**Que 1 : Opening files in different modes ('r', 'w', 'a', 'r+', ‘w+').**

In Python, you use the open() function to work with files.

Basic Syntax: file = open("filename.txt", "mode")

**Common File Modes**

|  |  |  |
| --- | --- | --- |
| **Mode** | **Name** | **Description** |
| r' | Read | Opens file for **reading only** (default). File **must exist**. |
| w' | Write | Opens file for **writing**. **Creates** a new file if it doesn’t exist, or **overwrites** it if it does. |
| a' | Append | Opens file to **add new content at the end**. Creates file if it doesn’t exist. |
| r+' | Read + Write | Opens file for **reading and writing**. File **must exist**. |
| w+' | Write + Read | Opens file for **reading and writing**, **clears file first** or creates it. |

1. 'r' – Read Mode

Example :

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

content = f.read()

print(content)

f.close()

2. 'w' – Write Mode

Example :

f = open("file.txt", "w")

f.write("Hello, this is new content.")

f.close()

3. 'a' – Append Mode

Example :

f = open("file.txt", "a")

f.write("\nThis will be added at the end.")

f.close()

4. 'r+' – Read and Write

Example :

f = open("file.txt", "r+")

print(f.read())

f.write("\nAdded new line.")

f.close()

5. 'w+' – Write and Read (Deletes existing)

Example:

f = open("file.txt", "w+")

f.write("Fresh content!")

f.seek(0) # Move to beginning to read

print(f.read())

f.close()

**Que 2 : Using the open() function to create and access files.**

The open() function is used to **create**, **read**, **write**, and **append** files in Python.

**Syntax:**

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

* "filename.txt" → name of the file (with or without path)
* "mode" → how you want to use the file ('r', 'w', 'a', etc.)

**1. Creating a New File**

Use 'w', 'a', or 'x' mode to **create a file**.

✔ **Example using 'w':**

file = open("newfile.txt", "w") # Creates newfile.txt if it doesn’t exist

file.write("Hello! This is a new file.")

file.close()

* If the file exists → content is **overwritten**
* If the file doesn’t exist → it will be **created**

**2. Writing to a File**

Use 'w', 'a', 'w+', or 'r+'.

file = open("example.txt", "w")

file.write("Writing some content to the file.")

file.close()

**Append more text:**

file = open("example.txt", "a")

file.write("\nAdding more text.")

file.close()

**3. Reading from a File**

Use 'r', 'r+', or 'w+'.

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

data = file.read()

print(data)

file.close()

**4. Checking if File Exists Before Opening**

To avoid errors in read mode:

import os

if os.path.exists("example.txt"):

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

print(file.read())

file.close()

else:

print("File does not exist.")

**5. Using with Statement (Best Practice)**

Automatically closes the file after use.

with open("example.txt", "r") as file:

content = file.read()

print(content)

**6. Creating a File in a Specific Folder**

file = open("foldername/myfile.txt", "w")

file.write("This file is inside a folder.")

file.close()

**Que 3 : Closing files using close().**

When you open a file using open(), it's important to **close** it after you're done. This frees up system resources and ensures all data is written properly (especially when writing to a file).

**Syntax:**

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

# perform file operations

file.close()

**Why is close() Important?**

|  |  |
| --- | --- |
| **Reason** | **Explanation** |
| Saves Data | If you're writing to a file, closing ensures all data is **flushed to disk**. |
| Frees Resources | Closing the file releases memory and system resources. |
| Avoids File Corruption | If a file isn't properly closed, data may be **incomplete or damaged**. |
| Allows Safe Re-access | Keeps files from being **locked** for too long. |

**Example:**

# Writing to a file

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

f.write("This is some text.")

f.close() # 👈 Closes the file properly

# Reading a file

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

print(f.read())

f.close()

**What if You Forget to Close?**

* Sometimes, especially in writing mode, the data **won’t actually be saved** until the file is closed.
* This could also lead to **file lock issues** or **memory leaks** in large programs.

**Best Practice: Use with Statement (Auto-Close)**

Instead of manually closing, Python can do it for you

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

data = f.read()

print(data)

# File is automatically closed here, even if an error happens inside the block