Dr. D. Y. PATIL INSTITUTE OF TECHNOLOGY, PIMPRI, PUNE-18

Department of First Year Engineering

Programming and Problem Solving

Unit- 6 Notes

Unit VI : File Handling and Dictionaries

6.1 Introduction

Q1. What are files? Why do we need them?

Ans.

- A file is a collection of data stored on a secondary storage device like hard disk.
- A file or a computer file is a chunk of logically related data or information which can be used by computer programs.

Need for file:

- When there is huge amount of data to be processed, then processing data that has been entered though computer keyboard using input () becomes very tedious.
- A better solution, therefore, is to combine all the input data into a file and then design a python program to read this data from the file whenever required.
- When a program is being executed, its data is stored in random access memory (RAM).
- Though RAM can be accessed faster than CPU, it is also volatile, which means when the program ends or computer shuts down, all the data is lost.
- If the data is to be used in future then it should be stored on permanent or non-volatile storage media such as hard disk, USB drive, DVD etc.
- Data on non-volatile storage media is stored in named locations on the media called files.
- A file is basically used because real life applications involve large amounts of data in such situations the console oriented I/O operations pose two major problems:
    - First, it becomes cumbersome and time consuming to handle huge amount of data through terminals.
    - Second, when doing I/O using terminals, the entire data is lost when either the program is terminated or computer is turned off.

Therefore, it becomes necessary to store data on permanent storage (the disks) in the form of files and read whenever necessary, without destroying data.

6.2 File Path

Q2. Explain the significance of root node. What are the types of file path?

Ans.

- Files are stored on storage medium like hard disk in such a way that they can be easily retrieved as and when required.
- Most of the files systems that are used today stores file in a tree (or hierarchical) structure.
- At the top of tree are one (or more) root nodes.
- Under the root node, there are files and folders (or directories) and each folder can in turn contain other files and folders.
- Even these folders can contain other files and folders and this can go almost limitless depth.
- The type of file is indicated by its extension.
- Consider the tree structure given in figure below:

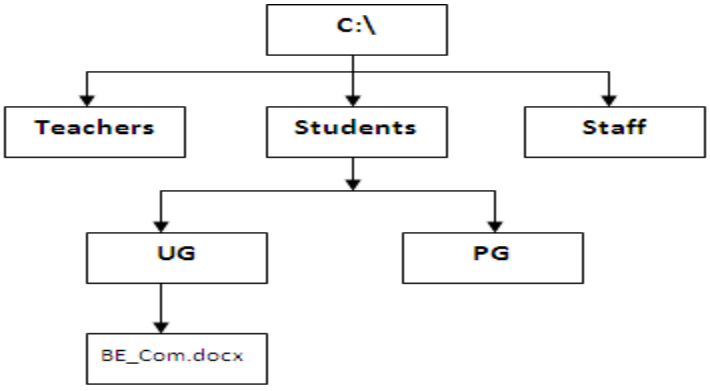

Fig: Files and Folders

- Every file is identified by its path that begins from root node or the root folder.
- In windows, C:\ is the root folder but you can also have a path that sarts from other drives like D:\, E:\, etc.
- The file path is known as pathname.

- In the above figure, C:\ is the root node, it has folder Students, it has sub-folder UG, which has file BE_Com.docx
- File path is written as:
  C:\Students\UG\ BE_Com.docx

Types of File Path - Absolute path and Relative path:

- A file path can be either relative or absolute.
- An absolute path always contains the root and the complete directory list to specify the exact location of file.
- Relative path, on the other hand, needs to be combined with another path in order to access a file.
- A relative path starts with respect to current working directory and therefore, lacks the leading slashes.
- For example:

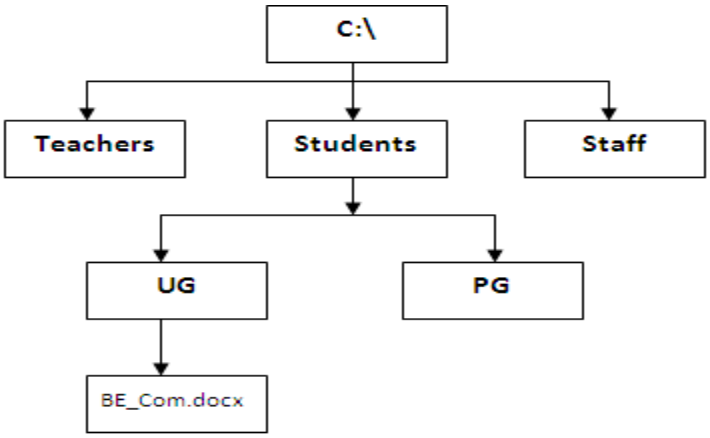

Fig: Files and Folders

- **Here, Absolute path is C:\Students\UG\ BE_Com.docx**
- All the information needed to locate file BE_Com.docx is contained in the above path.
- **Here, Relative path may be UG\ BE_Com.docx**
- Note that when a relative path, is specified, the relative path is joined with the current directory to create an absolute path.

6.3 Types of Files

Q3. What is text file and binary file?

Ans.

Python supports two types of files- text files and binary files.

A. ASCII Text Files

- A text file is a stream of characters that can be sequentially processed by a computer in forward direction.
- For this reason, a text file is usually opened for only one kind of operation (reading, writing or appending) at any given time.
- A text file is human readable.
- In text file each line contains any number of characters including one special character that denotes end of file (EOF).
- When data is written to a file, each new line character is converted to carriage return/ line feed character. Similarly, when data is read from the file, each carriage return feed character is converted to new line character.
- Each line in text file ends with newline character and each file ends with special character called EOF (End of File).

B. Binary Files

- Binary file is a file which contains data encoded in binary format.
- This data is mainly used for computer storage or for processing purpose.
- The binary data can be word processing document, images, sound or any executable program.
- We cannot read binary files easily as the data is in encoded format.
- The text file can be processed sequentially while binary file can be processed sequentially or randomly depending upon the need.
- Like text files, binary files also put EOF as endmarker.

Q4. Differentiate between text file and binary file.

Ans.

Sr. No	Text File	Binary File
1	Data is represented in the form of **characters.**	Data is represented in the form of **encoded form.**
2	The **plain text** can be stored in this type of file.	The **image, audio, text, executable and video** data can be stored in this type of file.
3	These file can be processed **sequentially**.	Binary file can be processed **sequentially or randomly** depending upon need.
4	It **cannot be corrupted** easily.	Even if **single bit is changed then the file get corrupted**.
5	It **can be opened and read using si**mple text editor like Notepad.	It **cannot be opened and read using simple** text editor like Notepad.
6	It can have file extension such as **.py or .txt**	It can have application defined extension.
7	It needs more memory to store the data.	It takes less memory space to store the data.
8	Example: In text file integer value 123 will be stored as a sequence of three characters and each character takes one byte of memory. So total **three byte** required to store 123	Example: In binary file the integer value 123 will be stored in **two bytes** in binary form.

6.4 Opening and Closing Files

Q5. How to open the file? Also the explain various modes in which the file is open.

OR

Explain the utility of open function. Explain any 4 modes of opening a file.

Ans.

- A file is a collection of data stored on secondary storage device like hard disk. So to read or write data from a file we need to open a file.
- Pyhon provides inbuilt functionality named as **open()** function.
- This function creates a file object, which will be used to invoke methods associated with it.
- Syntax:

  fileobj=open(file_name [, access_mode])
- here,

- ✓ **file_name** is a string value that specifies name of the file that you want to access.
- ✓ **Access_mode** indicates the mode in which the file has to be opened. ie- read, write, append etc.

- The open function returns a file object. This file object will be used to read, write or to perform any other operation on the file. It works like a file handle.

Acccess Mode:

- ✓ Access mode is optional parameter and the default file access mode is read(r).

Sr. No.	Access Mode	Purpose/ When to Use
1.	**r**	-This is the default mode of opening a file. -Which opens a file in read only mode. -File pointer is placed at the beginning of the file.
2.	**rb**	-This opens a file in read only mode in binary format. -File pointer is placed at the beginning of the file.
3.	**r+**	-This opens a file for both reading and wring mode. -File pointer is placed at the beginning of the file.
4.	**rb+**	-This opens a file for both reading and wring mode in binary format. -File pointer is placed at the beginning of the file.
5.	**w**	-This mode opens a file in writing only. -If file does not exist new file get created. - If the file already exist, the contents are overwritten.
6.	**wb**	-This mode opens a file in writing only in binary format. -If file does not exist new file get created. - If the file already exist, the contents are overwritten.
7.	**w+**	-This mode opens a file in both writing and reading. -If file does not exist new file get created. - If the file already exist, the contents are overwritten.
8.	**wb+**	-This mode opens a file in both writing and reading in binary format. -If file does not exist new file get created. -If the file already exist, the contents are overwritten.
9.	**a**	-This mode opens a file in appending mode. -File pointer is placed at the **end of the file**. -if file does not exist, it creates new file.
10.	**ab**	-This mode opens a file in appending mode in binary format. -File pointer is placed at the **end of the file**. -if file does not exist, it creates new file.
11.	**a+**	This mode opens a file for reading and appending mode. -File pointer is placed at the **end of the file**. -if file does not exist, it creates new file.
12.	**ab+**	This mode opens a file for reading and appending mode in binary format. -File pointer is placed at the **end of the file**. -if file does not exist, it creates new file.

Q6. With the help of an example, explain any 3 attributes of file objects.

Ans.

- Python has many inbuilt function and methods to manipulate files.
- To operate a file first we need to open a file using **open()** function. Once a file successfully opened, a file object is returned.
- This file object can be used to access different types of information related to that file.
- This information can be obtained by reading values of specific attributes of the file.
- File has a following file attributes

1. **Fileobj.closed :**
   - ✓ This attribute returns the True value if the file is closed.
   - ✓ And returns False if it is not closed.
2. **Fileobj.mode** :
   - ✓ This attribute returns the access mode with which file has been opened.
3. **Fileobj.name :**
   - ✓ This Attribute returns the name of the opend file.

- Example:

```
fileobj = open("file.txt", "wb")          #opens a binary file in write mode
print ("Name of the file is: ",fileobj.name)
print ("Is file closed" , fileobj.closed)
print ("Mode of opened file is ", fileobj.mode)
```

Output:

```
Name of the file is:  file.txt
Is file closed:  false
Mode of opened file is: wb
```

Q7. Explain the method of closing a file.

OR

Is it mandatory to call the close() method after using the file?

Ans.

- The close() method is used to close the file object.

- Once file object is closed, you cannot read/write file using that object.
- Syntax:

  fileobj.close()
- Program to show the use of open() and close() function

```
fileobj=open("file1.txt","w")
print("File is closed: ", fileobj.closed)
print("File is now being closed ")
fileobj.close()
print("File is closed: ", fileobj.closed)
```

Output:

```
File is closed:  False
File is now being closed
File is closed:  True
```

- If you have used the same object_name for other file, the previous file object is closed.
- But as a good programming habit, we should explicitly close the objects which we have opened.
- The close() function frees up system resources such as file descriptors, file locks, etc, that are associated with the files.
- There is an upper limit for a program to open number of files.
- If the limit exceeds then program may crash or perform unexpected operations. To avoid this we should close the unneeded objects.
- If you didn't closed the file object, python has garbage collector to clean up unreferenced objects.
- But still it is our responsibility to close the file and release the resources consumed by it.

6.5 Reading and Writing Files

Q8. With the help of suitable examples, explain different ways in which you can write data in a file.

OR

Explain write( ) and writeline( ) methods.

Ans.

1. write() method:

- Write method is used to write the string in already opened file.
- Ofcourse the string may include number, special characters and other symbols.
- To write data into the file, the file must be opened in write or append mode.
- While writing data keep in mind write() method does not add newline character (‘\n’) to the end of the string.
- Syntax: fileobj.write(string)
- Program that writes a message into the file

```
fileobj=open("file1.txt","w")
fileobj.write("Hello All, Hope you enjoying Python")
fileobj.close()
print("Data written to the file..")
```

Output:

```
Data written to the file..
```

- File1.txt has been created and content are written to the file.

2. writelines() method:

- This method is used to write a list of strings to a file.

```
fileobj=open("file1.txt","w")
lines=["Hello world\n", "Welcome to Python\n", "Enjoy\n"]
fileobj.writelines(lines)
fileobj.close()
```

```
fileobj=open("file1.txt","r")
print(fileobj.read())
print("Data written to the file..")
```

Output:

```
Hello world
Welcome to Python
Enjoy


Data written to the file..
```

Q9. Explain append( ) method with example.

OR

Write a python program to append data in the file.

Ans.

- Once you have created the file you can always open the file to write more data or append data to the file.
- When you open the file in append mode file pointer is set to end of the file, so that existing data will be kept as it is.
- To append data you must open the file in append mode 'a'.
- Note that if you have opened file in write ('w') mode existing data will be erased and new data is added from starting of the file.
- Python program to append data to the file:

```
fileobj=open("file1.txt","a")
fileobj.write("\nHello world \nWelcome to Python\n")
fileobj.close()
print("Data appended to the file..")
```

Output:

```
Data appended to the file..
```

- If you open the file in append mode and if the file is not exist then new file is created and data writing will begin from start.

Q10. Write a short note on different methods to read data from a file.

OR

Explain read( ) and readline( ) methods.

Ans.

- ➢ The read() method is used to read a string from an already opened fiile.
- ➢ The syntax of read() method is given as-
  fileobj.read([count])
- ➢ In the above syntax, count is an optional parameter which if passed to the read() method specifies the number of bytes to be read from the opened 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(that is till the end of file).
- ➢ Program to print the first 10 characters of the file File1.txt

```
file=open("File1.txt","r")

print(file.read(10))

file.close()
```

- ➢ Note that if you try to open a file for reading that does not exist, then you will get an error as given below-

```
file1=open("file2.txt","r")

print(file2.read())
```

- ➢ read() method returns newline as '\n'.
- ➢ The readline() method is used to read a single line from the file.
- ➢ The method retuns an empty string when the end of the file has been reached.

Example:-

Consider file name is File1.txt and program is to read its contents using the readline() method. The following are the contents of the file-

#File1.txt

```
Hello All,

Hope you are enjoying learning python

We have tried to cover every point in detail to avoid confusion

Happy Reading
```

Program code-

```
file=open("File1.txt","r")
print("First Line:",file.readline())
print("Second Line:",file.readline())
print("Third Line:",file.readline())
file.close()
```

- After reading a line from the file using the readline() method, the control automatically passes to the next line. That is why, when we call the readline() again, the next line in the file is returned.
- The readlines() method is used to read all the lines in the file.
- Program to demonstrate readlines() function

```
file=open("File1.txt","r")
print(file.readlines())
file.close()
```

- An efficient way to display a file is to loop over the file object.
- Program to display the contents of a file.

```
file=open("File1.txt","r")
for line in file:
    print(line)
file.close()
```

Q11. Give the significance of with keyword.

OR

How to open file using with keyword?

Ans.

Contents of file1.txt are as follows:

Hello World

Welcome to the world of Python Programming.

Program 1:

```
with open("file1.txt","rb") as file:
    for line in file:
        print(line)
print("Let's check if the file is closed:",file.close())
```

Output:

Hello World

Welcome to the world of Python Programming.
Let's check if the file is closed: True

Program 2:

```
file=open("file1.txt","rb")
for line in file:
        print(line)
print("Let's check if the file is closed:",file.close())
```

Output:

Hello World

Welcome to the world of Python Programming.
Let's check if the file is closed: False

- In the first code (Program 1), the file is opened using **with keyword.** After the file is used in the for loop, it is automatically closed as soon as the block of code comprising of the for loop is over.

- ➢ But when the file is opened **without the with keyword,** it is not closed automatically. We need to explicitly close the file after using it.
- ➢ **Advantage** of with keyword:
  - o The file is properly closed after it is used even if an error occurs during read or write operation or even when we forget to explicitly close the file.

Q12. Describe split() in file handling with suitable example.

Ans.

- Python allows us to read line(s) 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 1**-To split the line into a series of words and use space to perform the split operation.

```
with open("file1.txt","r") as file:
    line=file.readline()
    words=line.split()
    print(words)
```

Output:
['Hello', 'World', 'Welcome', 'to', 'the', 'world', 'of' ,'Python ','Programming']

- **Program 2**-To perform split operation whenever a comma is encountered.

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

Output:
['Hello World', 'Welcome to the world of Python Programming']

6.6 Directory methods

Q13. Explain the following directory methods:

1. mkdir() 2.getcwd() 3.chdir() 4.rmdir() 5.os.path.join()

Ans.

- A directory is a collection of files where each file may e of a different format.

- Python has various methods in the os module that help programmers to work with directories.
- These methods allow users to create, remove, and change directories.

1. **mkdir()**

- 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.
- The syntax of mkdir() is:

```
os.mkdir("new_dir_name")
```

- Program to create a new directory New Dir in the current directory.

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

- Just check the contents of C:\python34 directory. You will find a new directory name New Dir

2. getcwd()

- The getcwd() method is used to display the current working directory(cwd).
- All files and folders whose path does not exist in the root folder are assumed to be present in the current directory. So to know your cwd is quite important at times and for this getcwd() method is used.
- The syntax of getcwd() is:

```
os.getcwd()
```

- Program to display a current working directory(cwd).

```
import os
print("Current Working Directory: ", os.getcwd())
OUTPUT
Current Working Directory: C:\Python34
```

3. chdir()

- 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.
- The syntax of chdir() is:

```
os.chdir("dir_name")
```

- Program that changes the current directory to our newly created directory -- New Dir

```
import os
print("Current Working Directory: ", os.getcwd())
```

```
os.chdir("New_Dir")
print("After chdir, the current directory is now..... ", end=' ')
print(os.getcwd())
```

OUTPUT

```
Current Working Directory: C:\Python34
After chdir, the current directory is now.....C:\Python34\New Dir
```

- An error message will be displayed if you try to change to a directory that does not exist.

4. rmdir()

- The rmdir() method is used to remove or delete a directory.
- For this, it accepts the 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.
- The syntax of remove() method is:

```
os.rmdir("dir_name")
```

- Program to demonstrate the use of rmdir() method

```
import os
os.rmdir("New_Dir")
print("Directory Deleted.... ")
```

OUTPUT

```
Directory Deleted....
```

- The code given above will remove our newly created directory—New Dir.
  In case, the specified directory is not in the current directory, you should always specify the complete path of the directory as otherwise the method should search for that directory in the current directory only.
- Just check the C:\Python34 folder, the New Dir no longer exists in it. If you try to delete a non-empty directory, then you will get OSError:[WinError 145] The directory is not empty
- If you still want to delete the non-empty directory, use the rmtree() method defined in the shutil module as shown below:

```
import  shutil
shutil.rmtree("Dir1")
```

5. os.path.join()

- In Windows, path names are written using backslash (\ ) but in Unix, Linux, OS X and other operating system they are specified using the forward slash character.

- Python os module has a join() method . When you pass a string value of files and folder names that makes up the path, the os.path.join() method will return a string with a file path that has correct path separators.
- Program that uses os.path.join() method to form a valid file path

```
import os
print(os.path.join("c:", "students", "under graduate", "Btech.docx"))
OUTPUT
c:\students\under graduate\Btech.docx
```

- The above code is executed on Windows, therefore, the output has backward slashes. If the same code was run on another operating system(like Linux), you would have got forward slashes.
- We can use the join() method to form the file path and then pass the file path as an argument to open(). We first open the file in write mode to write some text in it, close the file, and the again open to read its content. The program to print the absolute path of a file using os.path.jon is given below:

```
import os
path= "d:\\="
filename= "First.txt"
abs_path=os.path.join(path,filname)
print("ABSOLUTE FILE PATH=",abc_path)
file=open(abs_path,"w")
file.write("Hello")
file.close()
file=fopen(abs_path,"r")
print(file.read())
OUTPUT
ABSOLUTE FILE PATH=d:\First.txt
Hello
```

6. makedirs()

- The makedirs() methods is used to create more than one folder.
- For example, if you pass string C:\Python34\Dir1\Dir2\Dir3 as an argument to make makedirs() method, the Python will create folder Dir1 in Python34 folder, Dir2 in Dir1 folder, and Dir3 in Dir2 folder.
- For example:

```
import os
os.makedirs(C:\\Python34\\Dir1\Dir2\\Dir3")
```

Q14. Discuss some directory methods present in the OS module.

Ans.

1. **The os. path.abspath() Method:**

- This method uses the string values passed to it to form an absolute path.
- Thus, it is another way to convert a relative path to an absolute path.
- Program to demonstrate the use of **os. path.abspath() Method**

```
import os
print(os. path.abspath("Python\\Strings.docs"))
```

- In the above code, the string Python\\Strings.docx is joined with the current working directory to form an absolute path.

2. **The os.path.isabs(path) Method:**

- This method accepts a file path as an argument and return **True** if the path is an absolute path and **False** otherwise.
- Program to demonstrate the use of **os.path.isabs(path) Method**

```
import os
print("os. path.isabs(\"Python\\String.docx\")= ",
        os. path.isabs("Python\\Strings.docs"))
print("os.path.isabs(\"C:\\Python34\\Python\\String.docx\")=
        ",os.path.isabs("C:\\Python34\\Python\\String.docx"))
OUTPUT
os. path.isabs(\"Python\\String.docx")=False
os.path.isabs(\"C:\\Python34\\Python\\String.docx\")= True
```

3. **The os.path.relpath(path, start) Method:**

- Thi 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.
- Program to demonstrate the use of **os.path.relpath(path, start) Method**

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

4. **The os.path.dirname(path) Method:**

- This method returns a string that includes everything specified in the path (passed as argument to the method) that comes **before** the last slash
- Program to demonstrate the use of **dirname() Method**

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

OUTPUT
os.path.dirname(\"C: \\Python \\Chapters \\First Draft \\Strings.docx \")= C: \Python \Chapters \First Draft

5. **The os.path.basename(path) Method:**

- This method returns a string that includes everything specified in the path (passed as argument to the method) that comes **after** the last slash.
- Program to demonstrate the use of **basename() Method**

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

OUTPUT
os.path.basename(\"C: \\Python \\Chapters \\First Draft \\Strings.docx \")= Strings.docx

6. **The os.path.split(path) Method:**

- This method accept a file path and returns its directory name as well as the base name.
- So it is equivalent to use two separate methods, os.path.dirname() and os.path.basename()
- Program to demonstrate the use of **os.path.split() Method**

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

OUTPUT
os.path.split(\"C: \\Python \\Chapters \\First Draft \\Strings.docx \")=(C: \\Python \\Chapters \\First Draft', Strings.docx)

7. **The os.path.getsize(path) Method:**

- This method returns the size of the file specified in the path argument.
- Program to demonstrate the use of **os.path. getsize () Method**

```
import os
print("os.path.getsize(\"C:\\Python34\\Try.py \")= " , os.path.getsize ( C:\\Python34\\Try.py"))
```

OUTPUT
os.path.getsize(\"C:\\Python34\\Try.py \")= 174

8. **The os.listdir(path) Method:**

- This method returns a list of file name in the specified path.
- Program to demonstrate the use of **os.path. getsize () Method**

```
import os
print("os.path. listdir (\"C:\\Python34\")= ",os.path.listdir(C:\\Python34"))
```

OUTPUT
os.path.listdir(\"C:\\Python34")=['Dir1','DLLs','Doc','File1.txt','include', 'Lib','libs','python.exe']

9. The os.path.exists(path) Method:

- The method as the name suggests accepts a path as an argument and returns **True <u>if the file or folder specified in the path exists</u>** and False otherwise.
- Program to demonstrate the use of **os.path.exits() Method**

```
import os
print("os.path. exists (\"C:\\Python34\Dir1\")=",
os.path.exists(C:\\Python34\Dir1"))
```

OUTPUT
os.path. exists (\"C:\\Python34\Dir1\")= True

10. The os.path.isfile(path) Method:

- The method as the name suggests accepts a path as an argument and returns **True <u>if the path specifies file</u>** and False otherwise
- Program to demonstrate the use of **os.path.isfile() Method**

```
import os
print("os.path. isfile (\"C:\\Python34\Dir1\")=",
os.path.isfile(C:\\Python34\Dir1"))
print("os.path. isfile (\"C:\\Python34\Try.py\")=",
os.path.isfile(C:\\Python34\Try.py"))
```

OUTPUT
os.path. isfile (\"C:\\Python34\Try.py\")= False
os.path. isfile (\"C:\\Python34\Try.py\")=True

11. The os.path.isdir(path) Method:

- The method as the name suggests accepts a path as an argument and returns **True <u>if the path specifies a an existing directory</u>** and False otherwise
- Program to demonstrate the use of **os.path.isdir() Method**

```
import os
print("os.path. isdir(\"C:\\Python34\Dir1\")=",
os.path.isdir(C:\\Python34\Dir1"))
print("os.path. isdir (\"C:\\Python34\Try.py\")=",
os.path.isdir(C:\\Python34\Try.py"))
```

OUTPUT
os.path. isdir (\"C:\\Python34\Try.py\")= False
os.path. isdir (\"C:\\Python34\Try.py\")=True

6.7 Dictionaries

Q15. Explain creating a dictionary. How to access, add, modify and delete items in dictionary.

Ans.

Dictionary

- Dictionary is a data type in python.
- It is unordered collection of items.
- Dictionary is a data structure which stores elements as **key: value** pairs.
  Key: unique, immutable of type strings, & case sensitive.
  Value: Can be any data type, can be duplicated(repeated)
- Each key is separated from its value by a **colon (:).** Consecutive **key: value** pairs are separated by **commas (,).**
- The entire items in a dictionary are enclosed in **curly brackets {}.**

Syntax:

```
Dictionary_name = {key1:value1, key2:value2, … }
```

Example:

```
dict1 = {1:'PPS', 2:'PHY'}
print(dict1)
```

- **Creation of Dictionary**
  - The syntax to create an empty dictionary can be given as:

```
Dict = { }
```

  - The syntax to create a dictionary with key value pair is:

```
Dict = {key1: value1, key2: value2, key3: value3}
```

Example1:

```
Dict = { }
print(Dict)
OUTPUT:
{}
```

```
Example2:
Dict = {1: "PPS", 2: "PHY"}
print(Dict)
OUTPUT:
{1: 'PPS', 2: 'PHY'}
```

- **Accessing values from Dictionary**
- To access values in dictionary, square brackets are used along with **the key** to obtain its value.
- Example:

```
Dict = {1: "PPS", 2: "PHY"}
```

```
print(Dict[1])
OUTPUT:
PPS
```

- Note that if you try to access an item with a key, which is not specified in dictionary, a **KeyError** is generated.

Adding and Modifying (Updating) an item in Dictionary

- Following syntax is used to add a new key: value pair in a dictionary.
- Syntax:

```
Dict[key] = value
```

Example

```
Dict = {1: "PPS", 2: "PHY"}
Dict[3] = "M-I"
print(Dict)
OUTPUT :
{1: 'PPS', 2: 'PHY', 3: 'M-I'}
```

- To modify an entry, just overwrite existing value as shown below:

```
Dict = {1: "PPS", 2: "PHY"}
Dict[2] = "SME"
print(Dict)
OUTPUT:
{1: 'PPS', 2: 'SME'}
```

Deleting items from Dictionary

- One or more items of dictionary can be deleted using **del keyword**.
- To remove all items in dictionary in just one line, **clear()** is used.
- To remove entire dictionary from memory, del statement can be used as **del Dict**.

```
Example:
Dict = {1: "PPS", 2: "PHY", 3: "SME"}
del Dict[2]
print(Dict)
Output:
{1: 'PPS', 3: 'SME'}

Dict1 = {1: "PPS", 2: "PHY", 3: "SME"}
Dict1.clear()
print(Dict1)
Output:
{}
```

Operations on Dictionary: Method	Description
clear()	Remove all items from the dictionary.
copy()	Return a shallow copy of the dictionary.
items()	Return a new view of the dictionary's items (key, value).
keys()	Return a new view of the dictionary's keys.
values()	Return a new view of the dictionary's values