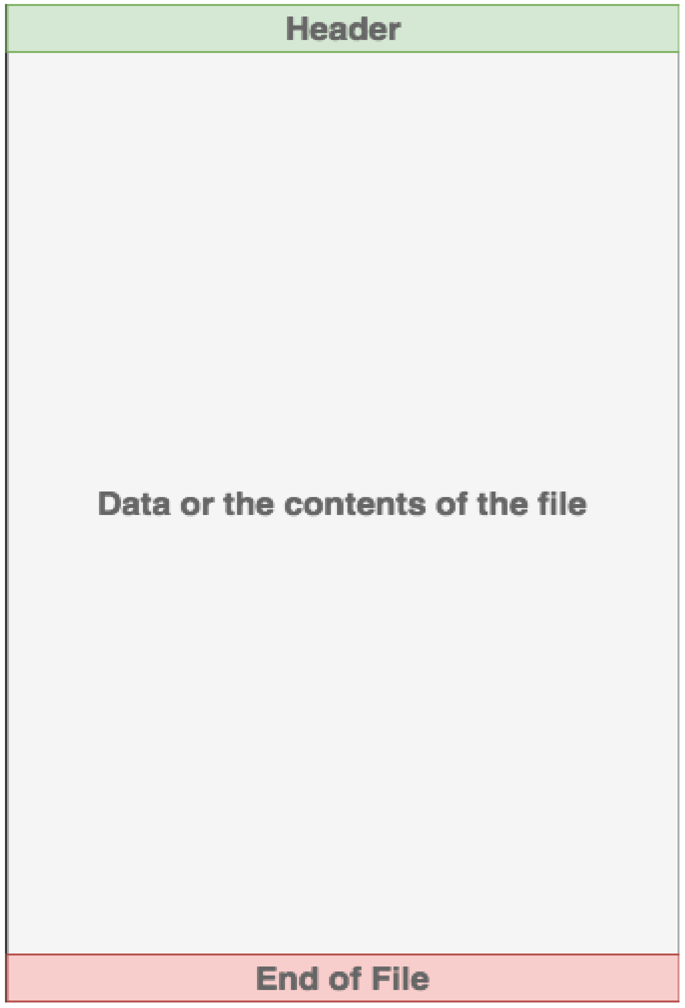
Files Manipulation

- **Header:** metadata about the contents of the file (file name, size, type, and so on)
- **Data:** contents of the file as written by the creator or editor
- **End of file (EOF):** special character that indicates the end of the file



Opening and closing a text file

- Before we can read the contents of the file we must tell Python which file we are going to work with and what we will be doing with the file
- This is done with the open() function
- open() returns a "file handle" - a variable used to perform operations on the file

Syntax:

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

A list of the different modes of opening a file:

Modes	Description
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 will be at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer will be 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.

A list of the different modes of opening a file:

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.

The *file* object atrributes:

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:

Attribute

- file.closed
  - Returns true if file is closed, false otherwise.
- file.mode
  - Returns access mode with which file was opened.
- file.name
  - Returns name of the file.
- file.softspace
  - Returns false if space explicitly required with print, true otherwise.

Reading from text files

- Files have to be opened before a program can read (write) data from it

Syntax

```
any_file = open( “sample1.txt”, “r” )
```

File object$\rightarrow$ <_io.TextIOWrapper name='sample1.txt' mode='r' encoding='cp1252'>

- Files also have to be closed when a program is finished with it

Syntax

```
any_file.close()
```

Reading From Text Files

Prepares the file for reading:

A. Links the file variable with the physical file (references to the file variable are references to the physical file).

B. Positions the file pointer at the start of the file.

Format:

<file variable> = open(*<file name>,* "r")

Example:

```
inputFile = open("data.txt", "r")
```

OR

```
filename = input("Enter name of input file: ")
inputFile = open(filename, "r")
```

Reading From Text Files

- Reading characters from a line
  - Text files are often line oriented and your program may have to read and process one line at a time
  - The method read_line() is used to read characters ***from the current line only***
  - Example

```
anystring = any_file.readline(1)
anystring = any_file.readline(5)
anystring = any_file.readline()
```

Reading From Text Files

- Reading all lines into a list
  - Another way to work with lines from a file is to read each line (string) into a list
  - The method read_lines() is used to read all the lines from a text file and store them into a list of lines (strings)
  - Example

```
any_list = any_file.readlines()
```

Reading From Text Files

- Looping through a text files
  - Text files are a type of sequence delimited by lines
  - Python programs can also read and process lines from text files by using iteration
  - Example

```
for line in any_file:
    print line
```

Writing To A Text File

Format

<name of file variable> = open(*<file name>*, "w")

Example

```
outputFile = open(“data.txt", "w")
```

OR

```
outputFileName = input("Enter the file name")
outputFile = open(outputFileName, "w")
```

Writing To A Text File

- Writing strings to a file
  - There are several functions for writing data to a file
  - To write a single string to a text file use the write() method
  - Example

```
any_file.write("This is a test…\n")
any_file.write("This is only a test\n" )
```

    - Note both strings could have been concatenated together into one string and have the same result using one write statement

Writing To A Text File

- Writing a list of strings to a file
  - The writelines() method is the complement function to readlines()
  - It takes a list of strings and prints them to a file
  - Example

    ```
    any_file.writelines( any_list )
    ```

    - The newline characters must be embedded in each string for proper formatting (as needed)

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.