Working with Files

- Reading, Writing, and Handling Data Persistently

Why Use Files?

- Data in variables is **temporary** (lost when the program ends).
- Files allow data to be **stored permanently** on secondary storage.
- Files are essential for:
  - Saving user settings
  - Logging information
  - Storing large datasets

RAM = short-term memory

Files = long-term memory.

Types of Files

- **Text files (.txt):** store human-readable data (strings, numbers, etc.).
- **Binary files:** store data in computer-readable form (images, executables, etc.).
- CSV files (.csv): **Comma-Separated Values**
- Json files(.json): Stores data in **key-value pairs** (similar to Python dictionaries).

- **OCR focus** → mainly text file handling in Python.

Basic File Operations

- **Open** a file
- **Read** from a file
- **Write** to a file
- **Close** a file

Python uses the open() function with different modes:

- "r" → read
- "w" → write (overwrites)
- "a" → append

Reading from a File

```
file = open("data.txt", "r")
for line in file:
    print(line.strip())
file.close()
```

strip() removes the newline character \n.

Writing to a File

```
file = open("output.txt", "w")
file.write("Hello, world!\n")
file.write("This is stored permanently.")
file.close()
```

"w" overwrites the file.

Appending to a File

```
file = open("output.txt", "a")
file.write("\nAdding another line.")
file.close()
```

New data is added to the end of the file.

Using with (Best Practice)

Automatic closing of files.

```
with open("data.txt", "r") as file:
    for line in file:
        print(line)
```

Cleaner, safer, avoids forgetting close().

Common OCR Exam Points

- Understanding **modes** (“r”, “w”, “a”).
- Using iteration (for line in file).
- Knowing why files are needed (persistent storage).
- Writing pseudocode vs Python.

Mini activity

- Create a text file called scores.txt.
- Write five student scores into it (one per line).
- Write a Python program to:
  - Read the scores from the file.
  - Calculate the average.
  - Write the result to a new file average.txt.

CSV Files

- **CSV = Comma-Separated Values**
- Stores data in a **table-like structure** (rows & columns).
- Each line = a record (row).
- Each value separated by commas (or sometimes tabs).

students.csv

```
Name,Score
Alice,85
Bob,73
Charlie,91
```

Working with CSV in Python

Output is a list per row: ['Alice', '85'].

```
import csv

with open("students.csv", "r") as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

Writing to CSV:

```
import csv

with open("students.csv", "a",
newline="") as file:
    writer = csv.writer(file)
    writer.writerow(["David", 77])
```

Json files

- **JSON = JavaScript Object Notation**
- Stores data in **key-value pairs** (similar to Python dictionaries).
- Human-readable and widely used in web APIs.

student.json

```
{
  "name": "Alice",
  "score": 85,
  "passed": true
}
```

JSON is stored as **dictionaries** in Python.

What is a Dictionary?

- A **data structure** in Python.
- Stores data as **key–value pairs**.
- Unlike lists (which are ordered by index), dictionaries are accessed by **keys**.

💡 Think of a real dictionary: you look up a *word* (key) to find its *definition* (value).

Creating a dictionary

```
# Empty dictionary
student = {}

# Dictionary with data
student = {
    "name": "Alice",
    "age": 17,
    "score": 92
}
```

Here:
"name", "age", "score" = keys
"Alice", 17, 92 = values

Accessing Values

Use the **key** in square brackets, not an index.

```
print(student["name"])    # Alice
print(student["score"])   # 92
```

Updating and Adding Data

Now student has a new key "grade".

```
student["age"] = 18          # update value
student["grade"] = "A"        # add new key-value pair
```

Iterating Through a Dictionary

```
for key, value in student.items():
    print(key, "→", value)
```

Working with JSON in Python

```
import json

# Reading
with open("student.json", "r") as file:
    data = json.load(file)
print(data["name"], data["score"])

# Writing
student = {"name": "Bob", "score": 73, "passed": False}
with open("student.json", "w") as file:
    json.dump(student, file, indent=4)
```

Summary

- Files allow data to be **stored beyond runtime**.
- Two main operations: **read** and **write**.
- Python file modes: "r", "w", "a".
- Best practice: use with open(...) as ....