Object Oriented Programming

BE(CSE) II-Semester

Prepared by
S. Durga Devi,
Assistant Professor,
CSE,CBIT

Unit-4.2

File Handling:

- File types, opening and closing files
- reading and writing files, file positions

File handling

- File is a collection of data
- Files are used to store the data permanently in secondary storage devices like hardDisk, CD etc. So that data can be accessed in the future.
- **Types of files:**

- like C , python supports two types of files.

1. Text files

- stores the data in form of ascii characters. Data in the text file processed sequentially.
- each line in text file ends with newline and file ends with EOF(end of the file

ex- student.txt

2. Binary files

- stores the data in form of binary like videos, images, audio files, zip files and executable files.
- Binary files are processed either sequentially or randomly.
- Binary files also ends with EOF(End of File)

Opening and closing files

What are the operations we can perform on files?

We can read, write, append, copy etc.

Before performing any operation on file(read, write, append), file must be opened once done required operation with the file , file should be closed.

- Python provides several built in methods to manipulate files

1. Opening a File:

- When a file is opening we must specify purpose of opening a file i.e mode.
- Syntax to open a file

```
Fileobject=open(filename,mode)
```

Where,

filename: name of the file with extension(like .c,.txt,.py etc)

mode: mode of a file

Note: filename and modes should be in string format

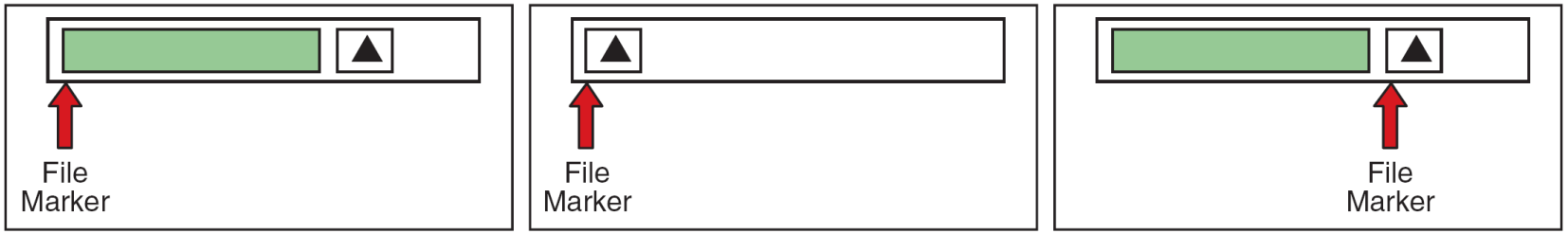
Types of modes applied on text file

Mode	Purpose
r	Opens existing file for read operation. File pointer points to beginning of a file. This is default mode, if file does not found gives error **FileNotFoundError**.
w	Opens existing file for write operation. Overrides the existing file, if file does not exist new file will be created.
a	Opens existing file for append operation. Adds new data at the end of the file, if file does not exist new file will be created. File pointer positioned at and of the file.
r+	opens a file for both read and write. Previous content of the file will not be deleted. File pointer positioned to beginning of a file
w+	Opens a file for write and read. Overrides the content of the file. File pointer positioned to beginning of a file
a+	Opens a file for append and read. Does not override the file. File pointer positioned at the end of a file.
x	Opens a file exclusive for write operation. If file already exist gives error that **FileExistsError**

- Note : r,w,a,r+,w+,a+ are applied on the text files. If these modes are suffixed with ‘b’ then the modes are used on the binary files.
- Modes on binary files are written as rb,wb,ab,rb+,wb+,ab+.
- Example

f1=open(“abc.txt”,”w”) # opens text file in write mode

f2= open(“abc.txt”,”wb”) # opens binary file in write mode.



Read Mode (r, r+) Write Mode (w, w+) Append Mode (a, a+)

Closing a file

- Closing a File:

  after completion of the operations on opened file, file is to be closed using **close()** method.

Syntax:

```
fileobject.close()
```

File object attributes

- Once file is opened, open() method returns File object. File object contains the information about the opened file.
- File information can be accessed by using following attributes.

1. **fileobject.name:** name of the file.
2. **fileobject.mode:** returns which mode file has been opened.
3. **fileobject.closed:** returns boolean values indicates whether file is closed or not
4. **fileobject.readable():** returns boolean value indicates whether file is readable or not
5. **fileobject.writable():** returns boolean value indicates whether file is writable or not.

Example

```
f=open("test.py","r")
print("Name of a opened file:",f.name)
print("file opened in :",f.mode)
print("file is readable:",f.readable())
print("file is writable", f.writable())
print("is file closed",f.closed)
```

output

```
Name of a opened file: test.py
file opened in : r
file is readable: True
file is writable False
is file closed False
```

FileattEx.py

Reading and writing files

Writing data to Text file:

- We can write character data on to the file by using following methods

1. write(str): write single string(single line) at a time on the text file
2. writelines(list of lines): write list of lines at a time on the text file.

```
f=open("temp.txt","w")
f.write("Name of faculty: S. Durga Devi\n")
f.write("Dept: CSE\n")
f.write("Designation: Assistant Professor\n")
print("Data written successfully")
f.close()
```

Output:

Goto your folder and open temp.txt file

Name of faculty: S. Durga Devi

Dept: CSE

Designation: Assistant Professor

writelines() method example

```
f=open("temp2.txt","w")
list=["S.Durga Devi\n","Assistant Professor\n","CSE\n"]
f.writelines(list)
print("data written successfully")
f.close()
```

append operation

To add data at the end of the file,open a file in appending mode using ‘a’.

Example: open a file temp.txt in append mode and add the details such as subjects taught by faculty.

- To add data at end of the file we can use write() for single line and writelines() method for multiple lines.

```
f=open("temp.txt","a")
f.write("subjects taught: Data structures, WT, MAD etc")
print("Data appended successfully")
f.close()
```

Reading data from Text file:

We can read a character data from text file using following methods

1. read(): reads entire data of a file
2. read(n): reads n no of characters
3. readline() : reads single line at a time.
4. readlines() : reads all lines into a list

All the read methods returns empty string when it reaches to end of the file.

1. read() method

```
f=open("writeEx.py","r")
data=f.read()
print(data)
f.close()
```

Output: displays the content of the writeEx.py file

```
f=open("temp.txt","w")
f.write("Name of faculty: S. Durga Devi\n")
f.write("Dept: CSE\n")
f.write("Designation: Assistant Professor\n")
print("Data written successfully")
f.close()
```

2. read(n) method:

```
f=open("writeEx.py","r")
data=f.read(23)
print(data)
f.close()
```

Output: reads first 23 characters from the 'writeEx.py' File and printed.

```
f=open("temp.txt","w")
```

3. readline() method- reads single line at a time

Create a file with name names.txt and add three names as Ram, Viswak, Anmisha.

names.txt

```
Ram
Viswak
Anmisha
```

readlineEx.py

```
f=open("names.txt","r")
line1=f.readline()
print(line1,end="")
line2=f.readline()
print(line2,end="")
line3=f.readline()
print(line3)
f.close()
```

output

```
Ram
Viswak
Anmisha
```

4. readlines() method : reads multiple lines into a list

```
f=open("names.txt","r")
list=f.readlines()
print(list)
```

```
['Ram\n', 'Viswak\n', 'Anmisha\n']
```

Open a file using 'with' statement

- we can also open a file using the statement 'with'.
- With statement is used to execute all the operations of a file as a block.
- If file opened using with statement, you need not close the file explicitly.
- After completion of all the operations on a file , file will be closed automatically even in case of the exceptions.

- syntax

```
with open("filename","mode") as f:
                                ---- define operations
```

```
with open("names.txt","r") as f:
                data=f.read()
                print(data)
                print("is file closed",f.closed)
print("is file closed",f.closed)
```

```
Ram
Viswak
Anmisha

is file closed False
is file closed True
```

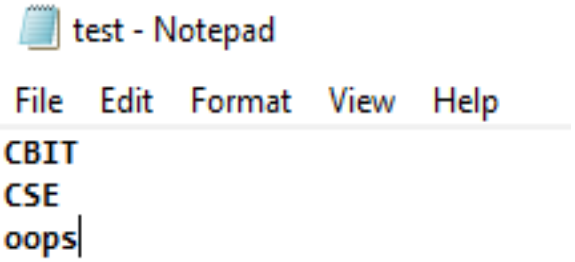
File positions

- Every file that is being opened is associated with a pointer is called file pointer.
- File pointer is used to move across file to read or write the data.
- When file is opened file pointer positioned at zero location means first character in the file.
- File pointer specifies the location within the file.
- Python provides methods to know the position of the file pointer.
- 1. tell() method
- 2. seek() method.

tell() method

- tell() method returns the current position of the cursor(file pointer) from the beginning of the file.

```
f=open('test.txt','r')
print(f.tell())
print(f.readline())
print(f.tell())
list=f.readlines()
for i in list:
                print(i)
print(f.tell())
```




```
0
CBIT

6
CSE

oops
15
```

seek() method

- seek(): Used to move the cursor(file pointer) to required position. Backward and forward.
- Syntax:

f.seek(offset, fromwhere)

- Where,

offset-> no of positions to move

from where has three attributes:

0 -> from the beginning of the file

1 -> from current position

2 -> from end of the file.

python2

- note: python2 support above 3 values and python3 does not support above 3 values python3 supports only 0.
- f.seek(20)

```
string="chaintanya bharathi institute of technology"
f=open('test.txt','w')
f.write(string)
with open('test.txt','r')as f:
                    data=f.read()
                    print(" current position is",f.tell())
                    f.seek(12)
                    print("current position is",f.tell())
                    print(f.tell())
                    f.seek(25)
                    print("current position is",f.tell())
print("successfully completed")
```

seekEx.py

programs

1. **Program to print the number of lines, words and characters present in the given file?**
2. **Create text file which contains all your friends names, display the count of each alphabet present in file**

   ex: chaitanya

   c-1, h-1,a-3,i-1,t-1,n-1,y-1

3. Write a program to copy the content of one file to another file.

4. Write a program to read file name from keyboard and open a file, convert a file into the upper case.

5. Read a file name from keyboard and open a file, convert a file into the lower case.

6. Take a text file which consists of numbers and alphabets, write a python program to find sum of numbers found in text file. For this program use regular expressions for that import module named as re.

Week8 question 5 solution

- 5. Case study: operator overloading
- Calculate monthly salary of a particular employee by defining following classes.
- 1. Define a class 'Employee' which defines following methods.
- 1. constructor with arguments name,salary
- 2. magic method __mul__(self,x)
- __mul__() method overloads * operator to calculate employee salary based on the no of days he presented in particular month.
- 2. Define a class 'employeeAttendance' which defines constructor with argument is 'days'

```
class Employee:
                def __init__(self,name,salary):
                                self.name=name
                                self.salary=salary
                def __mul__(self,other):
                                return self.salary*other.days
class EmployeeAtt:
                def __init__(self,days):
                                self.days=days
e=Employee('Durga',500)
t=EmployeeAtt(25)
print('This Month Salary:',e*t)
```


CHAITANYA BHARATHI
INSTITUTE OF TECHNOLOGY
1979


CHAITANYA BHARATHI
INSTITUTE OF TECHNOLOGY
1979