Objectives Of the Topic 🌟

- **Master Files:** Get hands-on with opening, reading, writing, and closing files in Python!
- **Handle Errors Like a Pro:** Learn how to catch and manage errors, ensuring your code runs smoothly even when things don't go as planned.

What is File Handling? 📁

File Handling in Python is the ability to perform various operations on files, like reading from and writing to them. Files are used to store data permanently (like text documents, images, or spreadsheets). Unlike variables, which lose their values when a program ends, files provide **persistent storage**.

File handling in Python allows you to:

1. **Open files** in different modes (e.g., read-only or write mode).
2. **Read and write data** in a variety of formats.
3. **Close files** to free up system resources.

Python's built-in `open()` function is at the heart of this process, and **learning how to handle files** can make your programs more robust, flexible, and useful.

File Operations in Python 🚪🔑 (1.5 hours)

Let's start with the basics of file handling!

- **Opening Files** 🔒
  - Use Python's `open()` function to access a file.
  - Syntax: `open(filename, mode)`, where:
    - `filename`: The name of the file you want to work with.
    - `mode`: The mode you want to open the file in.
  - Modes include:
    - `'r'`: Read mode, used for reading files.
    - `'w'`: Write mode, creates a new file or overwrites an existing one.
    - `'a'`: Append mode, adds new content without deleting existing data.
    - `'rb'`, `'wb'`: Binary modes for non-text files, like images.

  Example:

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

- **Reading Files** 📜
  - Python provides multiple ways to read file contents:
    - `.read()`: Reads the entire file.

        - `.readline()`: Reads a single line at a time.
        - `.readlines()`: Reads all lines and returns a list.
    - **Example:**

```
with open("example.txt", "r") as file: data = file.read()
print(data)
```

    - Use cases: **Processing large datasets** or analyzing **text documents.**
- **Writing & Appending to Files** ✍️
    - Writing is essential for **saving data**, like storing a user's progress or keeping a record.
    - `write()`: Overwrites content, while `append()` allowing adding without deleting.
    - **Example:**

```
with open("output.txt", "w") as file: file.write("Hello,
Python!")
```

- **Closing Files** 🚪
    - Files should be closed after processing to release system resources.
    - Python's `with` statement automatically handles closing, ensuring efficient resource management.