📃 Reading and Writing Files in Python

In professional Python programming, **file handling** is one of the most essential skills. Files store data permanently — unlike variables that disappear when a program ends.

Python provides built-in functions to **open**, **read**, **write**, and **close** files efficiently.

⚙️ 1. Opening a File

Before reading or writing, a file must be opened using the built-in **open()** function.

file = open("filename.txt", "mode")

Common Modes:

Mode	Description	File Created if Missing
'r'	Read only (default). Raises error if file not found.	❌ No
'w'	Write mode. Overwrites existing file.	✅ Yes
'a'	Append mode. Adds data to end of file.	✅ Yes
'r+'	Read and Write (no overwrite). File must exist.	❌ No
'w+'	Read and Write (overwrites).	✅ Yes
'a+'	Read and Write (append).	✅ Yes

Example:

f = open("data.txt", "r")

📖 2. Reading from a File

Once a file is opened in **read mode**, Python provides multiple ways to read its contents.

a) read()

Reads **entire file content** as a single string.

f = open("example.txt", "r")

content = f.read()

print(content)

```
f.close()
```

✅ *Best for small files.*

---

b) readline()

Reads **one line** at a time (including the newline \n).

```
f = open("example.txt", "r")
line1 = f.readline()
line2 = f.readline()
print(line1)
print(line2)
f.close()
```

✅ *Useful for processing files line by line.*

---

c) readlines()

Reads **all lines at once** and returns a **list**.

```
f = open("example.txt", "r")
lines = f.readlines()
for line in lines:
    print(line.strip())  # removes '\n'
f.close()
```

✅ *Best when you want to iterate over lines.*

✍️ 3. Writing to a File

You can write data using either write() or writelines().

a) write()

Writes a single string to the file.

```
f = open("newfile.txt", "w")
f.write("Hello, this is a new file.\n")
```

```
f.write("Second line of text.")

f.close()
```

✅ *Overwrites existing data if file already exists.*

---

b) writelines()

Writes **multiple strings** (usually from a list).

```
f = open("notes.txt", "w")

lines = ["First line\n", "Second line\n", "Third line\n"]

f.writelines(lines)

f.close()
```

✅ *Does NOT automatically add newlines — you must include \n manually.*

🧹 4. Closing the File

Always close your file using:

```
f.close()
```

This releases system resources and ensures data is properly written.

💡 5. Using the with Statement (Professional Way)

In modern Python code, professionals prefer using the with statement.
It **automatically closes** the file — even if an error occurs.

```
with open("data.txt", "r") as f:

    content = f.read()

    print(content)

# file is automatically closed here
```

🧠 6. Example: Read from One File, Write to Another

```
with open("input.txt", "r") as infile:

    data = infile.read()
```

```
with open("output.txt", "w") as outfile:
    outfile.write("Copied content:\n")
    outfile.write(data)
```

📊 7. Comparison Summary

Function	Purpose	Returns	Typical Use
read()	Read all content	String	Small files
readline()	Read one line	String	Sequential line processing
readlines()	Read all lines	List	Looping over lines
write()	Write one string	None	Writing text
writelines()	Write list of strings	None	Writing multiple lines

---

✅ Professional Tips

- Always use with open() instead of manual open() and close().
- Always handle files in text mode unless dealing with binary data ('rb', 'wb').
- Use **encoding='utf-8'** to support all languages:
- with open("file.txt", "r", encoding="utf-8") as f:
-     ... #Code
- Use try–except for error handling in production code.