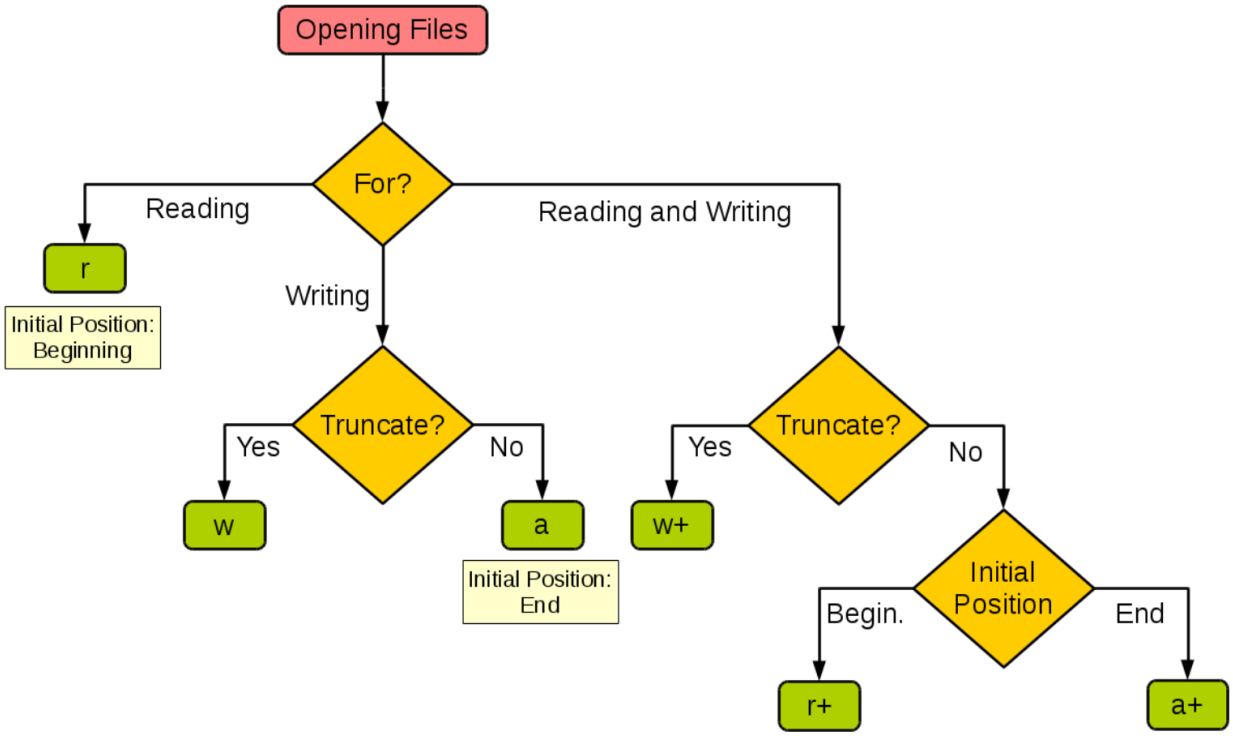
Dharmsinh Desai University, Nadiad
Faculty of technology,
Department of Computer Engineering
Subject : Software Project
Lab Manual

Lab :3 File Handling in Python.

❖ 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.



❖ The Open functions:

To open a file following function is used

file object = open(file_name [, access_mode][, buffering])

file_name	Name of file which we want to open
access_mode	Mode in which we want to open a file
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. If you specify the buffering value as an integer greater than 1, closethen buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

➢ In python, **following access mode** are supported

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.
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.
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.
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.

- **The close() Method:** The close() method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done. Python automatically closes a file when the reference object of a file is reassigned to another file

  Syntax : file_obj.close()

  - **Attribute of File Object:**

file_obj.closed	Returns true if file is closed, false otherwise.
file_obj.mode	Returns access mode with which file was opened.
file_obj.name	Returns name of the file.

  Example:

```
fo=open("file1.txt", 'w+')
print("File Name:",fo.name)
print("Closed?",fo.closed)
print("Opening mode:",fo.mode)
fo.close()
print("Closed?",fo.closed)
```

```
File Name: file1.txt
Closed? False
Opening mode: w+
Closed? True
```

- **The write() Method:** The **write()** method writes any string to an open file. The write() method does not add a newline character ('\n') to the end of the string.

  Syntax : file_obj.write(string)

  Example:

```
fo=open("file1.txt", 'w+')
fo.write("Hello World")
fo.close()
```

- **The read() Method:** The **read()** method reads a string from an open file.

  Syntax: fileObject.read([count])

  - Here, passed parameter is the number of bytes to be read from the opened file. This method starts reading from the beginning of the file and if *count* is missing, then it tries to read as much as possible, maybe until the end of file.

```
fo=open("file1.txt", 'r+')
print(fo.read(9))
fo.close()
```

```
Hello Wor
```

- **File Position:** The *tell()* method tells you the current position within the file; in other words, the next read or write will occur at that many bytes from the beginning of the file.

  - The *seek(offset[, from])* method changes the current file position. The **offset** argument indicates the number of bytes to be moved. The **from** argument specifies the reference position from where the bytes are to be moved.

- If *from* is set to 0, the beginning of the file is used as the reference position. If it is set to 1, the current position is used as the reference position. If it is set to 2 then the end of the file would be taken as the reference position.
- In text files (those opened without a b in the mode string), only seeks relative to the beginning of the file are allowed (the exception being seeking to the very file end with seek(0, 2)).
- If you open the file in binary mode, all data read from or written to the file would be in form of bytes objects, not str. You can declare byte object as ***B = b"DDU"***

```
fo=open("file1.txt", 'r+')
print("Read string is:",fo.read(9))
position=fo.tell()
print("Current position is:",position)
position=fo.seek(0,0)
print("Again read String is:",fo.read(10))
fo.close()
```

```
Read string is: Hello Wor
Current position is: 9
Again read String is: Hello Worl
```

Use of '*with*' operator:

```
fo=open("file1.txt", 'w+')
fo.write("Hello World")
fo.seek(0,0)
str1=fo.read(7)
print(str1)
fo.close()

print("\nUse of 'with' operator")
with open("file1.txt") as fo:
    print("File Content:",fo.read())
print("\nClosed?",fo.closed)
```

```
Hello W

Use of 'with' operator
File Content: Hello World

Closed? True
```

Summary:

	r	r+	w	w+	a	a+
read	+	+		+		+
write		+	+	+	+	+
write after seek		+	+	+		
create			+	+	+	+
truncate			+	+		
position at start	+	+	+	+		
position at end					+	+

Exercise:

(1) Write a program to read a text file and find unique words with their occurrences and display tabular form the analysis.

(2) Write a program which reads a sample C Program file as input, stores the content to the another file without any single line comments within.

Know that C Comments style:
// single line comment

For question 3 and 4 refer to **https://docs.python.org/3/library/csv.html#examples**

(3) Store students' records to a file named students.csv. Know that record contains following information <student_id, name, age, percentage>.
'.csv' format contains comma-separated values of record line by line.

(4) Read file students.csv and display tabular form students records. Also, count and at the end display total how many students are recorded in the file (count).

(5) Learn handling file I/O errors using exception handling in Python.

https://docs.python.org/3/tutorial/errors.html#exceptions

Run the given python program in below different scenarios and check output to understand exception handling:

- **if you are not using jupyter notebook then using terminal**
  - without having *myfile.txt* in current location.
  - Create datafile using command 'echo "1 2 3" > myfile.txt'
  - Overwrite file content 'echo "123" > myfile.txt'
  - chmod u-r myfile.txt

- **If you are using jupyter notebook then you can use following cases**:
  - without having *myfile.txt* in current location(from wherever you have started jupyter notebook)
  - Create datafile using below code:

```
with open("myfile.txt",'w+') as f1:
    f1.write("1 2 3")
    f1.seek(0,0)
    print(f1.read())
```

```
1 2 3
```

-Overwrite file content using below code:

```
with open("myfile.txt",'w+') as f1:
    f1.write("123")
    f1.seek(0,0)
    print(f1.read())
```

```
123
```

'''

```
import sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
    print(i)
except OSError as err:
    print("OS error: {0}".format(err))
except ValueError:
    print("Could not convert data to an integer.")
except:
    print("Unexpected error:", sys.exc_info()[0])
    raise
```

'''

(6) Also, practice JSON data handling using python.

https://docs.python.org/2/tutorial/inputoutput.html#saving-structured-data-with-json