Python File Handling

Till now, we were taking the input from the console and writing it back to the console to interact with the user.

Sometimes, it is not enough to only display the data on the console. The data to be displayed may be very large, and only a limited amount of data can be displayed on the console since the memory is volatile, it is impossible to recover the programmatically generated data again and again.

The file handling plays an important role when the data needs to be stored permanently into the file. A file is a named location on disk to store related information. We can access the stored information (non-volatile) after the program termination.

The file-handling implementation is slightly lengthy or complicated in the other programming language, but it is easier and shorter in Python.

In Python, files are treated in two modes as text or binary. The file may be in the text or binary format, and each line of a file is ended with the special character.

Hence, a file operation can be done in the following order.

- Open a file
- Read or write - Performing operation
- Close the file

Opening a file

Python provides an **open()** function that accepts two arguments, file name and access mode in which the file is accessed. The function returns a file object which can be used to perform various operations like reading, writing, etc.

Syntax:

1. file object = open(<file-name>, <access-mode>, <buffering>)

The files can be accessed using various modes like read, write, or append. The following are the details about the access mode to open a file.

SN	Access mode	Description
1	r	It opens the file to read-only mode. The file pointer exists at the beginning. The file is by default open in this mode if no access mode is passed.
2	rb	It opens the file to read-only in binary format. The file pointer exists at the beginning of the file.
3	r+	It opens the file to read and write both. The file pointer exists at the beginning of the file.
4	rb+	It opens the file to read and write both in binary format. The file pointer exists at the beginning of the file.
5	w	It opens the file to write only. It overwrites the file if previously exists or creates a new one if no file exists with the same name. The file pointer exists at the beginning of the file.
6	wb	It opens the file to write only in binary format. It overwrites the file if it exists previously or creates a new one if no file exists. The file pointer exists at the beginning of the file.
7	w+	It opens the file to write and read both. It is different from r+ in the sense that it overwrites the previous file if one exists whereas r+ doesn't overwrite the previously written file. It creates a new file if no file exists. The file pointer exists at the beginning of the file.
8	wb+	It opens the file to write and read both in binary format. The file pointer exists at the beginning of the file.

9	a	It opens the file in the append mode. The file pointer exists at the end of the previously written file if exists any. It creates a new file if no file exists with the same name.
10	ab	It opens the file in the append mode in binary format. The pointer exists at the end of the previously written file. It creates a new file in binary format if no file exists with the same name.
11	a+	It opens a file to append and read both. The file pointer remains at the end of the file if a file exists. It creates a new file if no file exists with the same name.
12	ab+	It opens a file to append and read both in binary format. The file pointer remains at the end of the file.

Let's look at the simple example to open a file named "file.txt" (stored in the same directory) in read mode and printing its content on the console.

Example

```
1. #opens the file file.txt in read mode
2. fileptr = open("file.txt","r")
3.
4. if fileptr:
5.     print("file is opened successfully")
```

Output:

```
<class '_io.TextIOWrapper'>
file is opened successfully
```

In the above code, we have passed **filename** as a first argument and opened file in read mode as we mentioned **r** as the second argument. The **fileptr** holds the file object and if the file is opened successfully, it will execute the print statement

The close() method

Once all the operations are done on the file, we must close it through our Python script using the **close()** method. Any unwritten information gets destroyed once the **close()** method is called on a file object.

We can perform any operation on the file externally using the file system which is the currently opened in Python; hence it is good practice to close the file once all the operations are done.

The syntax to use the **close()** method is given below.

Syntax

```
fileobject.close()
```

Consider the following example.

```
# opens the file file.txt in read mode
fileptr = open("file.txt","r")

if fileptr:
    print("file is opened successfully")

#closes the opened file
fileptr.close()
```

After closing the file, we cannot perform any operation in the file. The file needs to be properly closed. If any exception occurs while performing some operations in the file then the program terminates without closing the file.

We should use the following method to overcome such type of problem.

```
try:
   fileptr = open("file.txt")
   # perform file operations
finally:
   fileptr.close()
```

The with statement

The **with** statement was introduced in python 2.5. The with statement is useful in the case of manipulating the files. It is used in the scenario where a pair of statements is to be executed with a block of code in between.

The syntax to open a file using with the statement is given below.

```
1. with open(<file name>, <access mode>) as <file-pointer>:
2.     #statement suite
```

The advantage of using with statement is that it provides the guarantee to close the file regardless of how the nested block exits.

It is always suggestible to use the **with** statement in the case of files because, if the break, return, or exception occurs in the nested block of code then it automatically closes the file, we don't need to write the **close()** function. It doesn't let the file to corrupt.

Consider the following example.

Example

```
1. with open("file.txt",'r') as f:
2.     content = f.read();
3.     print(content)
```

Writing the file

To write some text to a file, we need to open the file using the open method with one of the following access modes.

w: It will overwrite the file if any file exists. The file pointer is at the beginning of the file.

a: It will append the existing file. The file pointer is at the end of the file. It creates a new file if no file exists.

Consider the following example.

Example

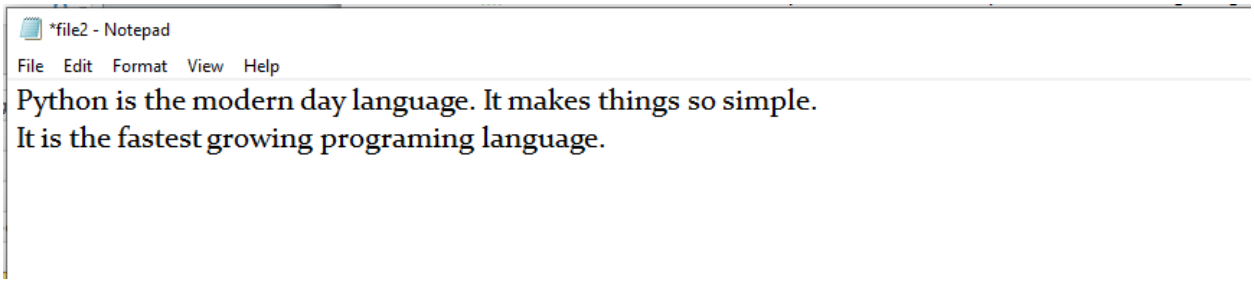
```
# open the file.txt in append mode. Create a new file if no such file exists.
fileptr = open("file2.txt", "w")

# appending the content to the file
fileptr.write('''Python is the modern day language. It makes things so simple.
It is the fastest-growing programing language''')

# closing the opened the file
fileptr.close()
```

Output:

File2.txt

```
Python is the modern-day language. It makes things so simple. It is the fastest
growing programming language.
```

Snapshot of the file2.txt



We have opened the file in **w** mode. The **file1.txt** file doesn't exist, it created a new file and we have written the content in the file using the **write()** function.

Example 2

```
#open the file.txt in write mode.
fileptr = open("file2.txt","a")

#overwriting the content of the file
```

```
4. fileptr.write(" Python has an easy syntax and user-friendly interaction.")
5.
6. #closing the opened file
7. fileptr.close()
```

Output:

```
Python is the modern day language. It makes things so simple.
It is the fastest growing programing language Python has an easy syntax and
user-friendly interaction.
```

Snapshot of the file2.txt

We can see that the content of the file is modified. We have opened the file in **a** mode and it appended the content in the existing **file2.txt**.

To read a file using the Python script, the Python provides the **read()** method. The **read()** method reads a string from the file. It can read the data in the text as well as a binary format.

The syntax of the **read()** method is given below.

Syntax:

fileobj.read(<count>)

Here, the count is the number of bytes to be read from the file starting from the beginning of the file. If the count is not specified, then it may read the content of the file until the end.

Consider the following example.

Example

```
#open the file.txt in read mode. causes error if no such file exists.
fileptr = open("file2.txt","r")
#stores all the data of the file into the variable content
content = fileptr.read(10)
# prints the type of the data stored in the file
print(type(content))
#prints the content of the file
print(content)
#closes the opened file
fileptr.close()
```

Output:

```
<class 'str'>
Python is
```

In the above code, we have read the content of **file2.txt** by using the **read()** function. We have passed count value as ten which means it will read the first ten characters from the file.

If we use the following line, then it will print all content of the file.

```
content = fileptr.read()
print(content)
```

Output:

```
Python is the modern-day language. It makes things so simple.
It is the fastest-growing programing language Python has easy an syntax and
user-friendly interaction.
```

Read file through for loop

We can read the file using for loop. Consider the following example.

```
#open the file.txt in read mode. causes an error if no such file exists.
```

```
2. fileptr = open("file2.txt","r");
3. #running a for loop
4. for i in fileptr:
5.     print(i) # i contains each line of the file
```

Output:

```
Python is the modern day language.

It makes things so simple.

Python has easy syntax and user-friendly interaction.
```

Read Lines of the file

Python facilitates to read the file line by line by using a function **readline()** method. The **readline()** method reads the lines of the file from the beginning, i.e., if we use the readline() method two times, then we can get the first two lines of the file.

Consider the following example which contains a function **readline()** that reads the first line of our file **"file2.txt"** containing three lines. Consider the following example.

Example 1: Reading lines using readline() function

```
1. #open the file.txt in read mode. causes error if no such file exists.
2. fileptr = open("file2.txt","r");
3. #stores all the data of the file into the variable content
4. content = fileptr.readline()
5. content1 = fileptr.readline()
6. #prints the content of the file
7. print(content)
8. print(content1)
9. #closes the opened file
10. fileptr.close()
```

Output:

```
Python is the modern day language.
```

```
It makes things so simple.
```

We called the **readline()** function two times that's why it read two lines from the file.

Python provides also the **readlines()** method which is used for the reading lines. It returns the list of the lines till the end of **file(EOF)** is reached.

Example 2: Reading Lines Using readlines() function

```
#open the file.txt in read mode. causes error if no such file exists.
fileptr = open("file2.txt","r");

#stores all the data of the file into the variable content
content = fileptr.readlines()

#prints the content of the file
print(content)

#closes the opened file
fileptr.close()
```

Output:

```
['Python is the modern day language.\n', 'It makes things so simple.\n',
'Python has easy syntax and user-friendly interaction.']
```

Creating a new file

The new file can be created by using one of the following access modes with the function open().

x: it creates a new file with the specified name. It causes an error a file exists with the same name.

a: It creates a new file with the specified name if no such file exists. It appends the content to the file if the file already exists with the specified name.

w: It creates a new file with the specified name if no such file exists. It overwrites the existing file.

Consider the following example.

Example 1

```
#open the file.txt in read mode. causes error if no such file exists.
fileptr = open("file2.txt","x")
print(fileptr)
if fileptr:
    print("File created successfully")
```

Output:

```
<_io.TextIOWrapper name='file2.txt' mode='x' encoding='cp1252'>
File created successfully
```

File Pointer positions

Python provides the tell() method which is used to print the byte number at which the file pointer currently exists. Consider the following example.

```
# open the file file2.txt in read mode
fileptr = open("file2.txt","r")

#initially the filepointer is at 0
print("The filepointer is at byte :",fileptr.tell())

#reading the content of the file
content = fileptr.read();

#after the read operation file pointer modifies. tell() returns the location of the fileptr.

print("After reading, the filepointer is at:",fileptr.tell())
```

Output:

```
The filepointer is at byte : 0
After reading, the filepointer is at: 117
```

Modifying file pointer position

In real-world applications, sometimes we need to change the file pointer location externally since we may need to read or write the content at various locations.

For this purpose, the Python provides us the seek() method which enables us to modify the file pointer position externally.

The syntax to use the seek() method is given below.

Syntax:

1. <file-ptr>.seek(offset[, **from**)

The seek() method accepts two parameters:

Syntax: *f.seek(offset, from_what), where f is file pointer*
Parameters:
Offset: *Number of positions to move forward*
from_what: *It defines point of reference.*
Returns: *Return the new absolute position.*

The reference point is selected by the **from_what** argument. It accepts three values:

- **0:** sets the reference point at the beginning of the file
- **1:** sets the reference point at the current file position
- **2:** sets the reference point at the end of the file

By default from_what argument is set to 0.
Note: Reference point at current position / end of file cannot be set in text mode except when offset is equal to 0.
Example 1: Let's suppose we have to read a file named "GfG.txt" which contains the following text:

```
"Code is like humor. When you have to explain it, it's bad."
```

- Python3

```
# Python program to demonstrate

# seek() method




# Opening "GfG.txt" text file

f = open("GfG.txt", "r")



# Second parameter is by default 0

# sets Reference point to twentieth

# index position from the beginning

f.seek(20)



# prints current position

print(f.tell())



print(f.readline())

f.close()
```

Output:

```
20
```

```
When you have to explain it, it’s bad.
```

Example 2: Seek() function with negative offset only works when file is opened in binary mode. Let’s suppose the binary file contains the following text.

```
b'Code is like humor. When you have to explain it, its bad.'
```

- Python3

```
# Python code to demonstrate
# use of seek() function

# Opening "GfG.txt" text file
# in binary mode
f = open("data.txt", "rb")
 # sets Reference point to tenth
# position to the left from end
f.seek(-10, 2)
 # prints current position
print(f.tell())

# Converting binary to string and
# printing
print(f.readline().decode('utf-8'))
 f.close()
```

Output:

```
47
, its bad.
```

Consider the following example.

Example

```
# open the file file2.txt in read mode
fileptr = open("file2.txt","r")

#initially the filepointer is at 0
print("The filepointer is at byte :",fileptr.tell())

#changing the file pointer location to 10.
```

```
8. fileptr.seek(10);
9.
10. #tell() returns the location of the fileptr.
11. print("After reading, the filepointer is at:",fileptr.tell())
```

Output:

```
The filepointer is at byte : 0
After reading, the filepointer is at: 10
```

Python OS module

Renaming the file

The Python **os** module enables interaction with the operating system. The os module provides the functions that are involved in file processing operations like renaming, deleting, etc. It provides us the rename() method to rename the specified file to a new name. The syntax to use the **rename()** method is given below.

Syntax:

```
1. rename(current-name, new-name)
```

The first argument is the current file name and the second argument is the modified name. We can change the file name bypassing these two arguments.

Example 1:

```
1. import os
2.
```

```
3. #rename file2.txt to file3.txt
4. os.rename("file2.txt","file3.txt")
```

Output:

The above code renamed current **file2.txt** to **file3.txt**

Removing the file

The os module provides the **remove()** method which is used to remove the specified file. The syntax to use the **remove()** method is given below.

```
1. remove(file-name)
```

Example 1

```
1. import os;
2. #deleting the file named file3.txt
3. os.remove("file3.txt")
```

Creating the new directory

The **mkdir()** method is used to create the directories in the current working directory. The syntax to create the new directory is given below.

Syntax:

1. mkdir(directory name)

Example 1

```
import os

#creating a new directory with the name new
os.mkdir("new")
```

The getcwd() method

This method returns the current working directory.

The syntax to use the getcwd() method is given below.

Syntax

```
os.getcwd()
```

Example

```
import os
os.getcwd()
```

Output:

```
'C:\\Users\\DEVANSH SHARMA'
```

Changing the current working directory

The chdir() method is used to change the current working directory to a specified directory.

The syntax to use the chdir() method is given below.

Syntax

```
chdir("new-directory")
```

Example

```
import os
# Changing current directory with the new directiory
os.chdir("C:\\Users\\DEVANSH SHARMA\\Documents")
#It will display the current working directory
os.getcwd()
```

Output:

```
'C:\\Users\\DEVANSH SHARMA\\Documents'
```

Deleting directory

The rmdir() method is used to delete the specified directory.

The syntax to use the rmdir() method is given below.

Syntax

1. os.rmdir(directory name)

Example 1

```
import os
#removing the new directory
os.rmdir("directory_name")
```

It will remove the specified directory.

Writing Python output to the files

In Python, there are the requirements to write the output of a Python script to a file.

The **check_call()** method of module **subprocess** is used to execute a Python script and write the output of that script to a file.

The following example contains two python scripts. The script file1.py executes the script file.py and writes its output to the text file **output.txt.**

Example

file.py

```
temperatures=[10,-20,-289,100]
def c_to_f(c):
    if c< -273.15:
        return "That temperature doesn't make sense!"
    else:
        f=c*9/5+32
        return f
for t in temperatures:
```

```
9.      print(c_,"to",_f(t))
```

file.py

```
1. import subprocess
2.
3. with open("output.txt", "wb") as f:
4.     subprocess.check_call(["python", "file.py"], stdout=f)
```

The file related methods

The file object provides the following methods to manipulate the files on various operating systems.

SN	Method	Description
1	file.close()	It closes the opened file. The file once closed, it can't be read or write anymore.
2	File.fush()	It flushes the internal buffer.
3	File.fileno()	It returns the file descriptor used by the underlying implementation to request I/O from the OS.
4	File.isatty()	It returns true if the file is connected to a TTY device, otherwise returns false.
5	File.next()	It returns the next line from the file.
6	File.read([size])	It reads the file for the specified size.

7	File.readline([size])	It reads one line from the file and places the file pointer to the beginning of the new line.
8	File.readlines([sizehint])	It returns a list containing all the lines of the file. It reads the file until the EOF occurs using readline() function.
9	File.seek(offset[,from)	It modifies the position of the file pointer to a specified offset with the specified reference.
10	File.tell()	It returns the current position of the file pointer within the file.
11	File.truncate([size])	It truncates the file to the optional specified size.
12	File.write(str)	It writes the specified string to a file
13	File.writelines(seq)	It writes a sequence of the strings to a file.

References:

https://www.javatpoint.com/python-files-io