4. Reading and Writing Files

❖Reading from a file using read(), readline(), readlines().

f.read() :

- Reads the entire file into a **single string**.
- Great when you need to process or search the whole file at once.
- **Caution**: reading a huge file may use a lot of memory.

f.readline() :

- Reads **one line** from the current file position, including the ending newline \n, then moves the cursor to the next line.
- Optional size argument limits the bytes read.
- Ideal for **stepwise** logic: e.g., process a header line, then loop through the rest.
- Continues returning an empty string when end-of-file is reached.

f.readlines() :

- Reads **all lines** into a **list of strings**, each representing a line (with newline chars included)
- Optional hint parameter reads until the total bytes exceed hint, finishing at the end of a line.
- Handy when you want random-access to lines or easy slicing .

❖Writing to a file using write() and writelines().

f.write() :

- **Purpose**: Write a *single string* to the file.
- **Returns**: The number of characters written.
- **Important**: Does *not* automatically add newline characters; you must include them manually:

Syntax:

with open("example.txt", "w", encoding="utf-8") as f:

```
count = f.write("Hello, world!\n")  # newline must be added explicitly
```

f.writelines() :

- **Purpose**: Write each string from an iterable (list, tuple, generator, etc.) to the file.
- **Returns**: None — doesn't report characters written.
- **Important**: Also **does not** add newline characters automatically. You must include them in the strings:

Syntax:

```
lines = ["First line\n", "Second line\n", "Third line\n"]
with open("example.txt", "w", encoding="utf-8") as f:
    f.writelines(lines)
```