1. What is a File?

A file is like a **notebook** where you store your information — such as data, notes, numbers, etc.

Think of:

- A `.txt` file = plain text like a note
- A `.csv` file = like an Excel sheet, used a lot in data analysis
- A `.json` file = structured data like a dictionary

---

2. Why File Handling Is Important in Data Analysis?

As a data analyst, you often:

- Read data from a file (like sales data in `.csv`)
- Clean or analyze that data
- Save the results into another file

📂 So, learning how to **read** and **write** files is the first step before doing any kind of analysis.

---

3. Opening and Reading Files

Python helps you open a file using a **built-in function** called `open()`.

Syntax:

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

- `"filename.txt"` → the name of the file

- `"mode"` → what you want to do with the file
  - `"r"` for reading
  - `"w"` for writing
  - `"a"` for appending
  - `"rb"` for reading in binary mode (not needed now)

Example 1: Reading a Text File

Step 1: Create a file first

Create a text file named `data.txt` with the following content:

```
Hello
Welcome to Python file handling
You are learning data analysis
```

Step 2: Now read this file using Python

```
# Open the file in read mode
file = open("data.txt", "r")

# Read the entire content of the file
content = file.read()

# Print the content
print("File Content:")
print(content)

# Close the file
file.close()
```

Let’s Understand This Code Step-by-Step

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

- This line **opens the file** named `data.txt` in **read mode**.
- Python stores it in a variable called `file`.

```
content = file.read()
```

- This reads the **entire content** of the file and stores it in a variable called `content`.

```
print(content)
```

- This simply prints what you just read from the file.

```
file.close()
```

- Always **close the file** after you're done. It's like putting the lid back on a jar after using it.

Example 2: Reading Line by Line

Sometimes, you don't want to read the whole file at once — especially if it's very big. You can read it **line by line** like this:

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

print("Reading line by line:")
for line in file:
    print(line.strip())  # .strip() removes the newline character

file.close()
```

Explanation:

- `for line in file:` → Goes through each line in the file one by one
- `line.strip()` → Removes the extra space or newline at the end

Common Errors and Solutions

Error	What It Means	How to Fix
`FileNotFoundError`	The file does not exist	Check if the filename and path are correct
`PermissionError`	You don't have access	Make sure you have permission to open the file
Forgot `.close()`	The file may stay open	Always use `file.close()` or `with open()` (we'll learn later)

Practice Exercise

Task:

1. Create a text file called `student_notes.txt`
2. Write 3 lines of notes about your course
3. Write a Python program to read and display the content

1. Writing Data to a File (`"w"` Mode)

When you use **write mode**, Python:

- Opens the file (creates a new one if it doesn't exist)

- Deletes everything inside the file if it already exists
- Writes new data from scratch

Example 1: Writing to a File

```
# Open file in write mode
file = open("report.txt", "w")

# Write text into the file
file.write("Name: Ravi\n")
file.write("Marks: 82\n")
file.write("Status: Pass\n")

# Close the file
file.close()

print("Data written successfully.")
```

Code Explanation:

- `"w"` means write mode — you want to write new data.
- `.write()` is used to send text into the file.
- `\n` means "new line" (it starts from the next line, like pressing Enter).
- After writing, we close the file using `.close()`.

Warning:

```
open("report.txt", "w")
```

If the file already **has content**, it will be **erased** when opened in `"w"` mode.

---

2. Appending Data to a File (`"a"` Mode)

When you use **append mode**, Python:

- Opens the file
- **Keeps the old content**
- **Adds new data at the end**

Example 2: Appending Data to the File

```
# Open file in append mode
file = open("report.txt", "a")

# Add new lines
file.write("Name: Priya\n")
file.write("Marks: 90\n")
file.write("Status: Pass\n")

# Close the file
file.close()

print("New data appended successfully.")
```

Code Explanation:

- `"a"` = append mode
- New content is added at the **end** of the existing file

- Old content is **safe** and remains untouched

---

3. Using `with open()` (Safe Way)

A better and safer way to open files is by using `with open(...) as` — this method **automatically closes** the file for you.

Example 3: Using `with open()` to Write

```
with open("students.txt", "w") as file:
    file.write("Student 1: Ramesh\n")
    file.write("Student 2: Suresh\n")

print("Written using with-open block.")
```

You don't need to write `file.close()` — Python handles it for you.

When to Use What?

Mode	Use When...	Caution
`"w"`	You want to **create** a new file or **overwrite** data	Deletes old content
`"a"`	You want to **add** new data without deleting old data	Can become long quickly
`"r"`	You want to **read** a file	File must already exist

Real-Life Use Case: Saving Feedback

```
name = input("Enter your name: ")
feedback = input("Enter your feedback: ")
```

```python
with open("feedback.txt", "a") as file:
    file.write(f"Name: {name}\n")
    file.write(f"Feedback: {feedback}\n")
    file.write("-----\n")

print("Thank you for your feedback!")
```

This example shows how you might collect feedback from users and save it to a file.

Quick Tips

- Always use `\n` when writing multiple lines.
- Use `"a"` mode to add logs, notes, or results without losing old data.
- Prefer `with open(...) as` for cleaner and safer code.

Practice Exercise

Task:

1. Create a file called `students.txt`.
2. Ask user to enter 3 students' names and marks.
3. Write each one on a new line using **append mode**.

Bonus: Use `with open()` to make your code cleaner!

1. What is a CSV File?

CSV stands for Comma-Separated Values

It's a simple file that stores **tabular data**, like an Excel sheet.
Each **line** in the file is a **row**, and values are separated by **commas**.

Example: `students.csv`

```
Name,Marks,Grade
Ravi,82,A
Priya,90,A+
Amit,75,B
```

This file has **columns** (Name, Marks, Grade) and **rows** (students' data).
It’s commonly used in data analysis because it's **easy to create, read, and share**.

---

2. Why CSV is Important in Data Analysis?

CSV is used to store:

- Survey data
- Sales data
- Student marks
- Stock market data
- Government open data

You’ll work with CSVs in nearly every data analysis project.

---

3. How to Read CSV Files in Python

Python has a built-in module called `csv` which makes reading easy.

Step-by-Step Code to Read a CSV File

Let's say we have a file called `students.csv` with this content:

```
Name,Marks,Grade
Ravi,82,A
Priya,90,A+
Amit,75,B
```

Code:

```
import csv  # Step 1: Import the csv module

# Step 2: Open the file
with open("students.csv", "r") as file:
    reader = csv.reader(file)  # Step 3: Create a CSV reader

    # Step 4: Loop through each row in the file
    for row in reader:
        print(row)
```

Code Explanation:

Line	What It Does
`import csv`	Loads the csv tools
`open("students.csv", "r")`	Opens the CSV file in read mode
`csv.reader(file)`	Turns the file into a reader object
`for row in reader`	Loops through each row

`print(row)`	Displays each row as a list

✅ **Output:**

```
['Name', 'Marks', 'Grade']
['Ravi', '82', 'A']
['Priya', '90', 'A+']
['Amit', '75', 'B']
```

🧠 Each row is shown as a **list of values**.

---

4. Skip the Header Row (Optional)

```
with open("students.csv", "r") as file:
    reader = csv.reader(file)
    next(reader)  # Skip the header row

    for row in reader:
        print("Name:", row[0])
        print("Marks:", row[1])
        print("Grade:", row[2])
        print("------")
```

Output:

```
Name: Ravi
Marks: 82
Grade: A
------
Name: Priya
Marks: 90
Grade: A+
------
Name: Amit
Marks: 75
Grade: B
------
```

`next(reader)` is used to skip the first line — which usually contains column names.

---

5. Real-Life Example: Reading Survey Results

Imagine you collected survey data in a file called `feedback.csv`:

```
Name,Rating,Comment
Ali,5,Very good
Meena,4,Good
John,3,Average
```

Now let's read and display each person's feedback:

```
import csv

with open("feedback.csv", "r") as file:
    reader = csv.reader(file)
    next(reader)  # Skip header

    for row in reader:
        print(f"{row[0]} rated us {row[1]}/5 and said: {row[2]}")
```

✅ Output:

```
Ali rated us 5/5 and said: Very good
Meena rated us 4/5 and said: Good
John rated us 3/5 and said: Average
```

Practice Exercise

Task:

Create a CSV file named `employees.csv` with this data:

```
Name,Department,Salary
Ramesh,Sales,30000
Suresh,IT,50000
Leena,HR,40000
```

1.
2. Write Python code to read this file and display data in a readable format like:

   ```
   Employee: Ramesh | Department: Sales | Salary: 30000
   ```

1. Why Write to CSV?

In real-world data analysis, you:

- Process data (clean, filter, calculate)
- Need to **save the results** into a file
- Share that file with others (team, boss, client)

CSV is the **universal format** — easy to read, edit (even in Excel), and share.

---

2. How to Write to a CSV File in Python

To write to a CSV, we use:

`csv.writer()`

Example 1: Write Student Marks to a CSV File

```
import csv

# Step 1: Open the file in write mode
with open("output.csv", "w", newline="") as file:
    writer = csv.writer(file)  # Step 2: Create a CSV writer

    # Step 3: Write header row (column names)
    writer.writerow(["Name", "Marks", "Grade"])

    # Step 4: Write data rows
    writer.writerow(["Ravi", 82, "A"])
    writer.writerow(["Priya", 90, "A+"])
    writer.writerow(["Amit", 75, "B"])

print("Data written to CSV successfully.")
```

Code Explanation:

Line	Meaning
`import csv`	Brings in CSV tools
`open("output.csv", "w")`	Creates or overwrites the file
`newline=""`	Ensures no extra blank lines
`csv.writer(file)`	Creates a writer object
`writer.writerow(...)`	Writes one row of data

Output CSV File (`output.csv`):

```
Name,Marks,Grade
Ravi,82,A
Priya,90,A+
Amit,75,B
```

---

3. Append New Data to an Existing CSV File

Let’s say you already have a file and want to **add more data at the end** without deleting the old content.

Use `"a"` mode (append):

Example 2: Append Data to CSV

```
import csv

with open("output.csv", "a", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(["Suresh", 88, "A"])

print("New data appended.")
```

---

4. Real-Life Use Case: Save Feedback into CSV

Let’s collect and save user feedback dynamically:

```
import csv

name = input("Enter your name: ")
rating = input("How would you rate us (1–5)? ")
comment = input("Write a comment: ")
```

```
with open("feedback.csv", "a", newline="") as file:
    writer = csv.writer(file)

    # Write header only if file is empty
    import os
    if os.stat("feedback.csv").st_size == 0:
        writer.writerow(["Name", "Rating", "Comment"])

    writer.writerow([name, rating, comment])

print("Thank you! Your feedback is saved.")
```

Common Mistakes to Avoid

Mistake	Fix
Extra blank lines in output	Use `newline=""` in `open()`
File gets erased	Use `"a"` for appending, not `"w"`
Forgetting headers	Always write column names first

Practice Exercise

Task:

1. Create a file `sales.csv`
2. Write this header: `Product,Quantity,Price`
3. Add 3 products manually using `writer.writerow()`

Bonus: Use `input()` to let user enter product info and append it to the file.

1. What is an Error in Programming?

An **error** is when Python doesn't understand something or something goes wrong during execution.

Example:

```
age = int(input("Enter your age: "))
```

If the user types `"hello"` instead of a number, it crashes:

```
ValueError: invalid literal for int() with base 10
```

---

2. What is `try-except`?

The `try` and `except` keywords are used to **catch and handle errors**.
This prevents your program from **crashing**, and lets you show a **friendly message** instead.

Syntax:

```
try:
    # Code that might cause an error
except:
    # What to do if there's an error
```

---

3. Basic Example: Safe Input

```
try:
    age = int(input("Enter your age: "))
    print("You are", age, "years old.")
except:
    print("Oops! Please enter a valid number.")
```

What Happened Here?

- If the input is a number (like `25`) → ✅ it runs normally.
- If the input is `"twenty"` → ❌ error caught, shows friendly message.

---

4. Real-Life File Error Example

Let’s say we try to open a file that doesn’t exist:

```
try:
    file = open("data.csv", "r")
    print(file.read())
    file.close()
except:
    print("File not found. Please check the file name.")
```

---

5. Catching Specific Errors

You can catch **specific types of errors**:

Example: Catching `FileNotFoundError`

```
try:
    file = open("students.csv", "r")
    print(file.read())
except FileNotFoundError:
    print("The file 'students.csv' was not found.")
```

Example: Catching `ValueError`

```
try:
    number = int(input("Enter a number: "))
except ValueError:
    print("That's not a number!")
```

Why Use Specific Errors?

- You know **exactly what went wrong**
- You can handle **different errors** in different ways

---

6. Multiple `except` Blocks

You can handle multiple error types:

```
try:
    file = open("students.csv", "r")
    data = int(input("Enter a number: "))
except FileNotFoundError:
    print("The file doesn't exist.")
except ValueError:
    print("Invalid number.")
```

Real-Life Example: Survey Input + Save to File

```python
import csv

try:
    name = input("Enter your name: ")
    rating = int(input("Rate us (1-5): "))
    with open("feedback.csv", "a", newline="") as file:
        writer = csv.writer(file)
        writer.writerow([name, rating])
    print("Thank you! Feedback saved.")
except ValueError:
    print("Please enter a valid number for rating.")
except:
    print("Something went wrong. Please try again.")
```

Practice Exercises

Task 1:

Ask the user for a number and catch any `ValueError` if the input is not a number.

Task 2:

Try to open a file named `data.csv`. If the file is missing, show a message without crashing.

Task 3:

Write a program that asks the user to:

- Enter a name
- Enter marks (number)
- Save it to a CSV
  Add error handling for invalid marks input or file write errors.