**Experiment Number – 18**

**Title – Write a function in Python to count and display the total number of words in a text file.**

**Theory-**

A file is a container in computer storage devices used for storing data.Python too supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, but like other concepts of Python, this concept here is also easy and short. Python treats files differently as text or binary and this is important. Each line of code includes a sequence of characters and they form a text file. Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character. It ends the current line and tells the interpreter a new one has begun.

When we want to read from or write to a file, we need to open it first. When we are done, it needs to be closed so that the resources that are tied with the file are freed.

Hence, in Python, a file operation takes place in the following order:

1. Open a file
2. Read or write (perform operation)
3. Close the file

**Opening Files in Python**

The **open()** Python method is the primary file handling function. The basic syntax is:

file\_object = open('file\_name', 'mode')

The **open()** function takes two elementary parameters for file handling:

1. The **file\_name** includes the file extension and assumes the file is in the current working directory. If the file location is elsewhere, provide the absolute or relative path.

2. The **mode** is an optional parameter that defines the file opening method. The table below outlines the different possible options:

|  |  |
| --- | --- |
| **Sr.No.** | **Modes & 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 | **rb+** - Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file. |
| 5 | **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. |
| 6 | **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. |
| 7 | **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. |
| 8 | **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. |
| 9 | **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. |
| 10 | **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. |
| 11 | **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. |
| 12 | **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 file Object Attributes**

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

Here is a list of all attributes related to file object –

|  |  |
| --- | --- |
| **Sr.No.** | **Attribute & Description** |
| 1 | **file.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. |
| 4 | **file.softspace**  Returns false if space explicitly required with print, true otherwise. |

## 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. It is a good practice to use the close() method to close a file.

### Syntax

fileObject.close()

## Reading and Writing Files

The file object provides a set of access methods to make our lives easier. We would see how to use read() and write() methods to read and write files.

## The write() Method

The write() method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.The write() method does not add a newline character ('\n') to the end of the string.

### Syntax

fileObject.write(string)

Here, passed parameter is the content to be written into the opened file.

## The read()/ readlines() Method

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

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.

fileObject.readlines([count])

Here, passed parameter is the number lines 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.

**File Positions**

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, it means use the beginning of the file as the reference position and 1 means use the current position as the reference position and if it is set to 2 then the end of the file would be taken as the reference position.

### Advantages:

* **Versatility**: File handling in Python allows you to perform a wide range of operations, such as creating, reading, writing, appending, renaming, and deleting files.
* **Flexibility**: File handling in Python is highly flexible, as it allows you to work with different file types (e.g. text files, binary files, CSV files, etc.), and to perform different operations on files (e.g. read, write, append, etc.).
* **User**–**friendly**: Python provides a user-friendly interface for file handling, making it easy to create, read, and manipulate files.
* **Cross-platform**: Python file handling functions work across different platforms (e.g. Windows, Mac, Linux), allowing for seamless integration and compatibility.

### Disadvantages:

* **Error-prone:** File handling operations in Python can be prone to errors, especially if the code is not carefully written or if there are issues with the file system (e.g. file permissions, file locks, etc.).
* **Security risks**: File handling in Python can also pose security risks, especially if the program accepts user input that can be used to access or modify sensitive files on the system.
* **Complexity**: File handling in Python can be complex, especially when working with more advanced file formats or operations. Careful attention must be paid to the code to ensure that files are handled properly and securely.
* **Performance**: File handling operations in Python can be slower than other programming languages, especially when dealing with large files or performing complex operations.

Exercise –

1. Write a Python program to reverse the content of a file and store it in another file
2. Write a python Copy all the content of one file to another file in uppercase
3. Write a python program to Append content of one text file to another