python™

File Handling-1

Prof. K. Adisesha

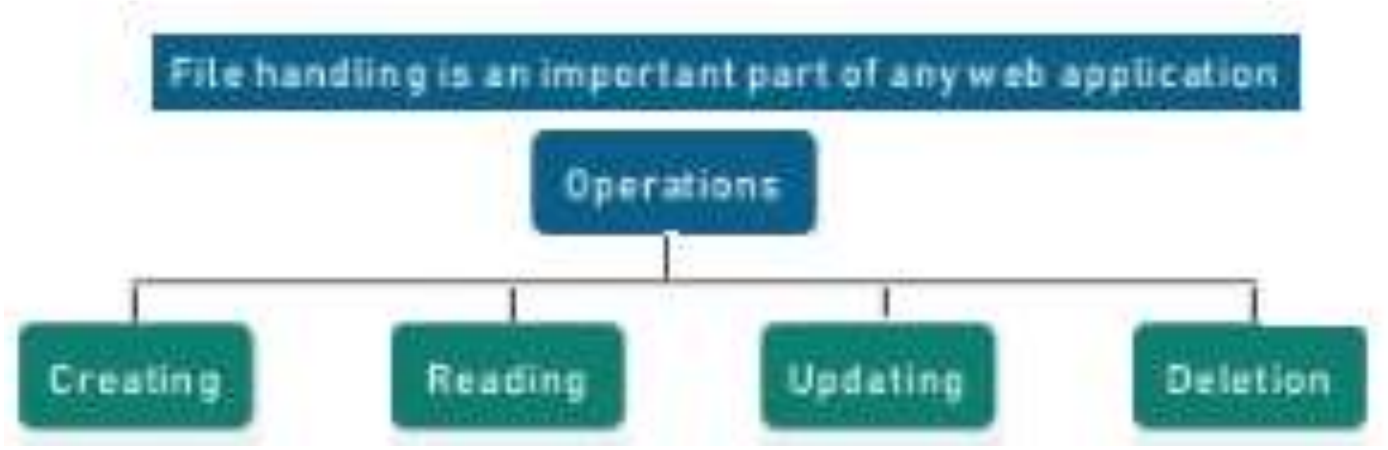
BE, MSc, M.Tech, NET, (Ph.D.)

Learning objectives

- Understanding Files
- Data Files
- Types of Files
- File Streams
- Understanding Text Files
- Opening and Closing Files
- Reading and Writing Files

File Handling

- File handling is an important part of any web application.
- A file in itself is a bunch of bytes stored ion some storage device like hard-disk, thumb-drive etc.,
- Python has several functions for creating, reading, updating, closing and deleting files.



Types of Files

- The data files are the files that store data pretaning to a specific application, for later use.
- Python allow us to create and manage three types of file

  1. TEXT FILE

  2. BINARY FILE

  3. CSV (Comma separated values) files

TEXT FILE

What is Text File?

- A text file is usually considered as sequence of lines.
- Line is a sequence of characters (ASCII or UNICODE), stored on permanent storage media.
- The default character coding in python is ASCII each line is terminated by a special character, known as End of Line (EOL).
- At the lowest level, text file will be collection of bytes.
- Text files are stored in human readable form and they can also be created using any text editor.

BINARY FILE

What is Binary File?

- A binary file contains arbitrary binary data i.e. numbers stored in the file, can be used for numerical operation(s).
- So when we work on binary file, we have to interpret the raw bit pattern(s) read from the file into correct type of data in our program.
- In the case of binary file it is extremely important that we interpret the correct data type while reading the file.
- Python provides special module(s) for encoding and decoding of data for binary file.

CSV files

What is CSV File?

- A comma-separated values (CSV) file is a delimited text file that uses a comma to separate values.
- Each line of the file is a data record.
- Each record consists of one or more fields, separated by commas.
- The use of the comma as a field separator is the source of the name for this file format.
- CSV file is used to transfer data from one application to another.
- CSV file stores data, both numbers and text in a plain text.

DIFFERENCE BETWEEN TEXT FILESAND BINARY FILES

Text Files	Binary Files
Text Files are sequential files	A Binary file contain arbitrary binary data
Text files only stores texts	Binary Files are used to store binary data such as image, video, audio, text
There is a delimiter EOL (End of Line \n)	There is no delimiter
Due to delimiter text files takes more time to process. while reading or writing operations are performed on file.	No presence of delimiter makes files to process fast while reading or writing operations are performed on file.
Text files easy to understand because these files are in human readable form	Binary files are difficult to understand
Text files are having extension .txt	Binary files are having .dat extension
Programming on text files are very easy.	Programming on binary files are difficult
Less prone to get corrupt as changes reflect as soon as the file is opened and can easily be undone	Can easily get corrupted, even a single bit change may corrupt the file.

Python File Streams

Standard Input, Output and Error Streams

- There are three different standard streams
  - Standard input device (stdin) – reads from the keyboard
  - Standard output device (stdout) – prints to the display and can be redirected as standard input.
  - Standard error device (stderr) - Same as stdout but normally only for errors.

Standard Input, Output devices as Files

- If you import sys module in your program then, for reading and writing for standard I/O devices:
  - sys.stdin.read() – Would let you read from Keyboard
  - sys.stdout.write()- Would let you write on Monitor

Python File Open

- The key function for working with files in Python is the open() function.
- The open() function takes two parameters; filename, and mode.
- There are four different methods (modes) for opening a file:
  - "r" - Read - Default value. Opens a file for reading, error if the file does not exist
  - "a" - Append - Opens a file for appending, creates the file if it does not exist
  - "w" - Write - Opens a file for writing, creates the file if it does not exist
  - "x" - Create - Creates the specified file, returns an error if the file exists
- In addition you can specify if the file should be handled as binary or text mode
  - "t" - Text - Default value. Text mode
  - "b" - Binary - Binary mode (e.g. images)
- These are the default modes. The file pointer is placed at the beginning for reading purpose, when we open a file in this mode.

Python File Open

To open a file for reading it is enough to specify the name of the file:

Syntax

```
f = open("demofile.txt")
```

- The code above is the same as:

```
f = open("demofile.txt", "rt")
```

- Because "r" for read, and "t" for text are the default values, you do not need to specify them.

- Note: Make sure the file exists, or else you will get an error.

FILE ACCESS MODES

- A file mode governs the type of operations like read/write/append possible methods in the opened file.

MODE	*Operations on File Opens in*
r+	Text File Read & Write Mode
rb+	Binary File Read Write Mode
w	Text file write mode
wb	Text and Binary File Write Mode
w+	Text File Read and Write Mode
wb+	Text and Binary File Read and Write Mode
a	Appends text file at the end of file, a file is created not exists.
ab	Appends both text and binary files at the end of file
a+	Text file appending and reading.
ab+	Text and Binary file for appending and reading.

- Example: **f=open("tests.dat", 'ab+')**

  tests.dat is binary file and is opened in both modes that is reading and appending.

Closing Files

- **close()**- method will free up all the system resources used by the file, this means that once file is closed, we will not be able to use the file object any more.
- fileobject. close() will be used to close the file object, once we have finished working on it.
- Syntax

```
<fileHandle>.close()
```

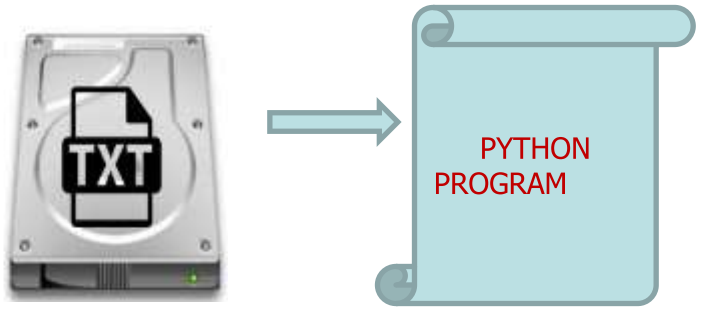
- For example:

```
fout.close()
```

Note: You should always close your files, in some cases, due to buffering, changes made to a file may not show until you close the file.

FILE READING METHODS

- A Program reads a text/binary file from hard disk. File acts like an input to a program.



- Followings are the methods to read a data from the file.
  1. read() METHOD
  2. readline() METHOD
  3. readlines() METHOD

read() METHOD

read() METHOD

- By default the read() method returns the whole text, but you can also specify how many characters you want to return:
  - The read() method is used to read entire file

  Syntax:

  ```
  fileobject.read()
  ```

  - Reads only Parts of the File

  Syntax:

  ```
  fileobject.read(size)
  ```

- The open() function returns a file object, which has a read() method for reading the content of the file:

  Example

  ```
  f = open("demofile.txt", "r")
  print(f.read(5))
  ```

  Returns the 5 first characters of the file "demofile.txt",

readline() METHOD

readline() METHOD

- readline() will return a line read, as a string from the file. First call to function will return first line, second call next line and so on.

Syntax:

```
fileobject.readline()
```

- The open() function returns a file object, which has a readline() method for reading the content of the file will return only one line from a file, but demofile.txt file containing three lines of text

Example

```
f = open("demofile.txt", "r")
print(f.readline())
```

Returns the first line of the file "demofile.txt",

readlines() METHOD

readlines() METHOD

- readlines() method will return a list of strings, each separated by \n
- readlines()can be used to read the entire content of the file. You need to be careful while using it w.r.t. size of memory required before using the function.

Syntax:

```
fileobject.readlines()
```

- As it returns a list, which can then be used for manipulation.

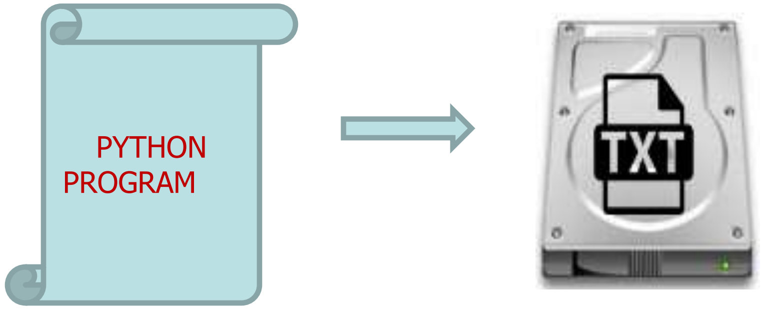
Example

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

The readlines() method will return a list of strings, each separated by \n of the file "demofile.txt",

FILE WRITING METHODS

- A Program writes into a text/binary file from hard disk.



- Followings are the methods to write a data to the file.
  1. **write () METHOD**
  2. **writelines() METHOD**

- To write to an existing file, you must add a parameter to the open() function:
  - **"a" - Append** - will append to the end of the file
  - **"w" - Write** - will overwrite any existing content

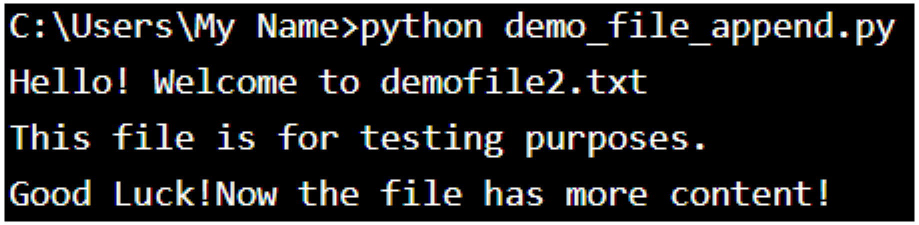
write () METHOD

- write() method takes a string (as parameter) and writes it in the file.
- For storing data with end of line character, you will have to add \n character to end of the string
- Example
- Open the file "demofile2.txt" and append content to the file:

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

Output:



```
C:\Users\My Name>python demo_file_append.py
Hello! Welcome to demofile2.txt
This file is for testing purposes.
Good Luck!Now the file has more content!
```

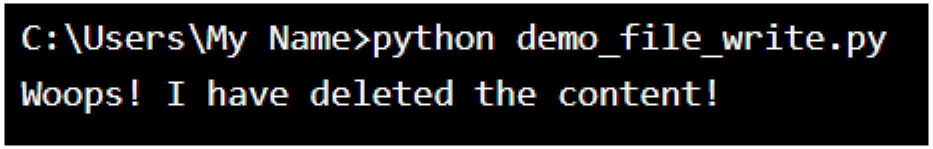
write () METHOD

- write() method takes a string (as parameter) and writes it in the file.
- if you execute the program n times, the file is opened in w mode meaning it deletes content of file and writes fresh every time you run.
- Example
- Open the file "demofile3.txt" and overwrite the content:

```
f = open("demofile2.txt", “w")
f.write("Woops! I have deleted the content!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

Output:



```
C:\Users\My Name>python demo_file_write.py
Woops! I have deleted the content!
```

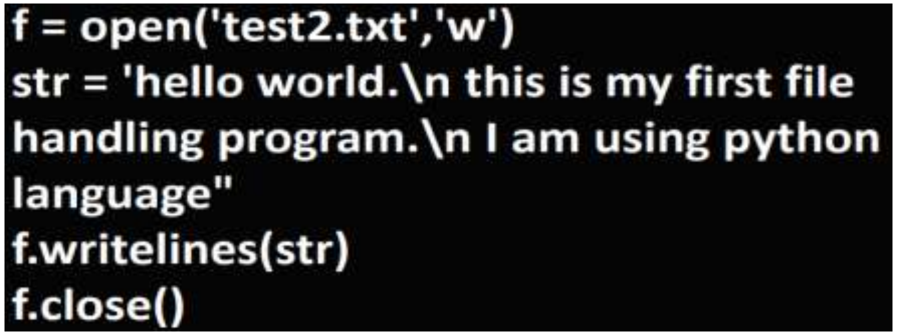
writelines() METHOD

- For writing a string at a time, we use write() method, it can't be used for writing a list, tuple etc. into a file.
- Sequence data type can be written using writelines() method in the file. It's not that, we can't write a string using writelines() method.
- So, whenever we have to write a sequence of string / data type, we will use writelines(), instead of write().

Syntax:

```
fileobject.writelines(seq)
```

Example



```
f = open('test2.txt','w')
str = 'hello world.\n this is my first file
handling program.\n I am using python
language"
f.writelines(str)
f.close()
```

RANDOM ACCESS METHODS

- All reading and writing functions discussed till now, work sequentially in the file.
- To access the contents of file randomly –following methods.

  1. seek method

  2. tell method

RANDOM ACCESS METHODS

Seek() method :

- seek()method can be used to position the file object at particular place in the file.

  syntax is :

  fileobject.seek(offset [, from_what])

- Here offset is used to calculate the position of fileobject in the file in bytes. Offset is added to from_what (reference point) to get the position.
- Value reference point:

  0 -beginning of the file

  1 -current position of file

  2 -end of file

- Default value of from_what is **0**, i.e. beginning of the file.

  Example: **f.seek(7)**

- keeps file pointer at reads the file content from 8th position onwards to till EOF.

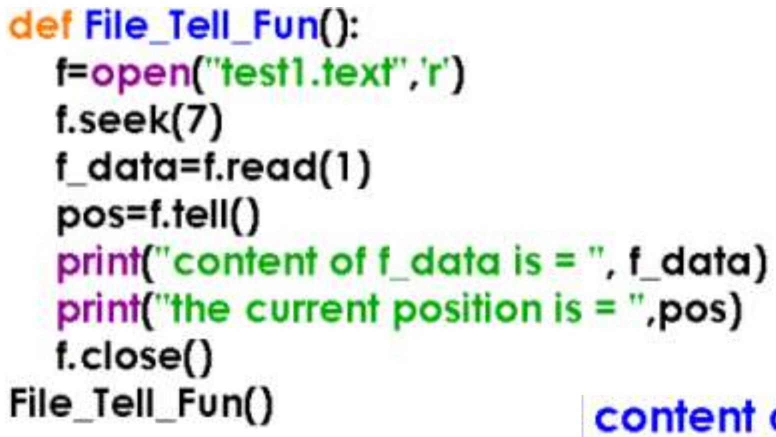
RANDOM ACCESS METHODS

tell method

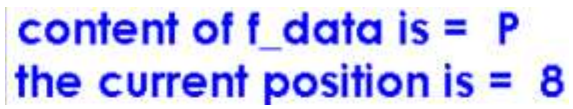
- tell() method returns an integer giving the current position of object in the file.
- The integer returned specifies the number of bytes from the beginning of the file till the current position of file object.

  Syntax:

  fileobject.tell()
- tell() method returns an integer and assigned to pos variable. It is the current position from the beginning of file.



```
def File_Tell_Fun():
    f=open("test1.text",'r')
    f.seek(7)
    f_data=f.read(1)
    pos=f.tell()
    print("content of f_data is = ", f_data)
    print("the current position is = ",pos)
    f.close()
File_Tell_Fun()
```



```
content of f_data is =  P
the current position is =  8
```

PYTHON FILE OBJECT ATTRIBUTES

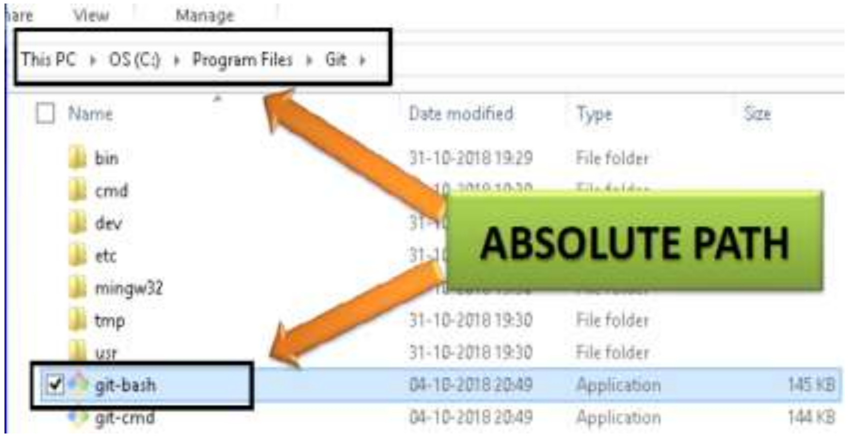
- File attributes give information about the file and file state.

Attribute	Function
name	Returns the name of the file
closed	Returns true if file is closed. False otherwise.
mode	The mode in which file is open.
softspace	Returns a Boolean that indicates whether a space character needs to be printed before another value when using the print statement.

HANDLING FILES THROUGH OS MODULE

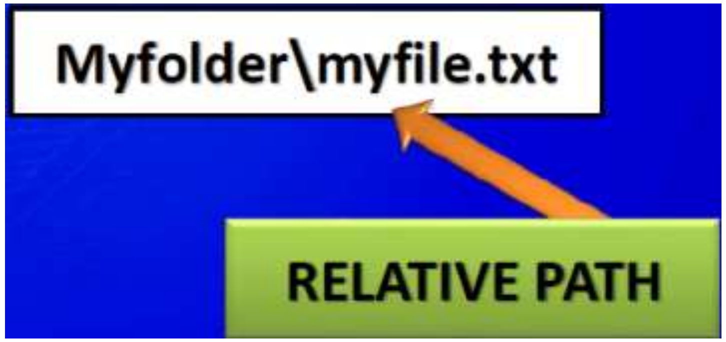
- The os module of Python allows you to perform Operating System dependent operations such as making a folder, listing contents of a folder, know about a process, end a process etc..
- Let's see some useful os module methods that can help you to handle files and folders in your program.
  - **ABSOLUTE PATH**
  - **RELATIVE PATH**

Absolute path of file is file location, where in it starts from the top most directory



Name	Date modified	Type	Size
bin	31-10-2018 19:29	File folder	
cmd			
dev			
etc			
mingw32			
tmp	31-10-2018 19:30	File folder	
usr	31-10-2018 19:30	File folder	
git-bash	04-10-2018 20:49	Application	145 KB
git-cmd	04-10-2018 20:49	Application	144 KB

Relative Path of file is file location, where in it starts from the current working directory



Conclusion!

Thank you