09 Prepare: 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 25. At line 29 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 31–37, 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
 3  def main():
 4      # Read the contents of a text file
 5      # named plants.txt into a list.
 6      text_list = read_list("plants.txt")
 7
 8      # Print the entire list.
 9      print(text_list)
10
11
12  def read_list(filename):
13      """Read the contents of a text file into a list and
14      return the list. Each element in the list will contain
15      one line of text from the text file.
16
17      Parameter filename: the name of the text file to read
18      Return: a list of strings
19      """
```

```
20      # Create an empty list named text_list.
21      text_list = []
22
23      # Open the text file for reading and store a reference
24      # to the opened file in a variable named text_file.
25      with open(filename, "rt") as text_file:
26
27          # Read the contents of the text
28          # file one line at a time.
29          for line in text_file:
30
31              # Remove white space, if there is any,
32              # from the beginning and end of the line.
33              clean_line = line.strip()
34
35              # Append the clean line of text
36              # onto the end of the list.
37              text_list.append(clean_line)
38
39      # Return the list that contains the lines of text.
40      return text_list
41
42
43  # Call main to start this program.
44  if __name__ == "__main__":
45      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 25 and nest the `for` loop inside the `with` block as shown at lines 29–37. 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 and each cell (column) separated by a comma. The following example shows the contents of a CSV file named `dentists.csv` that stores information about dental offices. Notice that the first row of the file contains column headings, the next four rows contain information 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
```

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 store it in a dictionary. Remember that each item in a dictionary is a key value pair. Notice in example 2, because of lines 11, 16, 56, and 60 that the code is using the dental office phone numbers as the keys in the dictionary.

```
 1  # Example 2
 2
 3  import csv
 4
 5
 6  def main():
 7      # Indexes of some of the columns
 8      # in the dentists.csv file.
 9      COMPANY_NAME_INDEX = 0
10      ADDRESS_INDEX = 1
```

```python
    PHONE_NUMBER_INDEX = 2

    # Read the contents of a CSV file named dentists.csv
    # into a dictionary named dentists. Use the phone
    # numbers as the keys in the dictionary.
    dentists = read_dict("dentists.csv", PHONE_NUMBER_INDEX)

    # Print the dentists dictionary.
    print(dentists)


def read_dict(filename, key_column_index):
    """Read the contents of a CSV file into a 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 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 line of the CSV file contains column
        # headings and not information, so this statement
        # skips the first line 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 in reader:

            # From the current row, retrieve the data
            # from the column that contains the key.
            key = row[key_column_index]

            # Store the data from the current row
            # into the dictionary.
            dictionary[key] = row

    # Return the dictionary.
    return dictionary


# Call main to start this program.
if __name__ == "__main__":
    main()
```

```
> python example_2.py
{'208-359-2224': ['Eagle Rock Dental Care', '556 Trejo Suite C', '…],
'208-359-1500': ['Apple Tree Dental', '33 Winn Drive Suite 2', '20…],
'208-356-5600': ['Rockhouse Dentistry', '106 E 1st N', '208-356-56…],
'208-356-4240': ['Cornerstone Family Dental', '44 S Center Street'…]}
```

The code in example 2 reads the contents of a CSV file and stores the contents in a dictionary. However, it is not always necessary to store the contents of a CSV file in a dictionary. Sometimes a program simply needs to use the values in the CSV file in calculations but doesn't need to store the contents of the CSV file. In such a situation, the program will still use a for loop but will not use a dictionary. The code in example 3 processes each line in the `dentists.csv` file to determine which dental office has the most patients per employee, but it doesn't use a dictionary.

```python
# Example 3

import csv

# Indexes of some of the columns
# in the dentists.csv file.
COMPANY_NAME_INDEX = 0
NUM_EMPLOYEES_INDEX = 3
NUM_PATIENTS_INDEX = 4


def main():
    # Open a file named dentists.csv and store a reference
    # to the opened file in a variable named dentists_file.
    with open("dentists.csv", "rt") as dentists_file:

        # Use the csv module to create a reader
        # object that will read from the opened file.
        reader = csv.reader(dentists_file)

        # The first line of the CSV file contains column headings
        # and not information about a dental office, so this
        # statement skips the first line of the CSV file.
        next(reader)

        running_max = 0
        most_office = None

        # Read each row in the CSV file one at a time.
        # The reader object returns each row as a list.
        for row in reader:

            # For the current row, retrieve the
            # values in columns 0, 3, and 4.
            company = row[COMPANY_NAME_INDEX]
            num_employees = int(row[NUM_EMPLOYEES_INDEX])
            num_patients = int(row[NUM_PATIENTS_INDEX])

            # Compute the number of patients per
            # employee for the current dental office.
            patients_per_employee = 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_employee > running_max:
                running_max = patients_per_employee
                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
```

Tutorials

If any of the concepts or topics in the previous section seem unfamiliar to you, reading these tutorials may help you understand the concepts better.

Python "for" Loops

Reading and Writing CSV 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 we write Python code to read the contents of a CSV file into a dictionary. Other times, we write Python code to process each row in a CSV file but not put the contents in a dictionary.