

Day 30



File IO in Python:

What is File I/O?

File I/O means:

- Input → Reading data from a file
- Output → Writing data to a file

Files allow data to be stored permanently, unlike variables (temporary).

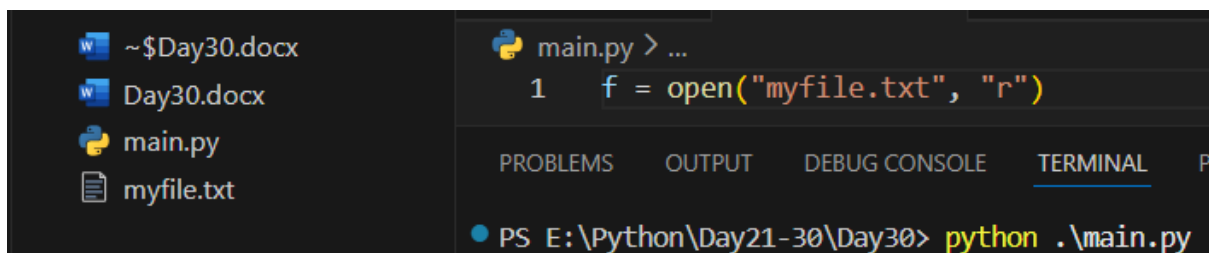
Types of Files

Python mainly works with:

1. Text files → .txt, .csv, .log
2. Binary files → .jpg, .pdf, .exe

Operation	Method
Open file	open()
Read	read(), readline(), readlines()
Write	write(), writelines()
Close	close()
Best practice	with open()

Example: opening the file.

A screenshot of a code editor interface. On the left, a file explorer shows a project structure with files: ~\$Day30.docx, Day30.docx, main.py, and myfile.txt. The main editor area shows a Python file named main.py with a single line of code: f = open("myfile.txt", "r"). Below the code editor, there are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The TERMINAL tab is active, showing a command prompt where the command 'python .\main.py' has been executed from the directory 'E:\Python\Day21-30\Day30'.

Example: reading the content of myfile.txt

```
1 f = open("myfile.txt", "r")
2 text = f.read()
3 print(text)
4 f.close()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● PS E:\Python\Day21-30\Day30> python .\main.py
Aditya is a boy.
```

Common File Modes

Mode	Meaning
r	Read (default)
w	Write (overwrite)
a	Append
x	Create new file
r+	Read + Write
b	Binary mode
t	Text mode (default)

Example: if we open a file which doesn't exist in 'w' mode, then that file automatically gets created.

~\$Day30.docx
Day30.docx
main.py
myfile.txt
myfile2.txt

```
main.py > ...
1 f = open("myfile2.txt", "w")
2
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● PS E:\Python\Day21-30\Day30> python .\main.py
PS E:\Python\Day21-30\Day30>
```

Example: writing in the file.

```
1 f = open("myfile2.txt", "w")
2 f.write("Adii")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

- PS E:\Python\Day21-30\Day30> python .\main.py
- PS E:\Python\Day21-30\Day30> cat .\myfile2.txt
Adii

Example: appending in the file.

```
1 f = open("myfile2.txt", "a")
2 f.write("Aditya")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

- PS E:\Python\Day21-30\Day30> python .\main.py
- PS E:\Python\Day21-30\Day30> cat .\myfile2.txt
AdiiAditya

Example: the number of times we run the code, the number of times it get appended.

```
main.py > ...
1 f = open("myfile2.txt", "a")
2 f.write("Aditya")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

- PS E:\Python\Day21-30\Day30> python .\main.py
- PS E:\Python\Day21-30\Day30> cat .\myfile2.txt
AdiiAditya
- PS E:\Python\Day21-30\Day30> python .\main.py
- PS E:\Python\Day21-30\Day30> cat .\myfile2.txt
AdiiAdityaAditya

Example: **Using with Statement (Best Practice)**: Automatically closes the file.

```
1 with open("data.txt", "r") as file:
2     content = file.read()
3     print(content)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

- PS E:\Python\Day21-30\Day30> python .\main.py
Aditya

read(), readlines() and other methods:

Before reading, a file must be opened in read mode (r).

read()

- Reads entire file content as one single string
- Can also read a specific number of characters

Example: basic reading of the file.

```
1 file = open("sample.txt", "r")
2 content = file.read()
3 print(content)
4 file.close()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● PS E:\Python\Day21-30\Day30> python .\main.py
Aditya
Utsav Kumar
```

Example: reading the first ten characters.

```
2 file = open("sample.txt", "r")
3 content = file.read(10) # Reads first 10 characters
4 print(content)
5 file.close()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● PS E:\Python\Day21-30\Day30> python .\main.py
Aditya
Ut
```

readline()

- Reads one line at a time
- Cursor moves to the next line automatically

Example: basic use of readline()

```
7 file = open("sample.txt", "r")
8 line1 = file.readline()
9 line2 = file.readline()
10 print(line1)
11 print(line2)
12 file.close()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Python\Day21-30\Day30> python .\main.py
Aditya

Utsav Kumar
```

readlines()

- Reads all lines at once
- Returns a list of strings

Example: a basic example of readlines()

```
14 file = open("sample.txt", "r")
15 lines = file.readlines()
16 print(lines)
17 file.close()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Python\Day21-30\Day30> python .\main.py
['Aditya \n', 'Utsav Kumar\n', 'Pihu']
```

Example: **File Cursor Methods: tell()** - Returns current cursor position.

```
19 with open("sample.txt", "r") as file:
20     print(file.tell())
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Python\Day21-30\Day30> python .\main.py
0
```

Example: **seek()** -> Moves cursor to a specific position.

```
23 with open("sample.txt", "r") as file:
24     file.seek(0) # Move to start
25     print(file.read())
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● PS E:\Python\Day21-30\Day30> python .\main.py
Aditya
Utsav Kumar
Pihu
```

Comparison Table

Method	Returns	Reads	Memory Usage
read()	String	Entire file	High
read(n)	String	n characters	Low
readline()	String	One line	Low
readlines()	List	All lines	High
File iteration	String	One line	Very Low

Summary:

- read() → Reads the entire file (or given number of characters) and returns a single string.
- readline() → Reads one line at a time and returns it as a string.
- readlines() → Reads all lines at once and returns them as a list of strings.
- File iteration (for line in file) → Reads the file line-by-line efficiently using minimal memory.
- tell() → Returns the current position of the file cursor.
- seek(pos) → Moves the file cursor to the specified position.
- write() → Writes a string to the file and returns the number of characters written.
- writelines() → Writes a list of strings to the file without adding newlines automatically.
- close() → Closes the file and frees system resources.
- with open() → Automatically opens and closes the file safely (best practice).

--The End--