File Handling in Python (Main concept)

File objects (also called file-handle) are used to to read and write data to a file on disk. The file object is used to obtain a reference to the file on disk and open it for a number of different tasks.

When you use file **open()**, Python stores the reference of mentioned file in the file-object. A file-object of Python is stream of bytes where the data can be read byte by byte or line by line or collectiely.

You can also use **“with”** statement as below

```
    with open(<filename>, <filemode>) as <filehandle>:
        <file manipulation statement>
```

The advantage of using a **with statement** is that it is guaranteed to close the file no matter how the nested block exists. You don’t need to use close() with it.

Close() : breaks the link of file-object and the file on disk. After close(), no operations can be performed on that file through file-object.

Note: open() is built-in-function while close() is a method used with file-handle object.

File Access Modes

Text File Mode	Binary File Mode	Description
‘r’	‘rb’	Read only
‘w’	‘wb’	Write only
‘a’	‘ab’	append
‘r+’	‘rb+’	Read & write
‘w+’	‘wb+’	Write & read
‘a+’	‘ab+’	Write & read

Reading From Text Files

1. read() file-object.read(n) reads at most n bytes ; if no n specified , reads the entire file

2. readline() file-object.readline(n) reads at most n bytes ; if no n specified , reads one line

3. readlines() file-object.readline(n) reads all line & return them in a list

Writing onto Text Files

1. write() file-object.write(str) writes str to file referenced by file-object

2. writelines() file-object.writelines(L) writes all strings in list L as lines to file referenced by file-object

Note: Make sure to use close() function on file-object after you have finished writing as sometimes, the content remains in memory buffer and to force-write the content on file and closing the link of file-object from file, close() is used.

However to force-write the contents of buffer onto storage , you can also use **file-object.flush()** function.

Significance of File Pointer in File Handling
Every file maintains a file pointer which tells the current position in the file where writing or reading will take place.

File modes & opening position of file-pointer

File Modes	Position
r, rb, r+, rb+, r+b	begining of file
w, wb, w+, wb+, w+b	begining of file
a, ab, a+, ab+, a+b	At end of file if exists otherwise creates new file

Note: The **file-object.tell()** function returns the current position of file pointer in an open file. And the **file-object.seek()** function places the file pointer at the specified by in an open file.

Working With Binary Files
In order to work with binary files you have to import python’s “pickle” module. And then , you may use **dump()** and **load()** methods of pickle module to write and read from an open binary file respectiely.

Note: There are two similar functions **dumps()** and **loads()** of pickle module.To know more please visit https://docs.python.org/3/library/pickle.html

Working With CSV Files
To work with csv files we used inbuilt “csv” module, use reader & writer methods as below

```
import csv
reader_obj = csv.reader(file-object)
writer_obj = csv.writer(file-object)
```

To know more on file handling please visit
https://www.datacamp.com/community/tutorials/reading-writing-files-python
https://docs.python.org/3/library/csv.html
https://openpyxl.readthedocs.io/en/stable/

Generate/Output csv with django
https://docs.djangoproject.com/en/3.1/howto/outputting-csv/

Generate/Output pdf with django
https://docs.djangoproject.com/en/3.1/howto/outputting-pdf/