FILE OPERATIONS

PRINTING ON SCREEN: We use the print() function to output data to the standard output device (screen).

SYNTAX: print()
E.G: print("Hello Python")
output: Hello Python

Here any statement written in print will be converted into a string

READING FROM KEYBOARD: We can rread any statement from user uisng raw_input and input keywords:

RAW INPUT: The raw_input([prompt]) function reads one line from standard input and returns it as a string.

Syntax: raw_input()
E.G: a= raw_input("Enter the string value")
print(a)

INPUT: The input([prompt]) function is equivalent to raw_input, except that it assumes the input is a valid Python expression and returns the evaluated result to you.

SYNTAX: input()
E.G: a-input("Enter the string")
print(a)

Every input taken by input keyword will be stored in form of string. So to rad a integer value we must use int keyword before input, so that it reads in form of integer.

E.G:int(input("Enter a va;ue, it will be read in form of integer"))

OPENING AND CLOSING A FILE: Python provides basic functions and methods necessary to manipulate files by default. You can do most of the file manipulation using a file object.

OPEN: Before you can read or write a file, you have to open it using Python's built-in *open()* function. This function creates a file object, which would be utilized to call other support methods associated with it.

Syntax: file_object= open(file_name.,[access_mode],[buffering]]

file_name: The file_name argument is a string value that contains the name of the file that you want to access.

access_mode: The access_mode determines the mode in which the file has to be opened.

buffering: If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file.

Here is a list of the different modes of opening a file −

r: Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.

rb: Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.

r+: Opens a file for both reading and writing. The file pointer placed at the beginning of the file.

rb+: Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.

w: Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

wb: Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

w+: Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing

wb+: Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

a: Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

ab: Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

a+: Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

ab+: Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

CLOSE(): The close() of a file object is used to close a file after which no write operation is going to be performed.

Syntax: fileObject.close();

E.G:
```
str = open("string.txt","wb")
print str.name
str.close()
```

READING AND WRITING FILES: The *file* object provides a set of access methods We would see how to use *read()* and *write()* methods to read and write files.

READ(): The *read()* method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

Syntax: file.read()

E.g:
```
file= open("file.text",r)
print(file.read())
```

The output of that command will display all the text inside the file

Another way to read a file is to call a certain number of characters.
For example, with the following code the interpreter will read the first five characters of stored data and return it as a string:

E.g:
```
file= open("file.text",r)
print(file.read(5))
```

If you want to read a file line by line then you use the *readline()* function.

E.g:
```
file= open("file.text",r)
print(file.readlines())
```

WRITE(): file write method is that it only requires a single parameter, which is the string you want to be written. Evrytime you use write mode it will clear the entire data and write the given data.

Syntax: file= open("file.text",w)

E.g:
```
file= open("file.text",w)
file.write("This is the first line")
file.write("This is the second line")
```

FUNCTIONS:

file.close();Close the file. A closed file cannot be read or written any more.

file.flush():Flush the internal buffer, like stdio's fflush. This may be a no-op on some file-like objects.

file.fileno():Returns the integer file descriptor that is used by the underlying implementation to request I/O operations from the operating system.

file.isatty():Returns True if the file is connected to a tty(-like) device, else False.

file.next():Returns the next line from the file each time it is being called.

file.read([size]): Reads at most size bytes from the file (less if the read hits ` EOF before obtaining size bytes).

file.readline([size]): Reads one entire line from the file. A trailing newline character is kept in the string.

file.seek(offset[, whence]): Sets the file's current position

file.tell():Returns the file's current position

file.truncate([size]): Truncates the file's size. If the optional size argument is present, the file is truncated to (at most) that size.

file.write(str): Writes a string to the file. There is no return value.

file.writelines(sequence): Writes a sequence of strings to the file. The sequence can be any iterable object producing strings, typically a list of strings.