Chapter 7: File I/O

Files

Files are named locations on disk to store related information. They are used to permanently store data in a non-volatile memory (e.g. hard disk).

Since Random Access Memory (RAM) is volatile (which loses its data when the computer is turned off), we use files for future use of the data by permanently storing them.

When we want to read from or write to a file, we need to open it first. When we are done, it needs to be closed so that the resources that are tied with the file are freed.

Hence, 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 Files in Python

Python has a built-in `open()` function to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

```
f = open("test.txt")    # open file in current directory
f = open("C:/Python38/README.txt")  # specifying full path
```

We can specify the mode while opening a file. In mode, we specify whether we want to read `r` write `w` or append `a` to the file. We can also specify if we want to open the file in text mode or binary mode.
The default is reading in text mode. In this mode, we get strings when reading from the file

On the other hand, binary mode returns bytes and this is the mode to be used when dealing with non-text files like images or executable files.

Mode	Description
`r`	Opens a file for reading. (default)
`w`	Opens a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
`x`	Opens a file for exclusive creation. If the file already exists, the operation fails.
`a`	Opens a file for appending at the end of the file without truncating it. Creates a new file if it does not exist.
`t`	Opens in text mode. (default)
`b`	Opens in binary mode.
`+`	Opens a file for updating (reading and writing)

```
f = open("test.txt")      # equivalent to 'r' or 'rt'
f = open("test.txt",'w')  # write in text mode
f = open("img.bmp",'r+b') # read and write in binary mode
```

Unlike other languages, the character `a` does not imply the number 97 until it is encoded using `ASCII` (or other equivalent encodings).

Moreover, the default encoding is platform dependent. In windows, it is `cp1252` but `utf-8` in Linux.

So, we must not also rely on the default encoding or else our code will behave differently in different platforms.

Hence, when working with files in text mode, it is highly recommended to specify the encoding type.

```
f = open("test.txt", mode='r', encoding='utf-8')
```

Closing Files in Python

When we are done with performing operations on the file, we need to properly close the file.

Closing a file will free up the resources that were tied with the file. It is done using the `close()` method available in Python.
Python has a garbage collector to clean up unreferenced objects but we must not rely on it to close the file.

```
f = open("test.txt", encoding = 'utf-8')
# perform file operations
f.close()
```

This method is not entirely safe. If an exception occurs when we are performing some operation with the file, the code exits without closing the file.

A safer way is to use a try … finally block.

```
try:
   f = open("test.txt", encoding = 'utf-8')
   # perform file operations
finally:
   f.close()
```

This way, we are guaranteeing that the file is properly closed even if an exception is raised that causes program flow to stop. We will discuss this topic in detail later in the course.

The best way to close a file is by using the `with` statement. This ensures that the file is closed when the block inside the `with` statement is exited.
We don't need to explicitly call the `close()` method. It is done internally.

```
with open("test.txt", encoding = 'utf-8') as f:
   # perform file operations
```

Writing to Files in Python

In order to write into a file in Python, we need to open it in write `w`, append `a` or exclusive creation `x` mode.
We need to be careful with the `w` mode, as it will overwrite into the file if it already exists. Due to this, all the previous data is erased.
Writing a string or sequence of bytes (for binary files) is done using the `write()`method. This method returns the number of characters written to the file.

```
with open("test.txt",'w',encoding = 'utf-8') as f:
   f.write("my first file\n")
   f.write("This file\n\n")
   f.write("contains three lines\n")
```

This program will create a new file named `test.txt` in the current directory if it does not exist. If it does exist, it is overwritten.
We must include the newline characters ourselves to distinguish the different lines.

Reading Files in Python

To read a file in Python, we must open the file in reading `r` mode.
There are various methods available for this purpose. We can use the `read(size)`method to read in the `size` number of data. If the `size` parameter is not specified, it reads and returns up to the end of the file.
We can read the `text.txt` file we wrote in the above section in the following way:

```
f = open("test.txt",'r',encoding = 'utf-8')
print(f.read() )   # read the whole file

print(f.read(4) )   # read the first 4 bytes

print(f.read(4) )   # read the next 4 bytes

print(f.read()  )   # read in the rest till end of file

print(f.read())  # further reading returns empty sting
```

We can see that the `read()` method returns a newline as `'\n'`. Once the end of the file is reached, we get an empty string on further reading.
We can change our current file cursor (position) using the `seek()` method. Similarly, the `tell()` method returns our current position (in number of bytes).

```
print(f.tell())   # get the current file position


f.seek(0)   # bring file cursor to initial position
0
```

```
print(f.read())  # read the entire file
```

We can read a file line-by-line using a **for** loop This is both efficient and fast.

```
f = open("test.txt",'r',encoding = 'utf-8')
for line in f:
  print(line, end = '')
```

In this program, the lines in the file itself include a newline character `\n`. So, we use the end parameter of the `print()` function to avoid two newlines when printing.
Alternatively, we can use the `readline()` method to read individual lines of a file. This method reads a file till the newline, including the newline character.

```
f = open("test.txt",'r',encoding = 'utf-8')
print(f.readline())
'This is my first file\n'

print(f.readline())
'This file\n'

print(f.readline())
'contains three lines\n'

f.readline()
```

Lastly, the `readlines()` method returns a list of remaining lines of the entire file. All these reading methods return empty values when the end of file (EOF) is reached.

```
f = open("test.txt",'r',encoding = 'utf-8')
print(f.readlines())
```

Python File Methods

There are various methods available with the file object. Some of them have been used in the above examples.

Here is the complete list of methods in text mode with a brief description:

Method	Description

close()	Closes an opened file. It has no effect if the file is already closed.
detach()	Separates the underlying binary buffer from the `TextIOBase` and returns it.
fileno()	Returns an integer number (file descriptor) of the file.
flush()	Flushes the write buffer of the file stream.
isatty()	Returns `True` if the file stream is interactive.
read(`n`)	Reads at most `n` characters from the file. Reads till end of file if it is negative or `None`.
readable()	Returns `True` if the file stream can be read from.
readline(`n`=-1)	Reads and returns one line from the file. Reads in at most `n` bytes if specified.
readlines(`n`=-1)	Reads and returns a list of lines from the file. Reads in at most `n` bytes/characters if specified.
seek(`offset`,`from`=`SEEK_SET`)	Changes the file position to `offset` bytes, in reference to `from` (start, current, end).
seekable()	Returns `True` if the file stream supports random access.
tell()	Returns the current file location.
truncate(`size`=`None`)	Resizes the file stream to `size` bytes. If `size` is not specified, resizes to current location.
writable()	Returns `True` if the file stream can be written to.
write(`s`)	Writes the string `s` to the file and returns the number of characters written.
writelines(`lines`)	Writes a list of `lines` to the file.

Python Directory and Files Management

In this section, we will learn about file and directory management in Python, i.e. creating a directory, renaming it, listing all directories, and working with them.

Python Directory

If there are a large number of files to handle in our Python program, we can arrange our code within different directories to make things more manageable.
A directory or folder is a collection of files and subdirectories. Python has the `os` module that provides us with many useful methods to work with directories (and files as well).

Get Current Directory

We can get the present working directory using the `getcwd()` method of the `os`module.
This method returns the current working directory in the form of a string.

```
import os

print(os.getcwd())
os.chdir('C:\\Python33').  # Change directory

print(os.listdir())        # List the contents of a directory
os.mkdir('test')           # create a directory
os.rename('test','new_one') # Rename a directory
os.remove('old.txt')        # delete a file
os.rmdir('new_one')         # delete an empty directory
import shutil
shutil.rmtree('test')       # Remove a non-empty directory
```

Working with CSV files

What is a CSV?

CSV stands for “Comma Separated Values.” It is the simplest form of storing data in tabular form as plain text. It is important to know to work with CSV because we mostly rely on CSV data in our day-to-day lives as data scientists.

Structure of CSV:

The csv file contains optionally a header containing comma separated . After the header, each line of the file is a **record**. The values of a record are separated by “comma.”

Reading a CSV

CSV files can be handled in multiple ways in Python.

Using csv.reader

Reading a CSV using Python’s inbuilt module called csv using **csv.reader** object.

Steps to read a CSV file:

Import the csv library

```
import csv
```

Open the CSV file

The .open() method in python is used to open files and return a file object.

```
file = open('Salary_Data.csv')

type(file)
```

The type of file is “**_io.TextIOWrapper**” which is a file object that is returned by the **open()** method.

Use the csv.reader object to read the CSV file

```
csvreader = csv.reader(file)
csv.DictReader(open("dept.csv")) ## If you want to read the file as a dictionary
```

More options available at python documentation https://docs.python.org/3/library/csv.html

Note: If you need to change the delimiter from "," to another character, use the option delimiter="/t"(for tab delimiter) in the reader function

Extract the field names

Create an empty list called header. Use the next() method to obtain the header.

The .next() method returns the current row and moves to the next row.

The first time you run next() it returns the header and the next time you run it returns the first record and so on.

```
header = []
header = next(csvreader)
print(header)
```

Extract the rows/records

Create an empty list called rows and iterate through the csvreader object and append each row to the rows list.

```
rows = []
for row in csvreader:
        rows.append(row)
print(rows)
```

Close the file

.close() method is used to close the opened file. Once it is closed, we cannot perform any operations on it.

```
file.close()
```

Complete Code

```
import csv
file = open("people.csv")
csvreader = csv.reader(file)
header = next(csvreader)
print(header)
rows = []
for row in csvreader:
    rows.append(row)
print(rows)
file.close()
```

Implementing the above code using with() statement:

Syntax: with open(filename, mode) as alias_filename:

Modes:

'r' – to read an existing file,

'w' – to create a new file if the given file doesn't exist and write to it,

'a' – to append to existing file content,

'+' – to create a new file for reading and writing

```
import csv
header=[]
rows = []
with open("people.csv", 'r') as file:
    csvreader = csv.reader(file)
    header = next(csvreader)
    for row in csvreader:
        rows.append(row)
print(header)
print(rows)
```

Note: **Invoking enumerate(csvreader) in the above statement will return the line number and the actual line as a tuple**

Using .readlines()

Now the question is – "Is it possible to fetch the header, rows using only open() and with() statements and without the csv library?" Let's see...

.readlines() method is the answer. It returns all the lines in a file as a list. Each item of the list is a row of our CSV file.

The first row of the file.readlines() is the header and the rest of them are the records.

What if we have a huge dataset with hundreds of features and thousands of records. Would it be possible to handle lists??

Here comes the pandas library into the picture.

Install python 3 pandas in the command line

```
pip3 install pandas
```

```
import pandas as pd
data= pd.read_csv("people.csv")
print(data)
print(data.columns)   # Print the column names
print(data.FirstName)  # Print the column
```

Writing csv files using pandas dataframes

```
import pandas as pd
# Define the header as list
header = ['Name', 'M1 Score', 'M2 Score']

# Define the rows List of Rows
data = [['Alex', 62, 80], ['Brad', 45, 56], ['Joey', 85, 98]]

# Invoke the Dataframe method of pandas with header and data
df = pd.DataFrame(data, columns=header)

#The data parameter takes the records and the columns parameter takes
#the columns/field names.

df.to_csv('Student_data.csv', index=False)

Syntax: DataFrame.to_csv(filename, sep=',', index=False)
**separator is ',' by default.
index=False to remove the index numbers.
```

```

```

Processing Excel files using Pandas

Install xlrd module using pip3

```
pip3 install xlrd
```

```
import pandas as pd
excel_file = 'people.xls'
people = pd.read_excel(excel_file)
print(people.head())
people_sheet2 = pd.read_excel(excel_file, sheet_name=1, index_col=0)
print(people_sheet2)
```

Reading and Writing XML Files in Python

Extensible Markup Language, commonly known as XML is a language designed specifically to be easy to interpret by both humans and computers altogether. The language defines a set of rules used to encode a document in a specific format. In this article, methods have been described to read and write XML files in python.
Note: In general, the process of reading the data from an XML file and analyzing its logical components is known as **Parsing**. Therefore, when we refer to reading a xml file we are referring to **parsing the XML document**.
We have two libraries that could be used for the purpose of xml parsing. They are:

- BeautifulSoup used alongside the lxml xml parser
- Elementtree library.

Using BeautifulSoup alongside with lxml parser

For the purpose of reading and writing the xml file we would be using a Python library named BeautifulSoup. In order to install the library, type the following command into the terminal.

```
Pip3 install beautifulsoup4
```

Beautiful Soup supports the HTML parser included in Python's standard library, but it also supports a number of third-party Python parsers. One is the lxml parser (used for parsing XML/HTML documents). lxml could be installed by running the following command in the command processor of your Operating system:

```
Pip3 install lxml
```

Firstly we will learn how to read from an XML file. We would also parse data stored in it.

Reading Data From an XML File

There are two steps required to parse a xml file:-

- Finding Tags
- Extracting from tags

Example:
XML File used: Copy the content and save it with a file named book.xml

```
<?xml version="1.0"?>

<catalog>

  <book id="bk101">

    <author name='test'>Gambardella, Matthew</author>

    <title>XML Developer's Guide</title>

    <genre>Computer</genre>

    <price>44.95</price>

    <publish_date>2000-10-01</publish_date>

    <description>An in-depth look at creating applications

    with XML.</description>

  </book>

  <book id="bk102">

    <author >Ralls, Kim</author>

    <title>Midnight Rain</title>

    <genre>Fantasy</genre>

    <price>5.95</price>
```

```
    <publish_date>2000-12-16</publish_date>

    <description>A former architect battles corporate zombies,

    an evil sorceress, and her own childhood to become queen

    of the world.</description>

  </book>

  </catalog>
```

Sample Code:

```
from bs4 import BeautifulSoup


# Reading the data inside the xml
# file to a variable under the name
# data
with open('books.xml', 'r') as f:
    data = f.read()

# Passing the stored data inside
# the beautifulsoup parser, storing
# the returned object
Bs_data = BeautifulSoup(data, "xml")

# Finding all instances of tag
# `unique`
book = Bs_data.find_all('book')

print(book)

# Using find() to extract attributes
# of the first instance of the tag
b_name = Bs_data.find('author')

print(b_name)

# Extracting the data stored in a
# specific attribute of the
# `child` tag
value = b_name.get('name')

print(value)
```

Exercises

Problem: Write a Python program to read an entire text file. Implement the reading of a file in a function which accepts the file name as a parameter

```
def file_read(fname):
        txt = open(fname)
        print(txt.read())

file_read('test.txt')
```

Problem: Write a Python program to append text to a file and display the text.
Implement the Writing of a file in a function which accepts the file name as a parameter

```
def file_read(fname):

        with open(fname, "w") as myfile:
                myfile.write("Python Exercises\n")
                myfile.write("Java Exercises")
        txt = open(fname)
        print(txt.read())
file_read('abc.txt')
```

Problem: Write a Python program to read a file line by line and store it into a list.

```
def file_read(fname):

        with open(fname) as f:

                #Content_list is the list that contains the read lines.

                content_list = f.readlines()

                print(content_list)
```

```
file_read(\'test.txt\')
```

Problem: Write a Python program to count the number of lines in a text file.

```
def file_lengthy(fname):
        with open(fname) as f:
                for i, l in enumerate(f):
                        pass
        return i + 1
print("Number of lines in the file: ",file_lengthy("test.txt"))
```

Problem : Write a Python program to read each row from a given csv file and print the 3rd column

Sample input:

```
department_id,department_name,manager_id,location_id
10,Administration,200,1700
20,Marketing,201,1800
30,Purchasing,114,1700
40,Human Resources,203,2400
50,Shipping,121,1500
60,IT,103,1400
70,Public Relations,204,2700
80,Sales,145,2500
90,Executive,100,1700
100,Finance,108,1700
```

```
import csv
with open('dept.csv', newline='') as csvfile:
 data = csv.reader(csvfile)
 for row in data:
   print(row[2])
```

Problem : Write a Python program to read a given CSV file having tab delimiter.

Sample input:

```
country_id country_name    region_id
AR    Argentina  2
AU    Australia  3
BE    Belgium    1
BR    Brazil     2
CA    Canada     2
```

```
CH    Switzerland      1
```

```
import csv
with open(country.csv', newline='') as csvfile:
 data = csv.reader(csvfile, delimiter='\t')
 for row in data:
   print(row)
```

Problem : Write a Python program to read a given CSV file as a dictionary.

Sample input:

```
department_id,department_name,manager_id,location_id
10,Administration,200,1700
20,Marketing,201,1800
30,Purchasing,114,1700
40,Human Resources,203,2400
50,Shipping,121,1500
60,IT,103,1400
70,Public Relations,204,2700
80,Sales,145,2500
90,Executive,100,1700
100,Finance,108,1700
```

```
import csv
data = csv.DictReader(open("dept.csv"))
print("CSV file as a dictionary:\n")
for row in data:
   print(row)
```

Problem : Write a Python program to read specific columns(Dept_id,dept_name) of a given CSV file and print the content of the columns. Use departments data with the DictReader.

```
import csv
with open('departments.csv', newline='') as csvfile:
 data = csv.DictReader(csvfile)
 print("ID Department Name")
 print("---------------------------------")
 for row in data:
   print(row['department_id'], row['department_name'])
```

Problem : Write a Python program that reads each row of a given csv file and skip the header of the file. Also print the number of rows and the field names.

```
import csv
fields = []
rows = []
with open('dept.csv', newline='') as csvfile:
 data = csv.reader(csvfile)
 # Following command skips the first row of the CSV file.
 fields = next(data)
 for row in data:
   print(', '.join(row))
print("\nTotal no. of rows: %d"%(data.line_num))
print('Field names are:')
print(', '.join(field for field in fields))
```