PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

File Handling in Python

Prof. Sindhu R Pai
PCPS Theory Anchor - 2024
Department of Computer Science and Engineering

Different ways of giving input to the program:

•Using command line arguments.

•Through the keyboard using input() function.

In both the cases the amount of input given would be **minimal** and also be **prone to errors**.

- To store and deal with large data, Files care helpful.

Advantages of Files:

- Data is persistent even after the termination of the program.
- Datasets used can be much larger.
- The data can be input much more quickly and with less chance of error.

In Python, a file operation takes place in the following order:

1. Open a file
2. Read or write (perform operation)
3. Close the file

Opening A File: open() Function

open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)

file is either a text or byte string giving the **name of the file** to be opened if it is in the current working directory, or the **path to a file** if it is in a different directory.

mode is an optional string that **specifies the mode** in which the file is opened. It defaults
to 'r' which means open for reading in text mode. Other modes are ‘a’, ‘a+’, etc.

We don’t need to worry about the other parameters right now.

Opening A File

•A file opening **may not always succeed**. If it fails, it throws an exception.

•A few possible exceptions:

- opening a non-existent file for reading.
- opening a file for writing where directory does not allow us to create files.

•If the required file exists, then a file object is returned.

•OS provides the required resources when the file gets created.

Reading From A File

After we open a file, we use the read() method to read its contents. File can be read in different ways:

- **read()** - Returns the contents of the file as a string
- **readline()** - Reads one line from the file and returns it as a string. The string returned by readline will contain the newline character at the end.
- **readlines()** - Returns a list containing each line in the file as a list item.

Writing To A File

•In order to write it to a file we may use:

- **write()**
- **print()**

Closing a File

•**close()** - We return the resources utilized back to OS by calling a function called close on the open file. This closes the connection with the file and deletes the file object.

Files

Example:

```
file1 = open("test.txt", "r")     # open a file
read_content = file1.read()    # read the file  print(read_content)
file1.close()                  # close the file
```

CSV Files

- CSV -> Comma Separated Values
- As the name suggests, it **stores values separated by commas**
- CSV files are used to **store tabular/structured data in plain text**, with each line typically representing one data record
- **Extension:** .csv

Working With CSV Files

- Process remains the same: open, perform operations, close
- However, to perform operations on a CSV file, we use a different Python module: **the csv module**. This has to be imported into your program when you're working with CSV files.
- The csv module allows you to **create CSV reader and writer objects** that will be used to interact with the CSV file without having to worry about maintaining or dealing with the formatting of CSV files.

Reading From CSV Files

- **csv.reader(csv_file_name):** Takes a CSV file name string as an argument and returns a CSV reader object
- Let's say your CSV reader object is stored in a variable called **csvreader**.
- The **CSV reader object is iterable**; i.e., we can use methods like **next()** and **looping constructs** to access information stored in it.
- **next(csvreader):** Returns the next row in the csvreader object as a list of strings
- You can also use **loops to access each row** in the csvreader object as a list of strings.

Reading From CSV Files - Example

```
import csv
with open('file.csv', newline='') as csvfile:
    csvreader = csv.reader(csvfile)
    header = next(csvreader)     # Skip the header row
    for row in csvreader:        # Process each remaining row in the file
        print(row)
```

Output:

```
['John', '30', 'Engineer']
['Sara', '25', 'Doctor']
['Mike', '28', 'Lawyer']
```

file.csv

```
Name,Age,Occupation
John,30,Engineer
Sara,25,Doctor
Mike,28,Lawyer
```

Reading From CSV Files Into Dictionaries

- **csv.DictReader(csv_file_name):** Returns a CSV DictReader object
- A DictReader object **reads each row of the file as a dictionary** with the **keys of the dictionary being the column headers** of the file
- The DictReader object is also an **iterable** object, which means **you can traverse it the same way** you would a regular reader object

Reading From CSV Files Into Dictionaries - Example

```
import csv
with open('file.csv', newline='') as csvfile:
    csvreader = csv.DictReader(csvfile)
    for row in csvreader:
        print(f"Name: {row['Name']}, Age: {row['Age']}, Occupation:
{row['Occupation']}")
```

Output:

Name: John, Age: 30, Occupation: Engineer
Name: Sara, Age: 25, Occupation: Doctor
Name: Mike, Age: 28, Occupation: Lawyer

file.csv

```
Name,Age,Occupation
John,30,Engineer
Sara,25,Doctor
Mike,28,Lawyer
```

PES UNIVERSITY
CELEBRATING 50 YEARS

Writing To CSV Files

- **csv.writer(csv_file_name):** Takes a CSV file name string as an argument and returns a CSV writer object
- The writer object writes rows of data to a CSV file, **with each row being a list/tuple in Python**
- Let's say your CSV writer object is stored in a variable called **csvwriter**.
- **csvwriter.writerow() –** Takes a single list/tuple and writes it as a row into the CSV file
- **csvwriter.writerows() –** Takes a list/tuples of lists/tuples and writes each of them as a row into the CSV file

Writing To CSV Files - Example

```
import csv
with open('output.csv', 'w', newline='') as csvfile:
    csvwriter = csv.writer(csvfile)
    csvwriter.writerow(['Name', 'Age', 'Occupation'])
    csvwriter.writerows([
        ['John', 30, 'Engineer'],
        ['Sara', 25, 'Doctor'],
        ['Mike', 28, 'Lawyer']
    ])
```

output.csv

```
Name,Age,Occupation
John,30,Engineer
Sara,25,Doctor
Mike,28,Lawyer
```

Writing Into CSV Files Using Dictionaries

- **csv.DictWriter(csv_file_name, fieldnames=fieldnames):** Returns a CSV DictWriter object
- Allows you to write to a CSV file using dictionaries, where **each dictionary becomes a row in the CSV file**
- The **fieldnames parameter defines the headers** in the CSV file
- This allows you to **write data easily with named parameters** instead of relying on positional indexes

Writing Into CSV Files Using Dictionaries

- Let's say your DictWriter object is stored in a variable called **csvDictWriter**
- **csvDictWriter.writeheader() –** Writes the headers using the content of fieldnames
- **csvDictWriter.writerows() –** Writes multiple rows of data; dictionaries provided here must have keys identical to the headers in fieldnames

Writing To CSV Files Using Dictionaries - Example

```
import csv
data = [
    {'Name': 'John', 'Age': 30, 'Occupation': 'Engineer'},
    {'Name': 'Sara', 'Age': 25, 'Occupation': 'Doctor'},
    {'Name': 'Mike', 'Age': 28, 'Occupation': 'Lawyer'}
]
with open('output.csv', 'w', newline='') as csvfile:
    fieldnames = ['Name', 'Age', 'Occupation’]
    csvDictWriter = csv.DictWriter(csvfile, fieldnames=fieldnames)
    csvDictWriter.writeheader()
    csvDictWriter.writerows(data)
```

output.csv

```
Name,Age,Occupation
John,30,Engineer
Sara,25,Doctor
Mike,28,Lawyer
```

THANK YOU

Department of Computer Science and Engineering

Dr. Shylaja S S, Director, CCBD & CDSAML, PESU
Prof. Sindhu R Pai – sindhurpai@pes.edu
Dr. Mohan Kumar A V

Ack: Teaching Assistant – Advaith Sanil Kumar