Computational Thinking with Programming

@cse_bennett @csebennett

Lecture Contents

- **File Handling:**
    - File Opening and Creating and Closing
    - Reading file
    - Writing file
    - Deleting file and Directory
    - File Positioning

What Is a Text File ?

.jpg
.png

Hello
Hi

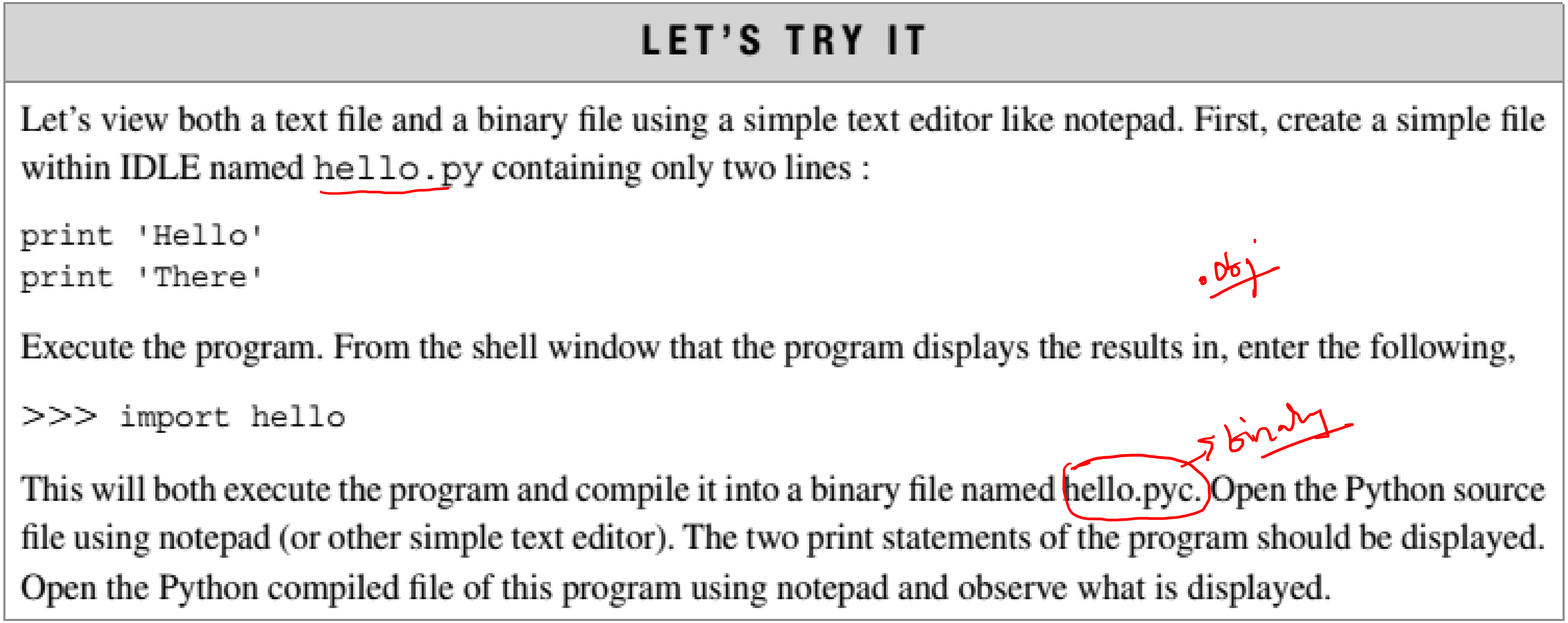
print("Hello\n Hi")

simple.py → .obj .pyc → binary

- **File** is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory.

- A **text file** is a file containing characters, structured as lines of text.
- In addition, text files also contain the nonprinting newline character, \n, to denote the end of each text line.

- A **binary file** is a file that is formatted in a way that only a computer program can read.



LET'S TRY IT

Let's view both a text file and a binary file using a simple text editor like notepad. First, create a simple file within IDLE named `hello.py` containing only two lines :

```
print 'Hello'
print 'There'
```

.obj

Execute the program. From the shell window that the program displays the results in, enter the following,

```
>>> import hello
```

→ binary

This will both execute the program and compile it into a binary file named hello.pyc. Open the Python source file using notepad (or other simple text editor). The two print statements of the program should be displayed. Open the Python compiled file of this program using notepad and observe what is displayed.

Note: we will be going to work mainly with txt files. Python **supports file handling in very easy and short manner compared to other languages.**

File Operations

.txt

- Python allows the user to handle the file by performing following operations on it:
  - Opening or creating a file
  - Reading a file
  - Writing into a file
  - Appending into a files
  - Close the file
- Python provides several functions for performing the above file operations

Opening Text Files: open( ) function

- All files must first be opened before they can be used. In Python, when a file is opened, a file object is created that provides methods for accessing the file.

- The open() is a key function for working with files. This function takes two parameters; *filename*, and *mode.*

open (filename)   textfile

- **Syntax:** *File_pointer = open(<file_name/path>, <access_mode>)*

- ***Note:*** *The value of* ***access_mode*** *depends on the which you want to perform with file. There are different modes while opening a file.*

File Access Modes

S. No.	Mode Value	Mode Name	Description
1.	"r"	Read	**Default value.** Opens a file for reading, error if the file does not exist.
2.	"w"	Write	Opens a file for writing, creates the file if it does not exist
3.	"a"	Append	Opens a file for appending, creates the file if it does not exist
4.	"x"	Create	Creates the specified file, returns an error if the file exists
5.	“r+"	Read + write	For both reading and writing
6.	"t"	Text	Default value. Text mode
7.	"b"	Binary	Binary mode (e.g. images)

File Access Modes

S. N.	Mode	Description
1.	"r"	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
2.	"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.
3.	"r+"	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
4.	"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.
5.	"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.
6.	"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.
7.	"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.

The *file* Object Attributes

- Once a file is opened and you have one *file* object, you can get various information related to that file.

file object f = open(" my file.txt")

pointer

S. No.	Attribute	Description
1	**file.closed** f.closed	Returns true if file is closed, false otherwise.
2	**file.mode**	Returns access mode with which file was opened.
3	**file.name**	Returns name of the file.

The *file* Object Attributes: Example

- Once a file is opened and you have one *file* object, you can get various information related to that file.

Example:

```
f = open("foo.txt", "wb")
print ("Name of the file: ", f.name)
print ("Closed or not : ", f.closed)
print ("Opening mode : ", f.mode)
```

Output:

```
Name of the file: foo.txt
Closed or not : False
Opening mode : wb
```

File Reading: Example

fil = open (" - - - " , - -)

```
#Reading Myfile.txt

File_ptr = open(“Myfile.txt", "r")
print(File_ptr.read())
print(File_ptr.read(5))
File_ptr.close()
```

5

Myfile.txt

Hello! Welcome to Myfile.txt
Reading the file is very easy.
Now you can read a file from your hard disk
Good Luck!

- The open() function returns a file object, which has a read() method for reading the content of the file.
- By default the read() method returns the whole text, but you can also specify how many characters you want to return.
- It is a good practice to always close the file when you are done with it.

Note: It is enough to specify only the name of the file while opening a file for reading.

File Reading: Another Example

- We can also split lines using split() function. This splits the variable when space is encountered.
- ***You can also split using any characters as we wish.***

readline( )

```
# Python code to illustrate split() function
   file = open("Testfile.txt","r")
with open(“Testfile.txt", "r") as file:
    data = file.readlines()
    for line in data:
        word = line.split()
        print(word)
```

```
# Contents of Testfile.txt:

Hello this test file.   \n
It is splitting the lines.
```

```
Output:

['Hello', 'this', 'test', 'file.’]
['It', 'is', 'splitting', 'the', 'lines.']
```

File Reading: Example

```
#Reading Myfile.txt
File_ptr = open("Myfile.txt", "r")
print(File_ptr.readline())
File_ptr.close()
```

Output:

```
Hello! Welcome to Myfile.txt
```

```
#Reading Myfile.txt
File_ptr = open(“Myfile.txt", "r")
print(File_ptr.readline())
print(File_ptr.readline())
File_ptr.close()
```

Output:

```
Hello! Welcome to Myfile.txt
Reading the file is very easy.
```

- *You can return one line by using the readline() method.*
- *By calling readline() two times, you can read the first two lines.*

How the file having large number of lines can be read ?

File Reading: Example

S = [1, 2, 3, 4]

for i in S:

- **By looping through the lines of the file, you can read the whole file, line by line.**

```
#Reading Myfile.txt

f = open(“Myfile.txt", "r")
for x in f:
  print(x)
f. close()
```

Myfile.txt

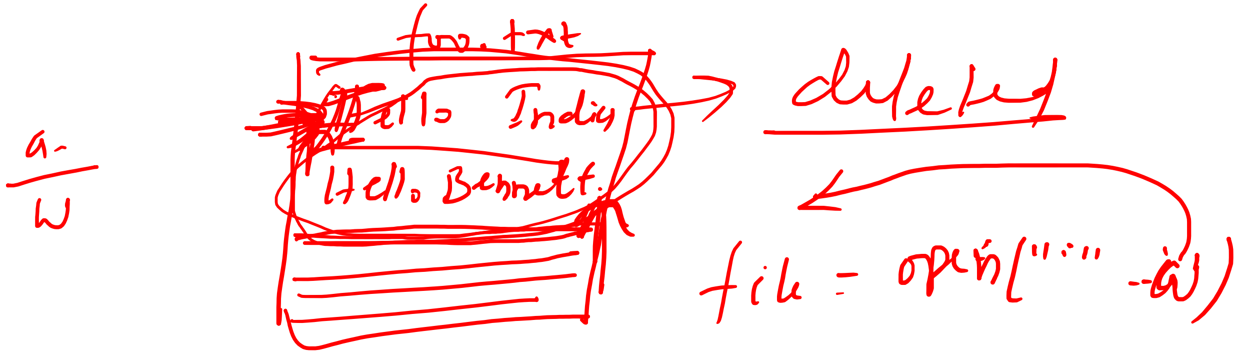
```
Hello! Welcome to Myfile.txt
Reading the file is very easy.
Now you can read a file from your hard disk
Good Luck!
```

Output:

```
Hello! Welcome to Myfile.txt
Reading the file is very easy.
Now you can read a file from your hard disk
Good Luck!
```

Note: The file object also provides a set of access methods to write files.

File Writing



- There are following ways to write into a file:
  - **Writing to an existing file:** 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
  - **Creating new file:** To create a new file in Python, use the open() method, with one of the following parameters.
    - "x" - Create - will create a file, returns an error if the file exist
    - "a" - Append - will create a file if the specified file does not exist
    - "w" - Write - will create a file if the specified file does not exist
- The ***write()*** method is used to write strings to a file.

Writing to an Existing File: Appending

#Example: Open the file “Myfile2.txt" and append content to the file

w

```
f = open(“Myfile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the
appending:
f = open(“Myfile2.txt", "r")
print(f.read())
f.close()
```

Myfile2.txt

```
Hello! Welcome to Myfile2.txt
This file is for testing purposes.
Good Luck!
```

n

```
Output:

Hello! Welcome to Myfile2.txt
This file is for testing purposes.
Good Luck!Now the file has more
content!
```

Writing to an Existing File: Overwriting

#Example: Open the file “Myfile2.txt" and overwrite the content

```
f = open(“Myfile2.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()

#open and read the file after the appending:
f = open(“Myfile2.txt", "r")
print(f.read())
f.close()
```

# Myfile2.txt

```
Hello! Welcome to Myfile.txt
This file is for testing purposes.
Good Luck!
```

Output:

```
Woops! I have deleted the content!
```

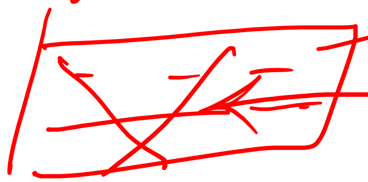
Create a New File

```
#Example: Create a file called "myfile3.txt"

f = open("myfile3.txt", "x")
f.write(“It is written in this file.”)
f.close()

#open and read the file after the appending:
f = open(“Myfile3.txt", "r")
print(f.read())
f.close()
```

my file.txt



Important

Result: a new empty file is created!

f = open(" my file. txt", 'w')

'x' → error

```
# Myfile3.txt

It is written in this file.
```

```
Output:

It is written in this file.
```

```
#Example: Create a new file if it does not exist"

f = open("myfile4.txt", "x")
f.write(“The contents are written using W mode.”)
f.close()
```

```
# Myfile4.txt

The contents are written using W mode.
```

Writing Multiple Lines into a File

Example1:

```
with open(“Test.txt",'w') as fp:
   fp.write(“My first file\n")
   fp.write("This file\n\n")
   fp.write("contains three lines\n")
   fp.close()
```

#Contents in Test.txt:

```
My first file
This file

contains three lines
```

Example2:

```
fp = open(“Test.txt",'w')
fp.write(“My first file\n")
fp.write("This file\n\n")
fp.write("contains three lines\n")
fp.close()
```

Note: By looping, you can write the large number of lines into the file.

OS.path.ex

File Deleting and Renaming

- To delete a file, you must import the OS module, and run its os.remove() function.

Example: Remove the file "demofile.txt":

```
import os
os.remove("demofile.txt")
```

- To avoid getting an error, you might want to check if the file exists before you try to delete it.

Example: Check if file exists, *then* delete it:

```
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

- The *rename()* method takes two arguments, the current filename and the new filename.

Example: # Rename a file from test1.txt to test2.txt

```
import os
os.rename( "test1.txt", "test2.txt" )
```

Create and Delete Folder/Directory

"tmp/test/" Hello.txt

- To delete an entire folder, use the os.rmdir() method:

```
#Example: Remove the folder "myfolder":

import os
os.rmdir("myfolder")

# This would remove "/tmp/test" directory.
os.rmdir( "/tmp/test" )
```

Note: You can only remove empty folders.

- You can use the os.*mkdir()* method of the **os** module to create directories in the current directory.

```
# Create a directory "test"

import os
os.mkdir("test")
```

Change and Get Current Directory

- *You can use the os.chdir() method to change the current directory.*

- *The os.getcwd() method displays the current working directory.*

```
import os

# Changing a directory to "/home/newdir"
os.chdir("/home/newdir")

# This would give location of the current directory
os.getcwd()
```

File Positions

- **Python provides the following file positioning functions:**

- ***tell()* method:** Tells you the current position of the file pointer within the file.
- In other words, the next read or write will occur at that many bytes from the beginning of the file.

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

File Positions: Example

```
# Open a file. Assume file is already created
fp = open(“Test.txt", "r+")
str = fp.read(10)
print ("Read String is : ", str)

# Check current position
position = fp.tell()
print ("Current file position : ", position)

# Reposition pointer at the beginning once again
position = fp.seek(0, 0);
str = fp.read(10)
print ("Again read String is : ", str)

# Close opend file
fp.close()
```

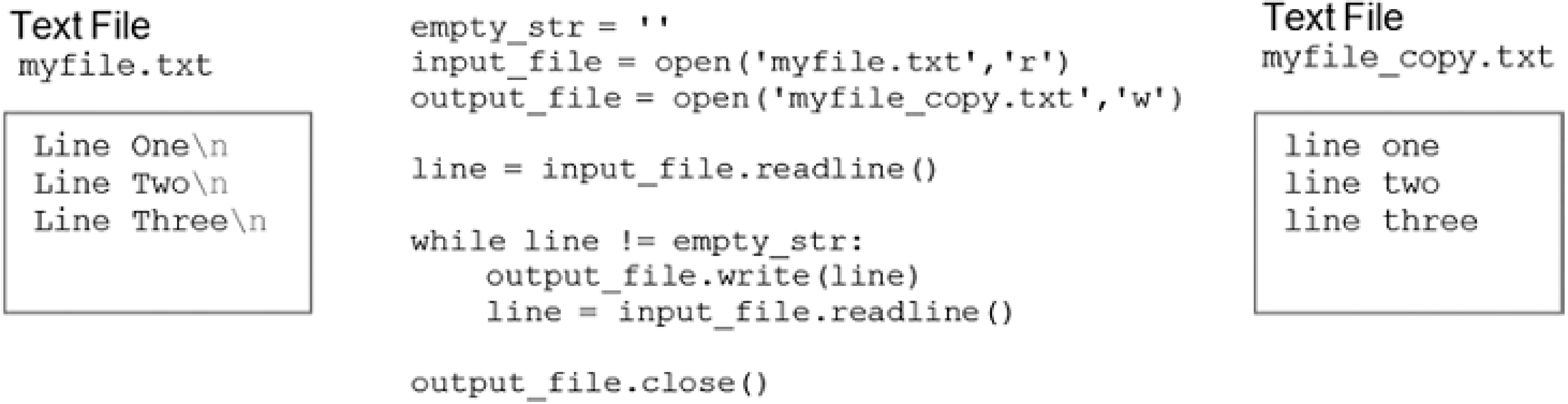
```
# Contents of Test.txt:

Python is a Programming
Language.
```

```
Output:
Read String is : Python is
Current file position : 10
Again read String is : Python is
```

Exercise1: Copying File

- Write a program to copy contents of file myfile.txt to myfile_copy.txt



Text File
myfile.txt

```
Line One\n
Line Two\n
Line Three\n
```

```
empty_str = ''
input_file = open('myfile.txt','r')
output_file = open('myfile_copy.txt','w')

line = input_file.readline()

while line != empty_str:
    output_file.write(line)
    line = input_file.readline()

output_file.close()
```

Text File
myfile_copy.txt

```
line one
line two
line three
```

Exercise 2

- **Write a code to read file and give you summary of the file.**

Number of words

Number of character (with space)

Number of character (without space)

Frequency of character

Number of paragraph

MCQs

1. Indicate which of the following reasons an *IOErro*r (exception) may occur when opening a file.
   a) Misspelled file name
   b) Unmatched uppercase and lowercase letters
   c) File not found in directory searched

2. Which one of the following is true?
   a) When calling the built-in *open* function, a second argument of 'r' or 'w' must always be given
   b) When calling the built-in *open* function, a second argument of 'r' must always be given when opening a file for reading.
   c) When calling the built-in *open* function, a second argument of 'w' must always be given when opening a file for writing.

3. Only files that are written to need to be opened first.
   a) True b) False

4. Which one of the following is true?
   a) There is more chance of an I/O error when opening a file for reading.
   b) There is more chance of an I/O error when opening a file for writing.

5. The *readline()* method reads every character from a text file up to and including the next newline character '\n'.
   a) True b) False

6. It is especially important to close a file that is open for writing.
   a) True b) False

MCQs: Answers

1. Indicate which of the following reasons an *IOError* (exception) may occur when opening a file.
   - **a) Misspelled file name**
   - b) Unmatched uppercase and lowercase letters
   - **c) File not found in directory searched**

2. Which one of the following is true?
   - a) When calling the built-in *open* function, a second argument of 'r' or 'w' must always be given
   - b) When calling the built-in *open* function, a second argument of 'r' must always be given when opening a file for reading.
   - **c) When calling the built-in *open* function, a second argument of 'w' must always be given when opening a file for writing.**

3. Only files that are written to need to be opened first.
   - a) True b) **False**

4. Which one of the following is true?
   - **a) There is more chance of an I/O error when opening a file for reading.**
   - b) There is more chance of an I/O error when opening a file for writing.

5. The *readline()* method reads every character from a text file up to and including the next newline character '\n'.
   - **a) True** b) False

6. It is especially important to close a file that is open for writing.
   - **a) True** b) False

Thank You

?