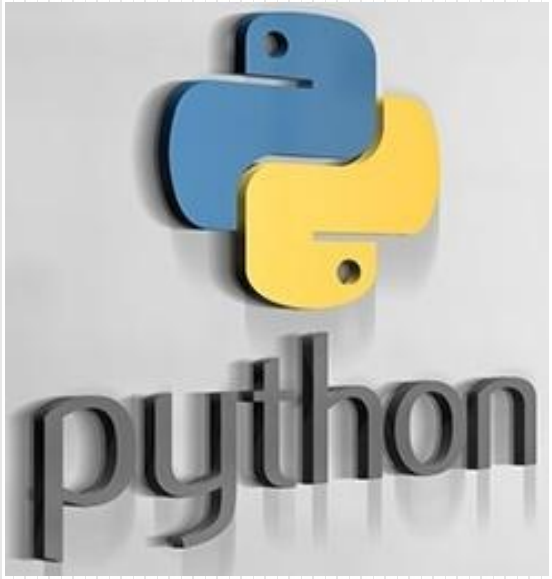
Scripting Language

UNIT-III



K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT- Basara, T.S, IND.

Syllabus

- Introduction to Files
- File Path
- Types of Files
- Opening
- Closing
- Reading
- Writing
- Merge Operations on Files

Introduction to Files

- A File is a collection of data stored on a secondary storage device like hard disk
- File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk)
- We were making use of input() to enter the data, but this becomes problem when we come to huge amount of data to be processed
- Therefore better solution is to create a file and store the data and read the data using File program
- Since, random access memory (RAM) is volatile which loses its data when the program end or computer is turned off,
- When we want to read from or write to a file we need to open it first then read the content previously written and write the new content. When we are done, it needs to be closed, so that resources that are tied with the file are freed

- A file is basically used because real life applications involve large amounts of data and such situations the console oriented I/O operations poses two major problems:
  - Time consumption to handle large amount of data through terminals
  - When performing I/O operations through terminal, the entire data is lost when program is terminated or computer is turned off
- In order to use files, we have to learn file input and output operations, how to read and write content to file

File Path

- Most file systems that are used today stores files in a tree(or hierarchical) structure
- At the top of the tree is one or more root nodes followed by other files and folders. The subfolders may also consists of files and folders to the atmost depth. The type of the file is indicated by its extension.
- Every file is identified by its path that begins from the root node or the root folder [pathname]
- Ex: C:\Python36\Scripts\pip.py [Delimiter backslash (\) is used as seperator
- Relative Path [Partial path not from the root] and Absolute Path [Complete Path from the root node till the particular file]
- When a relative path is specified , the relative path is joined with the current directory to create an absolute file path

Types of files

- Python supports two types of files:
  - Text Files
  - Binary Files
- **Text Files**
- A text file is a stream of characters that can be sequentially processed by a computer in forward direction.
- Hence text file is opened for only one kind of operation (reading, writing or appending) at any given time
- Text files can process characters, they can only read or write data one character at a time
- In a text file, each line contains zero or more characters and ends with one or more characters that specify end of line. Each line in a text file can have maximum of 255 characters
- Note: In a text file, each line of data ends with a newline character. Each file ends with a special character called the end-of-file (EOF)

- **Binary Files**
- A binary file is a collection of bytes, which may contain any type of data, encoded in binary form for computer storage and processing purpose
- It includes files such as word processing documents, pdfs,images spreadsheets, videos, zip files and other executable programs
- A binary file is also referred as character stream
- While text files can be processed sequentially, binary files are processed sequentially or randomly depending on the application
- Takes less amount data when compared to text files
- Binary files are mainly used to store data beyond text such as images,executable files etc.

Operations on Files:

Opening a File – The open( ) Function:

- Before working with Files(read,write) you have to open the File. To open a File, Python built in function open( ) is used. It returns an object of File which is used with other functions.
- Syntax: **fileObject=open(file_name [, access_mode])**

file_name: is a string value that specifies name of the file which you want to access.

access_mode: It specifies the mode in which File is to be opened i.e., read, write or append so on… access mode is optional parameter but by Default file access mode is read(r)

open( ): function returns a file object. This file object will be used to read, write or to perform any other operation. It works as a file handle

- **Example**
- f = open("python.txt",'w') # open file in current directory
- f = open("/home/rgukt/Desktop/python/hello.txt")
  # specifying full path

```
file=open("python.txt","w")# open file in current directory
print(file)


# specifying full path
file=open(("C:/Users/user/Desktop/python.txt"))
print(file)


#Program to print details of file object
file=open("sample.txt","rb")
print(file)
```

Access Modes of File

- There are different modes of file in which it can be opened. They are mentioned in the following table
  - A File can be opened in two modes: 1. Text Mode 2.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 image or exe files.

Mode	Purpose
r	This is the default mode of opening a file which opens the file for reading only. The file pointer is placed at the beginning of the file.
rb	This mode opens a file for reading only in binary format. The file pointer is placed at the beginning of the file.

r+	This mode opens a file for both reading and writing. The file pointer is placed at the beginning of the file.
rb+	This mode opens the file for both reading and writing in binary format. The file pointer is placed at the beginning of the file.
w	This mode opens the file for writing only. When a file is opened in w mode, two things can happen. If the file does not exist, a new file is created for writing. If the file already exists and has some data stored in it, the contents are overwritten.
wb	Opens a file in binary format for writing only. When a file is opened in this mode, two things can happen. If the file does not exist, a new file is created for writing. If the file already exists and has some data stored in it, the contents are overwritten.
w+	Opens a file for both writing and reading. When a file is opened in this mode, two things can happen. If the file does not exist, a new file is created for reading as well as writing. If the file already exists and has some data stored in it, the contents are overwritten.
wb+	Opens a file in binary format for both reading and writing. When a file is opened in this mode, two things can happen. If the file does not exist, a new file is created for reading as well as writing. If the file already exists and has some data stored in it, the contents are overwritten.
a	Opens a file for appending. The file pointer is placed at the end of the file if the file exists. If the file does not exist, it creates a new file for writing.
ab	Opens a file in binary format for appending. The file pointer is at the end of the file if the file exists. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both reading and appending. The file pointer is placed at the end of the file if the file exists. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file in binary format for both reading and appending. The file pointer is placed at the end of the file if the file exists. If the file does not exist, a new file is created for reading and writing.

Mode	Purpose
t	Open in text mode (Default)
b	Open in Binary mode
+	Open a file for updating (reading and writing)

Attributes of File Object

- Once a file is successfully opened, a file object is returned.
- Using this file object you can easily access different types of info. related to the file
- This information can be obtained by reading values of specific attributes of the file

Attribute	Returns [Description]
fileobject.name	Returns the name of the file
fileobject.mode	Returns the access mode in which file is being opened
fileobject.closed	Returns Boolean Value True, in case if file is closed else returns false

```
result = open("data.txt", "w")
print ("Name of the file:",result.name   )
print ("Opening mode:",result.mode   )
print   ("Closed or not:",result.closed)
result.close()
print   ("Closed or not:",result.closed)
```

```
Name of the file: data.txt
Opening mode: w
Closed or not: False
Closed or not: True
```

Closing a File:

- Once you are finished with the operations on File at the end you need to close the file object.
- Once a file object is closed,you cannot further read or write into the file
- Syntax: **fileobject.close( )**
- Close() method frees up any system resources such as file descriptors, file locks etc., which are associated with a particular file
- It is done by the close( ) method. close( ) method is used to close a File.
- Once the file is closed using the close() method, any attempt to use the file object will result in an error

```
result = open("data.txt", "w")
print("Name of the file:",result.name)
print("File is closed",result.closed)
print("File is now being closed. You cannot use the file object")
result.close()
print(" File is Closed :",result.closed)
print(result.read())
```

```
Name of the file: data.txt
File is closed False
File is now being closed. You cannot use the file object
 File is Closed : True
Traceback (most recent call last):
  File "C:/Users/user/Desktop/f_close.py", line 7, in <module>
    print(result.read())
ValueError: I/O operation on closed file.
```

- **Note:** 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.

Reading and Writing Files

- read( ) and write( ) are used to read data from file and write data to files

Writing to a File:

The write( ) and writelines( ) methods

- **write( )** method is used to write a string to an already opened file.
- String may include alphabets, numbers, special characters or other symbols
- Write( ) method does not add a newline character('\n') to the end of the string
  - Syntax: **fileobject.write(string)**
- As per the syntax, the string that is passed as an argument to the write() is written into the opened file
- The **writelines( )** method is used to write a list of strings

Program to write content to a file

```
file=open("File.txt","w")
file.write("Hello aLl, Hope you are enjoying learning my classes")
file.close()
print("Data Written into the file....")
```

- The writeline( ) method returns None
- The writelines( ) method is used to write a list of strings

```
# Program to write to a file using the writelines()method
file=open("File.txt","w")
lines=["Hello Welcome", "Come join us", " Have fun"]
file.writelines(lines)
file.close()
print("Data Written to files..")
```

append mode

- Once you have stored data in a file, you can always open that file again to write more data or append data to it.
- To append a file, you must open it using ‘a’ or ‘ab’ mode depending on whether it is a text file or a binary file
- If you open a file in append mode then the file is created if it did not exist
- Note that if you open a file in 'w' or 'wb' mode and then start writing data into it, then its existing contents would be overwritten. So always open the file in 'a' or 'ab' mode to add more data to existing data stored in the file.
- Appending data is especially essential when creating a log of events or combining a large set of data into one file.

- **Append Mode:** Syntax: **fileobject.open("filename","a")**

```
file=open("File.txt","a")
file.write("\n Start working with multiple frameworks")
file.close()
print("Data Written into the file....")

f = open("try.txt","a")
f.writelines("hello\nhi\nhow\n")
f.writelines("hello\nhi\nhow\n")
f.close()
```

Reading from a File:

read( ) and readline( ) methods

- read( ) method is used to read a string from an already opened file data from the file.
- Syntax: **fileobject.read([count])**
- Parameter ‘count’ is an optional which specifies the number of bytes to be read from the file
- The read method starts reading from the beginning of the file and if count is missing or has a negative value, then it reads the entire contents of the file [till the end of the file]
- If size parameter is not specified, it reads and returns up to the end of the file

```
# Program to print the first 10 characters of the file
file=open("File.txt","r")
print(file.read(10))
file.close()
```

Program to read and write data from a file

```
obj=open("abcd.txt","w")
obj.write("Welcome to the world of Python\n Yeah its great\n")
obj.close()

obj1=open("abcd.txt","r")
s=obj1.read( )
print(s )
obj1.close()

obj2=open("abcd.txt","r")
s1=obj2.read(20)
print(s1 )
obj2.close()
```

```
f=open("test.txt","w")
f.write("my first file\n")
f.write("This file\n\n")
f.write("contains three lines\n")
f.write('''hello
this is
first program
in files''')#we can write multiple lines
f.close()

f = open("test.txt",'r')
print (f.read(4))
print (f.read())
```

readline( ) and readlines( ) methods

- we can use readline() method to read individual lines of a file. This method reads a file till the newline**.** Reads in at most n bytes/characters if specified.
- the readlines() method returns a list of remaining lines of the entire file. It , including the newline character. Reads in at most n bytes/characters if specified.
- **Example**

```
f = open("try.txt","r")
print(f.readline())
print (f.readline())
print(f.readlines())
f.close()
```

Opening Files using with keyword

- This has the advantage that the file is properly closed after it is used even if an error occurs during read or write operation or even when you forget to explicitly close the file

```
# with using with keyword
with open("C:\Users\user\Desktop\file1.txt","rb")as file:
    for line in file:
        print(line)
print("Lets's check if the file is closed:",file.close())


#without using without keyword
file=open("file1.txt","rb")
for line in file:
    print(line)
print("Lets's check if the file is closed:",file.close())
```

- **Replace a word with another word**

```
# Read in the file
with open('try.txt', 'r') as file :
filedata = file.read()
# Replace the target string
filedata = filedata.replace('that', 'somes')
# Write the file out again
with open('try.txt', 'w') as file:
file.write(filedata)
```

Splitting Words

- Python allows us to read lines from a file and splits the line (treated as a string) based on a character.
- By default, this character is space but we can even specify any other character to split words in the string

```
'''Program to split the line into a series of words and
    use space to perform the split operation'''

with open("C:/Users/user/Desktop/file1.txt","r") as file:
    line=file.readline()
    words=line.split()
    print(words)

''' Progrma to perform split operation whevever a comma is encountered'''

with open("C:/Users/user/Desktop/file1.txt","r") as file:
    line=file.readline()
    words=line.split(',')
    print(words)
```

Python File Methods

- There are various methods available with the file object

Method	Description
close()	Close an open file. It has no effect if the file is already closed.
detach()	Separate the underlying binary buffer from the TextIOBase and return it.
fileno()	Return an integer number (file descriptor) of the file.
flush()	Flush the write buffer of the file stream.
isatty()	Return True if the file stream is interactive.
read(n)	Read atmost n characters form the file. Reads till end of file if it is negative or None.
readable()	Returns True if the file stream can be read from.

Method	Description
readline(n=-1)	Read and return one line from the file. Reads in at most n bytes if specified.
readlines(n=-1)	Read and return a list of lines from the file. Reads in at most n bytes/characters if specified.
seek(offset,from=SEEK_SET)	Change 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)	Resize the file stream to size bytes. If sizeis not specified, resize to current location.
writable()	Returns True if the file stream can be written to.
write(s)	Write string s to the file and return the number of characters written.
writelines(lines)	Write a list of lines to the file.

File Positions

- With every file, the file management system associates a pointer often known as *file pointer* that facilitate the movement across the file for reading and/ or writing data. The file pointer specifies a location from where the current read or write operation is initiated. Once the read/write operation is completed, the pointer is automatically updated.
- Python has various methods that tells or sets the position of the file pointer. For example, the tell() method tells the current position within the file at which the next read or write operation will occur. It is specified as number of bytes from the beginning of the file. When you just open a file for reading, the file pointer is positioned at location 0, which is the beginning of the file.
- The seek(offset[, from]) method is used to set the position of the file pointer or in simpler terms, move the file pointer to a new location. The offset argument indicates the number of bytes to be moved and the from argument specifies the reference position from where the bytes are to be moved.

seek and tell method

- We can change our current file cursor (position) using the **seek( )** method.
- Similarly, the **tell( )** method returns our current position (in number of bytes).
- **Example**

```
#filename=input("Enter your file name")
f = open("try.txt","r+w")
f.seek(6)
print (f.tell())
f.write("that")
print (f.read())
f.close()
```

File Positions - Example

```
file = open("File1.txt", "rb")
print("Position of file pointer before reading is : ", file.tell())
print(file.read(10))
print("Position of file pointer after reading is : ", file.tell())
print("Setting 3 bytes from the current position of file pointer")
file.seek(3,1)
print(file.read())
file.close()
```

OUTPUT

```
Position of file pointer before reading is : 0
Hello All,
Position of file pointer after reading is : 10
Setting 3 bytes from the current position of file pointer
pe you are enjoying learning Python
```

Mail Merge

- To perform mail merge, create three files – Names.txt that stores names of the recerivers, Body.txt that stores the content or body of mail(or message) to be sent and the main program file which is a python script that opens both the files and merges them
- We open both the files in reading mode and iterate over each name using a for loop
- New files with "Name.txt" are created, where [Name] is the name of the receiver as specified in the file
- The strip() methods has been used to clean up leading and trailing whitespaces
- The write() method is used to write the body (contents of the mail) into the new[name].txt files

```
# Contents of the Names.txt
#Ravikanth
#Anusha
#Rithwik

#Contents of Body.txt
#Greetings
# We wlcome you all to join us in start-up process initated by RGUKT
# Looking forward for all your valuable support


# Program for Mail Merge
names_file=open("names.txt",'r')
body_file=open("body.txt",'r')
body = body_file.read()
for name in names_file:
        mail = "Hello "+name+body
        mail_file=open(name.strip()+".txt",'w')
        mail_file.write(mail)
        mail_file.close()
names_file.close()
body_file.close()
```

Renaming and Deleting Files

- The **os module** in Python has various methods that can be used to perform file-processing operations like renaming and deleting files.
- **The rename( ) Method**: The rename( ) method takes two arguments, the current filename and the new filename.

  Syntax : os.rename(old_file_name, new_file_name)
- **The remove( ) Method:** This method can be used to delete file(s). The method takes a filename (name of the file to be deleted) as an argument and deletes that file.

  Syntax is: os.remove(file_name)

```
import os
os.rename("File1.txt", "Students.txt")
print("File Renamed")
```

OUTPUT

```
File Renamed
```

```
import os
os.remove("File1.txt")
print("File Deleted")
```

OUTPUT

```
File Deleted
```

Directory Methods

- **The mkdir( ) Method:** The mkdir( ) method of the OS module is used to create directories in the current directory. The method takes the name of the directory (the one to be created) as an argument.

  Syntax : os.mkdir("new_dir_name")

  #mkdirs() is used to create more than one folder.

- **The getcwd( ) Method:** The getcwd( ) method is used to display the current working directory (cwd)

  Syntax: os.getcwd( )

- **The chdir( ) Method:** The chdir( ) method is used to change the current directory. The method takes the name of the directory which you want to make the current directory as an argument.

  Syntax : os.chdir("dir_name")

- **The rmdir( ) Method:** The rmdir( ) method is used to remove or delete a directory. For this, it accepts name of the directory to be deleted as an argument. However, before removing a directory, it should be absolutely empty and all the contents in it should be removed.

  Syntax:os.rmdir(("dir_name")

Directory Methods - Examples

```
import os
os.rename("File1.txt", "Students.txt")
print("File Renamed")
```

OUTPUT

```
File Renamed
```

```
import os
os.remove("File1.txt")
print("File Deleted")
```

OUTPUT

```
File Deleted
```

```
import os
os.mkdir("New Dir")
print("Directory Created")
```

OUTPUT

```
Directory Created
```

```
import os
print("Current Working Directory is : ", os.getcwd())
os.chdir("New Dir")
print("After chdir, the current Directory is now..... ",
end = ' ')
print(os.getcwd())
```

OUTPUT

```
Current Working Directory is : C:\Python27
After chdir, the current Directory is now.....  C:\Python27\New Dir
```

Programming Tip: OS Object Methods: This provides methods to process files as well as directories.

Methods from the os Module

- **os.path.abspath( )** method uses the string value passed to it to form an absolute path. Another way to convert a relative path to an absolute path
- **os.path.isabs(path)** method accepts a file path as an argument and returns True if the path is an absolute path and False otherwise.
- **os.path.relpath(path, start)** method accepts a file path and a start string as an argument and returns a relative path that begins from the start. If start is not given, the current directory is taken as start.
- **os.path.dirname(path)** returns a string that includes everything specified in the path (passed as argument to the method) that comes before the last slash.
- **os.path.basename(path)** returns a string that includes everything specified in the path (passed as argument to the method) that comes after the last slash.

Methods from the os Module

- **os.path.split(path)** : It accepts a file path and returns its directory name as well so it is equivalent to using two separate methods os.path.dirname() and os.path.basename()
- **os.path.getsize(path)** :returns the size of the file specified in the path argument.
- **os.listdir(path)** : It returns a list of filenames in the specified path.
- **os.path.exists(path)** :As the name suggests accepts a path as an argument and returns True if the file or folder specified in the path exists and False otherwise.
- **os.path.isfile(path) :** As the name suggests accepts a path as an argument and returns True if the path specifies a file and False otherwise.
- **os.path.isdir(path)** :As the name suggests accepts a path as an argument and returns True if the path specifies a folder and False otherwise.

Methods from the os Module — Examples

```
import os
print("os.path.isabs(\"Python\\Strings.docx\") = ",
os.path.isabs("Python\\Strings.docx"))
print("os.path.isabs(\"C:\\Python34\\Python\\Strings.docx\") = ",
os.path.isabs("C:\Python34\Python\\Strings.docx"))
```

OUTPUT

```
os.path.isabs("Python\Strings.docx") =  False
os.path.isabs("C:\Python34\Python\Strings.docx") =  True
```

```
import os
print("os.path.relpath(\"C:\\Python\\Chapters\\First
Draft\\Strings.docx\") = ", os.path.relpath("C:\Python\Chapters\First
Draft\Strings.docx", "C:\Python"))
```

OUTPUT

```
path.relpath("C:\Python\Chapters\First Draft\Strings.docx")= Chapters\First Draft\
Strings.docx
```

For Details Contact Me @ :

9247448766

ravikanth27787@gmail.com

python

Programming Language