09 Prepare: Text Files

During this lesson, you will learn about reading data from text files in a Python program.

Concepts

Most computers permanently store lots 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.

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, movies, and other media as numbers in a format that is not directly readable by humans. In this lesson, you will learn how to write Python code that reads data from text files.

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.

After the code that calls the built-in `open` function, we can write a `for` loop that reads the text from the open file. The `for` loop will read from the file one line of text at a time and will repeat the statements in the body of the `for` loop once for each line of text in the file.

After the `for` loop that reads from the file, we can write a call to the `file.close` method. However, most programmers use a `with` block when calling the `open` function and nest the `for` loop inside the `with` block. When the `with` block ends, the computer will automatically close the file, so that the programmer doesn't have to write a call `file.close` method.

Example 1 contains a program that opens a text file for reading at line 25 and then reads the text in the file one line at a time. In the body of the `for` loop at lines 29–37, the code removes surrounding white space 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.
```

```
    text_list = read_list("plants.txt")

    # Print the entire list.
    print(text_list)


def read_list(filename):
    """Read the contents of a text file into a list and
    return the list. Each element in the list will contain
    one line of text from the text file.

    Parameter filename: the name of the text file to read
    Return: a list of strings
    """
    # Create an empty list named text_list.
    text_list = []

    # Open the text file for reading and store a reference
    # to the opened file in a variable named text_file.
    with open(filename, "rt") as text_file:

        # Read the contents of the text
        # file one line at a time.
        for line in text_file:

            # Remove white space, if there is any,
            # from the beginning and end of the line.
            clean_line = line.strip()

            # Append the clean line of text
            # onto the end of the list.
            text_list.append(clean_line)

    # Return the list that contains the lines of text.
    return text_list


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

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 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 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 that the dental office phone number is used as the key, and the other cells are used as the value in each item in the dictionary.

```python
# Example 2

import csv


def main():
    # Indexes of some of the columns
    # in the dentists.csv file.
    COMPANY_NAME_INDEX = 0
    ADDRESS_INDEX = 1
    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 contens 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 a 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 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 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()
```

The previous example 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 next code example 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.

```
# 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()
```

Documentation

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