02-Python intermediate

File Handling, File operation in Python

Posted on September 18, 2021
Last updated on February 13, 2023

2. File Handling

As the part of programming requirement, we have to store our data permanently for future purpose. For this requirement we should go for files.

Files are very common permanent storage areas to store our data.

There are 2 types of file

- `Text Files`
  - Usually we can use text files to store character data
- `Binary Files`
  - Usually we can use binary files to store binary data like images,video files, audio files etc..

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

- Open a file
- Read or write (perform operation)
- Close the file

2.1 Various properties of File Object

- open()
  - Open the file
- name
  - Name of opened file
- mode
  - Mode in which the file is opened
- closed
  - True if file is closed else False
- readable()
  - True if file is readable else False
- writable()
  - True if file is writable else False
- close()
  - Close the file

```
f=open("abc.txt",'w')
print("File Name: ",f.name)
print("File Mode: ",f.mode)
print("Is File Readable: ",f.readable())
print("Is File Writable: ",f.writable())
print("Is File Closed : ",f.closed)
f.close()
```

2.2 Allowed Modes

Mode	Description
“r”	Read only. Default mode.
“rb”	Read only in binary format
“r+”	Read and write
“rb+”	Read and write in binary format
“w”	Write only. Overwrites existing file or creates a new file.
“wb”	Write only in binary format. Overwrites existing file or creates a new file.
“w+”	Read and write. Overwrites existing file or creates a new file.

Mode	Description
“wb+”	Read and write in binary format. Overwrites existing file or creates a new file.
“a”	Append to existing file or creates new file.
“ab”	Append to existing file or creates new file in binary format.
“a+”	Read and append. Overwrites existing file or creates a new file.
“ab+”	Read and append in binary format. Overwrites existing file or creates a new file.

2.3 Open and close file

Before performing any operation (like read or write) on the file, first we have to `open that file`. For this we should use Python’s inbuilt function `open()`

After completing our operations on the file, it is highly `recommended to close the file`. For this we have to use `close()` function.

```
# Close manually
file_obj = open(file_name, "<mode>")
file_obj.close()

# File is auto closed
with open("new_test.txt" , "r") as f :
    print()
```

2.4 The with statement

The with statement can be used while `opening a file`. We can use this to group file operation statements within a block.

The advantage of with statement is it will take care closing of file,after completing all operations automatically even in the case of exceptions also, and we are `not required to close explicitly`.

2.5 seek() and tell() methods

tell()

We can use `tell()` method to return current position of the cursor(file pointer) from beginning of the file. The position(index) of first character in files is zero just like string index.

seek()

We can use seek() method to move cursor(file pointer) to specified location

```
# offset represents the number of positions
f.seek(offset, fromwhere)
```

```
# The allowed values for second attribute(from where) are
0 --> From beginning of file(default value)
1 --> From current position
2 --> From end of the file
```

2.6 Writing data into files

- write(str)
- writelines(list of lines)

```
f1 = open("test.txt" ,'w' )
f1.write("Data Science Masters course")

# Data availabe in after after file is closed
f1.close()
```

Write json to file

```
data = {
    "name" : "sudh",
    "mail_id" : "sudh@gmail.com",
    "phone_number" : 91345435,
    "subject" :["data science" , "big data" , "data analytics"]
}

import json

with open("test1.json", "w") as f :
    json.dump(data,f)
```

Write a CSV file

```
data = [["name" , "email_id" , "number"],
       ["sudh" , "sudh@gmail.com" , 92342342],
        ["krish" , "krish@gmail.com" , 9324324242]
       ]

import csv

with open("test3.csv" , 'w') as f :
    w = csv.writer(f)
    for i in data:
        w.writerow(i)
```

Write binary

```
with open("test4.bin", 'wb') as f :
    f.write(b"\x01\x02\x03")
```

```
f1=open("rossum.jpg","rb")
f2=open("newpic.jpg","wb")
bytes=f1.read()
f2.write(bytes)
f1.close()
f2.close()
```

2.7 Reading Data from files

- read()

    - To read total data from the file
- read(n)
    - To read ‘n’ characters from the file
- readline()
    - To read only one line
- readlines()
    - To read all lines into a list

Read whole file at once

```
data = open("test.txt" , "r")
data.read()
# Data Science Masters course

data.read()
# ''
# Empty coz, courses reached end of file

# reset cursor
data.seek(0)
data.read()
# Data Science Masters course

data.close()
```

Read file line by line

```
data1 = open("test.txt", 'r')
for i in data1:
    print(i)
data1.close()
```

Read a json file

```
with open("test1.json" , 'r') as f :
    data1 = json.load(f)

print(data1)
print(data1['subject'][1])
```

Read a CSV file

```
with open("test3.csv" , 'r') as f :
    read = csv.reader(f)
    for i in read:
        print(i)
```

Read binary

```
with open("test4.bin" , "rb") as f :
    print(f.read())
```

2.8 Buffer Read and Write file

A buffer `holds a chunk of data` from the operating system's file stream until it is used upon which more data comes in, which is similar to video buffering.

Buffering is useful when you don't know the size of the file you are working with if the file `size is greater than computer memory` then the processing unit will not function properly. The buffer size tells how much data can be held at a time until it is used. io.DEFAULT_BUFFER_SIZE can tell the default buffer size of your platform.

```
import io

# write
with open("test.txt", "wb") as f :
    file = io.BufferedWriter(f)
    file.write(b"Data Science Masters course")
    file.write(b"this is my second line that i am trying to write")
    file.flush()

# read
with open("test.txt" , "rb") as f :
    file = io.BufferedReader(f)
    data = file.read(10000)
    print(data)
```

2.9 Zipping and Unzipping Files

It is very common requirement to zip and unzip files. To perform zip and unzip operations, Python contains one in-bulit module zip file. This module contains a class : ZipFile

The main advantages are:

1. To improve `memory utilization`
2. We can `reduce transport time`
3. We can improve `performance.`

```
from zipfile import *
f=ZipFile("files.zip",'w',ZIP_DEFLATED)
f.write("file1.txt")
f.write("file2.txt")
f.write("file3.txt")
f.close()
```

```
from zipfile import *
f=ZipFile("files.zip",'r',ZIP_STORED)
names=f.namelist()
for name in names:
print( "File Name: ",name)
print("The Content of this file is:")
f1=open(name,'r')
print(f1.read())
print()
```

.

.

.

.

.