## Introduction to File Handling

When writing programs in Python, often you need to store or retrieve data from external files. This is crucial for tasks like:

- Persisting data for future use
- Sharing data between programs
- Working with large datasets
- Logging application activity

File handling is essential in programming, especially in areas like AI/ML where we process large datasets.

## What is File Handling?

File handling refers to the process of interacting with external files using code. This includes:

- Opening files
- Reading content
- Writing data
- Appending new data
- Closing files after use

Python variables only store temporary data which is lost after execution. To persist data, files or databases are used.

## Why Use Files?

- Persist data beyond program execution
- Allow inter-program communication
- Store logs of application operations
- Process large external datasets (CSV, JSON, etc.)

## Syntax for Opening Files

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

Modes:

- `r` – Read (default mode)
- `w` – Write (creates new file or overwrites)

- a – Append (adds content at the end)
- r+ – Read and write

Always close the file after use:

```
file.close()
```

## File Modes and Their Behavior

1. Read Mode (r)
    - Opens existing file
    - Fails if file doesn't exist
    - Use: file.read()
2. Write Mode (w)
    - Overwrites existing content
    - Creates file if not present
    - Use carefully to avoid data loss
3. Append Mode (a)
    - Adds new data at end of file
    - Preserves previous content
4. Read & Write Mode (r+)
    - Reads and writes to same file
    - Allows both operations

## Hands-on Code Examples

**Creating and Writing to File:**

```
with open('example.txt', 'w') as file:
file.write('This is a new file created using write mode.')
print('File written successfully.')
```

**Reading File Content:**

```
try:
    with open('example.txt', 'r') as file:
        content = file.read()
print('File Content:', content)
except FileNotFoundError:
print('Error: The file does not exist!')
```

**Appending New Line:**

```
with open('example.txt', 'a') as file:
file.write('\nThis is a new line added using append mode.')
print('New content appended successfully.')
```

**Read and Write Mode:**

```
with open('example.txt', 'r+') as file:
content = file.read()
print('Current Content:', content)
file.write('\nAdding new content in read+write mode.')
print('Content updated successfully.')
```

## JSON in Python

JSON (JavaScript Object Notation) is a universal format for storing and exchanging data.
Why JSON?

- Cross-language support
- Lightweight and structured
- Ideal for APIs and web applications

  Working with JSON in Python

```
import json
# Sample dictionary
data = {'name': 'Haris', 'age': 25, 'city': 'Lahore'}
# Writing JSON data to a file
with open('data.json', 'w') as file:
json.dump(data, file)
# Reading JSON data from a file
with open('data.json', 'r') as file:
loaded_data = json.load(file)
print(loaded_data)
```

## Exception Handling in File Operations

If a file doesn’t exist and you try to read it, an error occurs. Handle it using try-except:

```
Try:
with open('non_existent_file.txt', 'r') as file:
        content = file.read()
        print(content)
except FileNotFoundError:
```

```
print('Error: The file does not exist!')
```

Exception handling makes your program robust and user-friendly.

## Additional Notes on File Handling Behavior

- Write mode (`w`) creates file if it doesn't exist, otherwise overwrites it.
- Append mode (`a`) adds data to existing content.
- Read mode (`r`) fails if file is missing.
- Always close files after use to prevent resource locking.

## Best Practices for File Handling

- Always close files using `file.close()` or use `with open()` for auto-close
- Use exception handling for safe operation
- Be cautious with write mode (`w`) as it overwrites data
- Prefer JSON for structured data storage

## Summary of Key Concepts

- Use `open()` with appropriate mode (`r`, `w`, `a`, `r+`)
- Handle files using `.read()`, `.write()`, `.close()`
- JSON files are structured and cross-platform friendly
- Handle errors using `try-except`

## Final Thoughts

File handling is a fundamental concept in Python that enables data storage, transfer, and logging. JSON further enhances Python's capabilities by enabling structured and scalable data exchange. Continue experimenting with `.txt`, `.json`, and `.csv` files to build real-world solutions.