains column headings.

You might recall that in CSE 110, you wrote a program that reads from a CSV file without using a **csv.reader**. That program split each row of text from the CSV file using the string **split** method. Unfortunately, using the **split** method will not work for all CSV files. Consider the following **hymns.csv** file that contains rows for the hymns "Far, Far Way on Judea's Plains" and "Oh, Come, All

Ye Faithful". Both of these hymns have commas in their titles. If we use the string **split** method to separate the columns in this CSV file, the hymn titles will be split. A **csv.reader** will correctly split rows in all valid CSV files.

```
Title,Author,Composer
"Far, Far Way on Judea's Plains",John Mcfarlane,John Mcfarlane
"Oh, Come, All Ye Faithful",John Wade,John Wade
"Christ the Lord is Risen Today",Charles Wesley,Anonymous
```

Processing Each Row in a CSV File

After reading each row from a CSV file, the **for** loop in the previous example simply prints the row list to a terminal window. Of course, a **for** loop can do much more than simply print each row. Consider the following CSV file named **dentists.csv** that stores data about dental offices. Notice that the first row of the file contains column headings, the next four rows contain data about four dental offices, and each row contains five columns separated by commas.

```
Company Name,Address,Phone Number,Employees,Patients
Eagle Rock Dental Care,556 Trejo Suite C,208-359-2224,7,1205
Apple Tree Dental,33 Winn Drive Suite 2,208-359-1500,10,1520
Rockhouse Dentistry,106 E 1st N,208-356-5600,12,1982
Cornerstone Family Dental,44 S Center Street,208-356-4240,8,1453
```

The program in example 3 processes each row in the **dentists.csv** file to determine which dental office has the most patients per employee. Notice that the first row of the **dentists.csv** file contains column headings. The headings contain no numbers and aren't needed for the calculations, so the program skips the first row by calling the built-in **next** function at line 19.

```
 1 # Example 3
 2 import csv
 3 # Indexes of some of the columns
 4 # in the dentists.csv file.
 5 COMPANY_NAME_INDEX = 0
 6 NUM_EMPS_INDEX = 3
 7 NUM_PATIENTS_INDEX = 4
 8 def main():
 9     # Open a file named dentists.csv and store a reference
10     # to the opened file in a variable named dentists_file.
11     with open("dentists.csv", "rt") as dentists_file:
12         # Use the csv module to create a reader
13         # object that will read from the opened file.
14         reader = csv.reader(dentists_file)
15         # The first row of the CSV file contains column
16         # headings and not data about a dental office,
17         # so this statement skips the first row of the
18         # CSV file.
19         next(reader)
20         running_max = 0
21         most_office = None
22         # Read each row in the CSV file one at a time.
23         # The reader object returns each row as a list.
24         for row_list in reader:
25             # For the current row, retrieve the
```

```
            # values in columns 0, 3, and 4.
            company = row_list[COMPANY_NAME_INDEX]
            num_employees = int(row_list[NUM_EMPS_INDEX])
            num_patients = int(row_list[NUM_PATIENTS_INDEX])
            # Compute the number of patients per
            # employee for the current dental office.
            patients_per_emp = num_patients / num_employees
            # If the current dental office has more
            # patients per employee than the running
            # maximum, assign running_max and most_office
            # to be the current dental office.
            if patients_per_emp > running_max:
                running_max = patients_per_emp
                most_office = company
    # Print the results for the user to see.
    print(f"{most_office} has {running_max:.1f}"
            " patients per employee")
# Call main to start this program.
if __name__ == "__main__":
    main()
```

```
> python example_3.py
Cornerstone Family Dental has 181.6 patients per employee
```

Reading a CSV File into a Compound List

The program in example 3 reads and processes each row in a CSV file. That program needs to access the data in each row once only. If a program needs to access the contents of a CSV file multiple times, the program can read the contents of the file into a compound list and then access the data from the list. The program in example 4 contains a function named **`read_compound_list`** that reads the contents of a CSV file into a compound list.

```
# Example 4
import csv
def main():
    # Read the contents of the dentists.csv file
    # into a compound list.
    dentists_list = read_compound_list("dentists.csv")
    # Print the entire list.
    print(dentists_list)
def read_compound_list(filename):
    """Read the contents of a CSV file into a compound
    list and return the list. Each element in the
    compound list will be a small list that contains
    the values from one row of the CSV file.
    Parameter filename: the name of the CSV file to read
    Return: a list of lists that contain strings
    """
    # Create an empty list that will
    # store the data from the CSV file.
    compound_list = []
```

```
    # Open the CSV file for reading and store a reference
    # to the opened file in a variable named csv_file.
    with open(filename, "rt") as csv_file:
        # Use the csv module to create a reader object
        # that will read from the opened CSV file.
        reader = csv.reader(csv_file)
        # Read the rows in the CSV file one row at a time.
        # The reader object returns each row as a list.
        for row_list in reader:
            # If the current row is not blank,
            # append it to the compound_list.
            if len(row_list) != 0:
                # Append one row from the CSV
                # file to the compound list.
                compound_list.append(row_list)
    # Return the compound list.
    return compound_list
# Call main to start this program.
if __name__ == "__main__":
    main()
```

```
> python example_4.py
[['Company Name', 'Address', 'Phone Number', 'Employees',
'Patients'], ['Eagle Rock Dental Care', '556 Trejo Suite C',
'208-359-2224', '7', '1205'], ['Apple Tree Dental',
'33 Winn Drive Suite 2', '208-359-1500', '10', '1520'],
['Rockhouse Dentistry', '106 E 1st N', '208-356-5600', '12',
'1982'], ['Cornerstone Family Dental', '44 S Center Street',
'208-356-4240', '8', '1453']]
```

Reading a CSV File into a Compound Dictionary

If the values in one of the columns of a CSV file are unique, then a progam can read the contents of a CSV file into a compound dictionary and then use the dictionary to quickly find data. Recall that each item in a dictionary is a key value pair. The values from the unique column in a CSV file will be the keys in the dictionary. The program in example 5 shows how to read the data from a CSV file into a compound dictionary. Notice in example 5, because of lines 6, 10, 44, and 47, that the program uses the dental office phone numbers as the keys in the dictionary.

```
# Example 5
import csv
def main():
    # Index of the phone number column
    # in the dentists.csv file.
    PHONE_INDEX = 2
    # Read the contents of the dentists.csv into a
    # compound dictionary named dentists_dict. Use
    # the phone numbers as the keys in the dictionary.
    dentists_dict = read_dictionary("dentists.csv", PHONE_INDEX)
    # Print the dentists compound dictionary.
    print(dentists_dict)
def read_dictionary(filename, key_column_index):
```

```
    """Read the contents of a CSV file into a compound
    dictionary and return the dictionary.
    Parameters
        filename: the name of the CSV file to read.
        key_column_index: the index of the column
            to use as the keys in the dictionary.
    Return: a compound dictionary that contains
        the contents of the CSV file.
    """
    # Create an empty dictionary that will
    # store the data from the CSV file.
    dictionary = {}
    # Open the CSV file for reading and store a reference
    # to the opened file in a variable named csv_file.
    with open(filename, "rt") as csv_file:
        # Use the csv module to create a reader object
        # that will read from the opened CSV file.
        reader = csv.reader(csv_file)
        # The first row of the CSV file contains column
        # headings and not data, so this statement skips
        # the first row of the CSV file.
        next(reader)
        # Read the rows in the CSV file one row at a time.
        # The reader object returns each row as a list.
        for row_list in reader:
            # If the current row is not blank, add the
            # data from the current to the dictionary.
            if len(row_list) != 0:
                # From the current row, retrieve the data
                # from the column that contains the key.
                key = row_list[key_column_index]
                # Store the data from the current
                # row into the dictionary.
                dictionary[key] = row_list
    # Return the dictionary.
    return dictionary
# Call main to start this program.
if __name__ == "__main__":
    main()
```

```
> python example_5.py
{'208-359-2224': ['Eagle Rock Dental Care', '556 Trejo Suite C', '208-359-2224', 7, 1205],
'208-359-1500': ['Apple Tree Dental', '33 Winn Drive Suite 2', '208-359-1500', 10, 1520],
'208-356-5600': ['Rockhouse Dentistry', '106 E 1st N', '208-356-5600', 12, 1982],
'208-356-4240': ['Cornerstone Family Dental', '44 S Center Street', '208-356-4240', 8, 1453]}
```

Additional Information

The following tutorials contain additional information that you may find helpful. You are not required to read these tutorials.

- Python "for" Loops
- Writing Text Files in Python

Summary

A text file stores words and numbers as human readable text. During this lesson, you are learning how to write Python code to read from text files. To read from a text file, your program must first open the file by calling the built-in **`open`** function. You should write the code to open a file in a Python **`with`** block because the computer will automatically close the file when the **`with`** block ends, and you won't need to remember to write code to close the file.

A CSV file is a text file that contains rows and columns of data. CSV is an acronym that stands for comma separated values. Within each row in a CSV file, the data values are separated by commas. Python includes a standard module named **`csv`** that helps us easily write code to read from CSV files. Sometimes a program simply needs to use the values in a CSV file in calculations, so we write Python code to perform calculations for each row. Other times, we write Python code to read the contents of a CSV file into a compound list or compound dictionary.

Checkpoint

Purpose

Check your understanding of text files and lists by writing a program that reads the contents of a text file into a list and then changes some of the values in the list.

Helpful Documentation

- The Reading Files article explains how to setup VS Code so that your Python program can read from a text file.
- The Text Files section of the preparation content for this lesson contains example code that shows how to read a text file into a list.
- The Lists section of the preparation content for week 04 contains example code that shows how to replace and remove elements in a list.
  - The list.count method counts and returns the number of times a value appears in a list.

Assignment

You must do the following to setup VS Code so that your program can read from a text file:

1. Download the **`provinces.txt`** file and save it in the same folder where you will save your Python program.
2. Using VS Code, open the folder where you saved the **`provinces.txt`** file by doing the following:
   - On a computer running the *Mac OSX* operating system:
     1. Select the **File** menu, then **Open Fldер...**
     2. Find and select the folder where you saved the **`provinces.txt`** file.
     3. Select the **Open** button.
   - On a computer running the *Windows* operating system:

1. Select the File menu, then Open Folder...
2. Find and select the folder where you saved the **`provinces.txt`** file.
3. Select the Select Folder button.

Now that you have correctly setup VS Code so that your program can read the **`provinces.txt`** file, open that file in VS Code and examine it. Notice that the file contains a list of names of Canadian Provinces. Notice also that the file contains "AB" several times which is an abbreviation for Alberta.

Write a Python program named **`provinces.py`** that reads the contents of **`provinces.txt`** into a list and then modifies the list. Your program must do the following:

1. Open the **`provinces.txt`** file for reading.
2. Read the contents of the file into a list where each line of text in the file is stored in a separate element in the list.
3. Print the entire list.
4. Remove the first element from the list.
5. Remove the last element from the list.
6. Replace all occurrences of "AB" in the list with "Alberta".
7. Count the number of elements that are "Alberta" and print that number.

Testing Procedure

Verify that your program works correctly by following each step in this testing procedure:

1. Run your program and ensure that your program's output matches the output below. (Your program may wrap the province names differently.)

```
> python provinces.py
['Alberta', 'Ontario', 'Prince Edward Island', 'Ontario',
'Quebec', 'Saskatchewan', 'AB', 'Nova Scotia', 'Alberta',
'Northwest Territories', 'Saskatchewan', 'Nunavut',
'Nova Scotia', 'Prince Edward Island', 'Alberta',
'Nova Scotia', 'Nova Scotia', 'Prince Edward Island',
'British Columbia', 'Ontario', 'Ontario',
'Newfoundland and Labrador', 'Ontario', 'Ontario',
'Saskatchewan', 'Nova Scotia', 'Prince Edward Island',
'Saskatchewan', 'Ontario', 'Newfoundland and Labrador',
'Ontario', 'British Columbia', 'Manitoba', 'Ontario',
'Alberta', 'Saskatchewan', 'Ontario', 'Yukon', 'Ontario',
'New Brunswick', 'British Columbia', 'Manitoba', 'Yukon',
'British Columbia', 'Manitoba', 'Yukon',
'Newfoundland and Labrador', 'Ontario', 'Yukon', 'Ontario',
'AB', 'Nova Scotia', 'Newfoundland and Labrador', 'Yukon',
'Nunavut', 'Northwest Territories', 'Nunavut', 'Yukon',
'British Columbia', 'Ontario', 'AB', 'Saskatchewan',
'Prince Edward Island', 'Saskatchewan',
'Prince Edward Island', 'Alberta', 'Ontario', 'Alberta',
'Manitoba', 'AB', 'British Columbia', 'Alberta']
Alberta occurs 9 times in the modified list.
```

Sample Solution

When your program is finished, view the sample solution for this assignment to compare your solution to that one. Before looking at the sample solution, you should work to complete this checkpoint program. However, if you have worked on it for at least an hour and are still having problems, feel free to use the sample solution to help you finish your program.

Submission

Continue to the next activity.

Up Next

- W05 Learning Activities: Exceptions

Useful Links:

- Return to: Week Overview | Course Home

>