Chapter 6: File Input/Output (I/O)

Contents

Syllabus Learning Outcomes

2.3 Implementing Algorithms and Data Structures
    2.3.4 Store data in and retrieve data from serial and sequential text files.

---

1. Introduction

File is a named location on a disk to store related information.

- It uses non-volatile memory, e.g. hard disk, to store data permanently.
- A file can be in text or binary format.

Advantages of using a file:

- It is more efficient to read large amounts of data from a file, compared to manual data entry.
- We are able to output the results to a file for ease of future reference.

A file operation takes place in the following order:

- Open a file
- Perform operations
- Close the file

2. Open and Close Files

Python has a built-in function `open(file_path)` to open a file.

- The `open()` function returns a file object, also called a file handler, as it is used to read or modify the file accordingly.

To read data from a file object, use its `read()` method.

To close a file object, use its `close()` method. It flushes any unwritten information and closes the file object, after which no more writing can be done. This prevents accidental access to an open file.

Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the `close()` method to close a file to release the resources.

2.1 Opening a non-existent file

It is important to ensure the file resides in the same folder as your Python notebook or file. Assuming `test.txt` is not in the same folder as the Python file, you will get similar error messages.

```
[ ]  file = open("test.txt")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-1-c1d6406a1bac> in <module>()
----> 1 f = open("test.txt")

FileNotFoundError: [Errno 2] No such file or directory: 'test.txt'
```

2.2 Using the `with` method

Another way of opening a file is using the **`with`** method. This method automatically handles the closing of the file.

```
[ ]  with open('quote.txt') as file:
         print(type(file))

     print(file.closed)
```

```
<class '_io.TextIOWrapper'>
True
```

3. Read from a file

There are 3 main modes when opening a file

- read from the file
- write to the file
- append to the file

The default mode when calling `open()` is to read from the file. We can choose to include the default mode when opening the file, as a reminder to ourselves.

There are 3 methods to read a file.

Operator	How it reads
`read()`	Reads the entire text as a single string
`readline()`	Reads the text line by line
`readlines()`	Read the entire text and returns a set of lines

3.1 Example: `read()`

Make sure you have downloaded `quote.txt` in the same folder as your Python notebook or file.

```
[ ]  with open('quote.txt', 'r') as file: # mode to open with is after the file path
         text = file.read()
         print(text)


     print()

     # for developers we prefer to print repr()
     with open('quote.txt') as file: # the mode 'r' is optional in this case because it is the default mode
         text = file.read()
         print(repr(text)) # prints a representation of the text
         # note the quotation marks and \n
```

```
Give a man a program, frustrate him for a day.
Teach a man to program, frustrate him for a lifetime.

'Give a man a program, frustrate him for a day.\nTeach a man to program, frustrate him for a lifetime.'
```

3.2 Example: `readline()`

```
[ ]  # read the text line by line
     with open('quote.txt', 'r') as file:
         line = file.readline()
         print(line)



     # how do we print all the lines?

     with open('quote.txt', 'r') as file:
         while True: # assume we do not know how many lines there are
             line = file.readline()
             if line: # we have not reached the end of the file
                 print(line.strip())
             else: # we have reached the end of the file
                 break
```

```
Give a man a program, frustrate him for a day.

Give a man a program, frustrate him for a day.
Teach a man to program, frustrate him for a lifetime.
```

3.3 Example: `readlines()`

```
[ ]  # read the text line by line
     with open('quote.txt', 'r') as file:
         lines = file.readlines()
         print(lines)
```

```
['Give a man a program, frustrate him for a day.\n', 'Teach a man to program, frustrate him for a lifetime.']
```

Often, we do not store the files in the same directory as the script or notebook. In this case, we have to make use of the `os` module or `Path` module. For the sake of simplicity, this will not be covered here but you are encouraged to find out how to use the above modules to access files in a different directory. A good starting point is to go to website link 7 in the Referenced Readings section of this document.

4. Writing and Appending to a File

When we want to write to a file, we open the file with the mode `w`.

Things to note when using the write mode.

- If the file does not exist, a new file will be created.
- Warning: If the file exists, it will be overwritten. (All previous content will be removed.)
- `write()` is similar to `print()`, except it does not include newline character (hence the `\n`)
- Data is written as a text string.

When using append mode,

- If the file does not exist, a new file will be created.
- If the file exists, new data will be added to the end of the string.
- As with write mode, use `write()`.
- Data is written as a text string.

4.1 Example: `write()`

Complete the following operations using `with` code block.

- Write `"Alexa, "` to a file `test.txt`. This operator will overwrite any content in the file.
- Append `"Good morning!\n"` to the file `test.txt`.
- Append `"What time is it?"` to the file `test.txt`.
- Read and print out content from the file `test.txt`.

```
[1]  # write mode
     with open('test.txt', 'w') as file:
         file.write("Alexa, ")

     # read mode
     with open('test.txt', 'r') as file:
         print(file.read())
```

```
Alexa,
```

```
#append mode
with open('test.txt', 'a') as f:
    f.write("Good morning!\n" )
    f.write("What time is it?")

# default mode is read
with open('test.txt') as f:
    print(f.read())

# clear text file
with open('test.txt', 'w') as f:
    pass # we are not writng anything, hence this will clear the file

with open('test.txt', 'r') as f:
    print(f.read())
    print("end")
```

```
Alexa, Good morning!
What time is it?

end
```

4.2 Example: `writelines()`

Similar to `readlines()`, `writelines()` will write a list of strings into the target file.

Write the strings in `text_list` into `test.txt`.

```
[ ] text_list = ['Hello', 'World', '\nfrom', '\nVJC']

    with open ('test.txt', 'w') as f:
        f.writelines(text_list)

    with open('test.txt', 'r') as f:
        print(f.read())
```

```
HelloWorld
from
VJC
```

5. Text File & Binary File

By default, `open()` assumes the file is a text file. To work with binary files, e.g. images and sound files, adding `b` to the mode. Examples of binary content include jpg files.

- Use `rb` to read a binary file
- Use `wb` to write binary content to a file

6. CSV File I/O

Comma-Separated Values (CSV) is a common text file format to store tabular data.

- It has a simple structure where each row contains one record
- A record may have multiple attributes delimited by common delimiter such as `,`
- Normally, the first line is the header

Sample CSV data:

```
Name,Email,Phone Number,Address
Bob Smith,bob@example.com,123-456-7890,123 Fake Street
Mike Jones,mike@example.com,098-765-4321,321 Fake Avenue
```

Why Use CSV?

- CSV is a common format for data exchange because it is simple and compact.
- Most relational databases provide tools to import and export CSV files.
- CSV files can be easily opened in Excel.

6.1 Python CSV Module

While we could use the built-in `open()` function to work with CSV files in Python, there is a dedicated `csv` module that makes working with CSV files much easier. It contains following built-in functions:

```
csv.reader
csv.writer
writerow()
writerows()
```

6.2 Example: `csv.reader()`

Make sure you have the file `info.csv` in the same folder as the notebook. Print out all the rows in the file.

```
[ ]  import csv

     with open('info.csv', 'r') as file:
         reader = csv.reader(file)
         for r in reader:
             print(r)
```

```
['id', 'name', 'dob', 'email', 'mobile']
['1', 'Lim Ah Seng', '1/1/1995', 'limahseng@hotmail.com', '12345678']
['2', 'Tan Ah Lian', '31/12/1995', 'tanahlian@yahoo.com', '87654321']
```

6.3 Example: Save result as a list

```
import csv # by right, we only  need to import it once, just in case we want to run this cell on its own.

result = []

with open('info.csv', 'r') as file:
    reader = csv.reader(file)
    for r in reader:
        result.append(r)

print(result)
```

```
[['id', 'name', 'dob', 'email', 'mobile'], ['1', 'Lim Ah Seng', '1/1/1995', 'limahseng@hotmail.com', '12345678'], ['2', 'Tan Ah Lian', '31/12/1995', 'tanahlian@yahoo.com', '87654321']]
```

6.4 Example: Save header and data as two separate lists

```
[ ]  result = []

     with open('info.csv', 'r') as file:
         reader = csv.reader(file)
         header = next(reader) # this is done before the for loop
         for r in reader: # start appending data from where the cursor is at
             result.append(r)

     print(header)
     print()
     print(result)
```

```
['id', 'name', 'dob', 'email', 'mobile']

[['1', 'Lim Ah Seng', '1/1/1995', 'limahseng@hotmail.com', '12345678'], ['2', 'Tan Ah Lian', '31/12/1995', 'tanahlian@yahoo.com', '87654321']]
```

6.5 Using `csv.writer()`

A `csv.writer()` can be used to write a CSV file. The `csv.writer()` function returns a writer object that converts the user's data into a delimited string and write to file using its `writerow()` function.

The newline argument is set to `""` when opening a file which the `csv.writer()` will write each row in a line.

6.6 Example: `writerow()` and `writerows()`

Write the following data to a csv file.

```
Name,Email,Phone Number,Address
Robert,bob@example.com,123-456-7890,123 Fake Street
Michael,mike@example.com,098-765-4321,321 Fake Avenue
```

```
[ ]  header = ('Name','Email','Phone Number','Address')

     data = [
         ('Robert','bob@example.com','123-456-7890','123 Fake Street'),
         ('Michael','mike@example.com','098-765-4321','321 Fake Avenue')
         ]

     import csv

     # write mode
     with open('test2.csv', 'w', newline='') as file: #newline set to '',to remove empty rows
         writer = csv.writer(file)
         writer.writerow(header)

     with open('test2.csv', 'r') as file:
         reader = csv.reader(file)
         for r in reader:
             print(r)

     print()
     # append mode
     with open('test2.csv', 'a', newline='') as file: #newline set to '',to remove empty rows
         writer = csv.writer(file)
         writer.writerows(data)

     with open('test2.csv', 'r') as file:
         reader = csv.reader(file)
         for r in reader:
             print(r)
```

```
['Name', 'Email', 'Phone Number', 'Address']

['Name', 'Email', 'Phone Number', 'Address']
['Robert', 'bob@example.com', '123-456-7890', '123 Fake Street']
['Michael', 'mike@example.com', '098-765-4321', '321 Fake Avenue']
```

7. JSON Processing

JSON is another popular syntax for storing and exchanging data. JSON is text, written with JavaScript object notation. Python denotes JSON data as a dictionary.

Python has a built-in package called `json`, which can be used to work with JSON data. Some useful built-in functions include

- loads
- dumps

7.1 Example: `dumps()`

```
[ ]  import json

     # Python data (can be int, float, string, unicode, list, dictionary, tuple)
     pdata = [{'a': 1, 'c': 3, 'b': 2}]
     print('Python data:', pdata)

     # encode python data as json, resultant is a string
     encoded = json.dumps(pdata)
     print('json data:', repr(encoded))
```

```
Python data: [{'a': 1, 'c': 3, 'b': 2}]
json data: '[{"a": 1, "c": 3, "b": 2}]'
```

7.2 Example: More readable version

```
encoded_sort_indent = json.dumps(pdata, sort_keys=True, indent=2)
print(encoded_sort_indent)
```

```
[
  {
    "a": 1,
    "b": 2,
    "c": 3
  }
]
```

8. Types of files

There is a wide variety of file types. Whatever the file type, the content is stored using a defined binary code that allows the file to be used that it is intended for.

For storing data to be used by a program, there are only 2 defined file types - text file or a binary file. A text file contains data stored according to a defined character code. A binary file stores data in its internal representation and is created using a specific program. For instance, integer values may be stored in 2 bytes in two's complement representation.

The organisation of a binary file is based on the concept of a record. A file contains records and each record contains fields. Each field contains a value.

There are generally 2 types of files that we will be dealing with - serial and sequential. The main difference between the two is that in sequential files, the order of the data is maintained.

Other types of files include index sequential files for large data sets and random files or direct-access files which allow you to access any item without searching through the file from the start.

8.1 Serial Files

Serial files contain records which have no defined order. Specifically, they are stored in chronological order, that is, as each record is received it is stored in the next available storage position. This type of file organisation means that the records are in no particular order and therefore, to retrieve a single record, the whole file needs to be read from beginning to end. An example of a serial file would be a bank's records of transactions.

A text file can be considered a type of serial file but it is different because the file has repeating lines which are defined by end-of-line characters. There is no end-of-record character. This means that a record in a serial file must have a defined format to allow data to be input and output correctly.

Refer to this video for more details: https://www.youtube.com/watch?v=ULHh_3tBhvU

8.2 Sequential Files

Sequential files are serial files whose records are sorted in ascending or descending on a particular key field. Key fields contain values that are unique and sequential but not necessarily consecutive. They are the types of files suited to long term storage of data. Therefore, they can be considered as an alternative to a database but do note that primary keys in a database table are required to be unique but not to be sequential. In a sequential file, a particular record is found sequentially reading the value of the key field until the required value is found.

Note that the physical order of the records on the disk is not necessarily sequential, as most manufacturers support an organisation where certain records (inserted after the file has been set up) are held in a logical sequence but are physically placed into an overflow area. They are no longer physically contiguous with the preceding and following logical records, but they can be retrieved in sequence.

Refer to this video for more details: https://www.youtube.com/watch?v=5czzGLlLjVY

9. File Paths

File paths specify the location of individual files so that we can access these files. There are 2 types of file paths - absolute and relative.

9.1 Absolute Path

Absolute file paths are the exact file location in the computer or server. We usually use them to specify the location of files over the internet and will always include a domain name as part of the link. For example, in CSS, we can denote the location of a file as a such:

```
<img src = "https://www.booble.com/images/bee.png">
```

Other examples:

```
http://website.com/assets/image.jpg
```

Meaning: Go to this domain to access the image

```
//website.com/assets/image.jpg
```

Meaning: The image is uploaded to the above location using http or https protocols

9.2 Relative Path

Relative file path points to the location of the file in the root folder of an individual web project with reference to the current working file. We usually use this in our coding and programming projects.

Refer to the examples below. As you can see, we use relative paths for internal use, provided the image is on the same server.

```
image.jpg
```

Meaning: The image is in the same location as the document e.g. python file which is calling the image.

```
./image.jpg
```

Meaning: Same as above. The image is in the same place as the document calling the image.

```
/assets/image.jpg
```

Meaning: This is similar to the absolute path method except that the protocol and domain name are omitted. It is telling the computer to search for the image starting from the root folder `/`, and then look for it in `assets/`

```
assets/image.jpg
```

Meaning: The assets folder is in the same location as the document. The image is in the assets folder.

```
../assets/image.jpg
```

Meaning: From where the document is, go one folder back `../` and look for the image in the assets folder.

```
../../image.jpg
```

Meaning: From where the document is, go two folders back `../../` and access the image.

```
../../assets/image.jpg
```

Meaning: From where the document is, go two folders back `../../` and then, look into assets to find the image.

Referenced Readings

1. https://computersciencewiki.org/images/b/b1/Programming_companion_Python.pdf
2. https://towardsdatascience.com/getting-start-with-python-i-o-9a28ce635ba4
3. https://realpython.com/working-with-files-in-python/
4. https://realpython.com/read-write-files-python/
5. https://www.w3schools.com/python/python_file_handling.asp
6. https://www.analyticsvidhya.com/blog/2021/08/python-tutorial-working-with-csv-file-for-data-science/

7. https://medium.com/@ageitgey/python-3-quick-tip-the-easy-way-to-deal-with-file-paths-on-windows-mac-and-linux-11a072b58d5f

8. https://discuss.codecademy.com/t/what-does-the-newline-argument-do/463575

9. https://medium.com/@Linda_Ikechukwu/understanding-file-paths-165c07ec5cf0#:~:text=File%20paths%20specify%20the%20location%20of%20individual%20files.,two%20types%20%3A%20Absolute%20and%20Relative.&text=Relative%20file%20paths%20on%20the,to%20the%20current%20working%20file.

10. Langfield, S., & Duddell, D. (2018). *Cambridge International as and A level computer science*. Cambridge University Press.