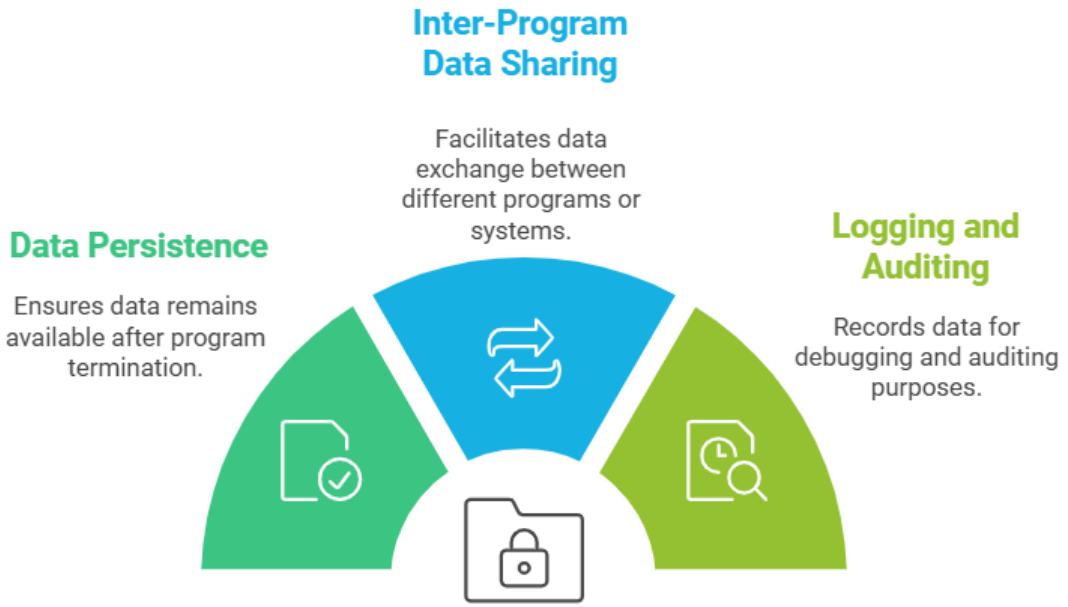
File Handling in Python

Dr. Muhammad Sajjad
RA. Wajahat Ullah

Introduction to File Handling

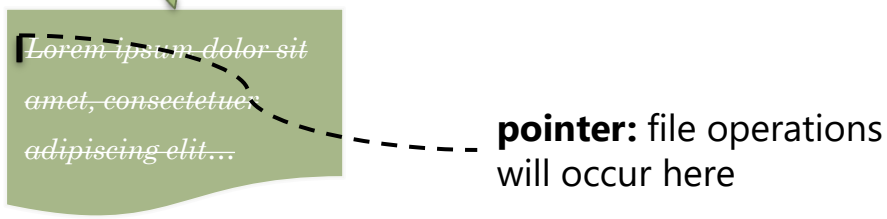
- File handling is the process of storing to and retrieving data from a persistent memory.
- It is essential part of any programming language.
- Data stored in variables is lost when the program terminates. Files allow you to store data permanently.
- Files can be used to share data between different programs or systems.
- Files are often used to log data for debugging or auditing purposes.

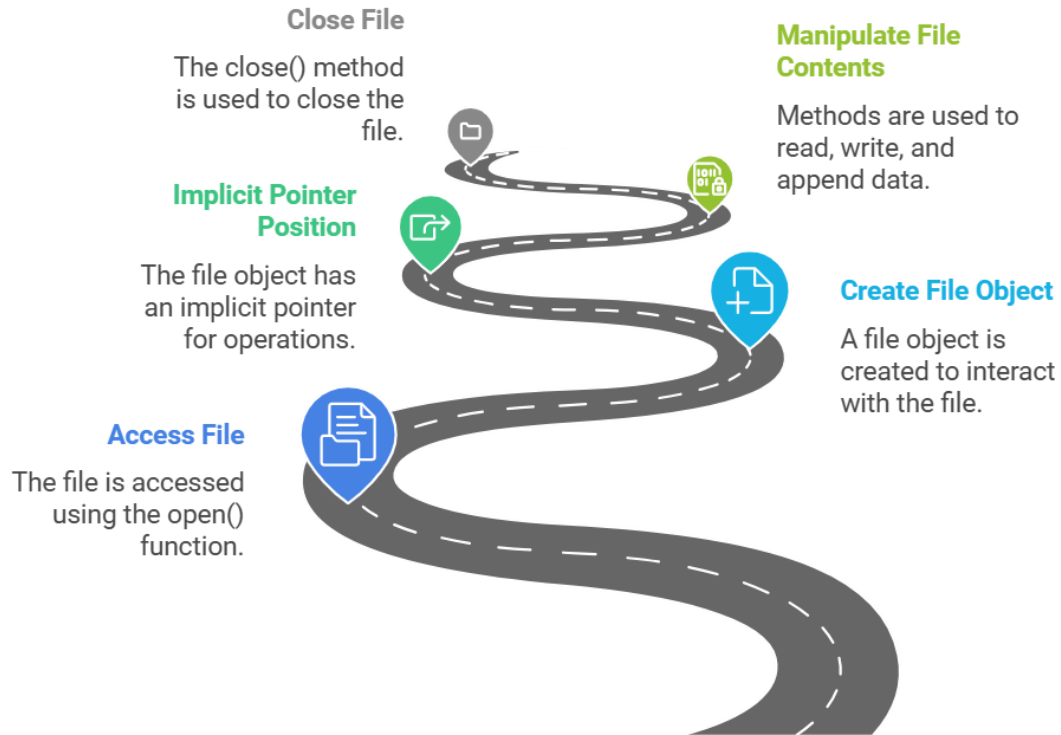


File Handling in Python

- Python provides an **object-oriented** approach to file handling.
- A **file** is accessed using the `open()` function which returns a file **object**.
- The file object has an implicit pointer which indicates the current position where the next read or write operation will occur.
- This file object has several methods to manipulate its contents like reading, writing, and appending.
- After we finish working with a file, the `close()` method is used to close the file and free any resources.

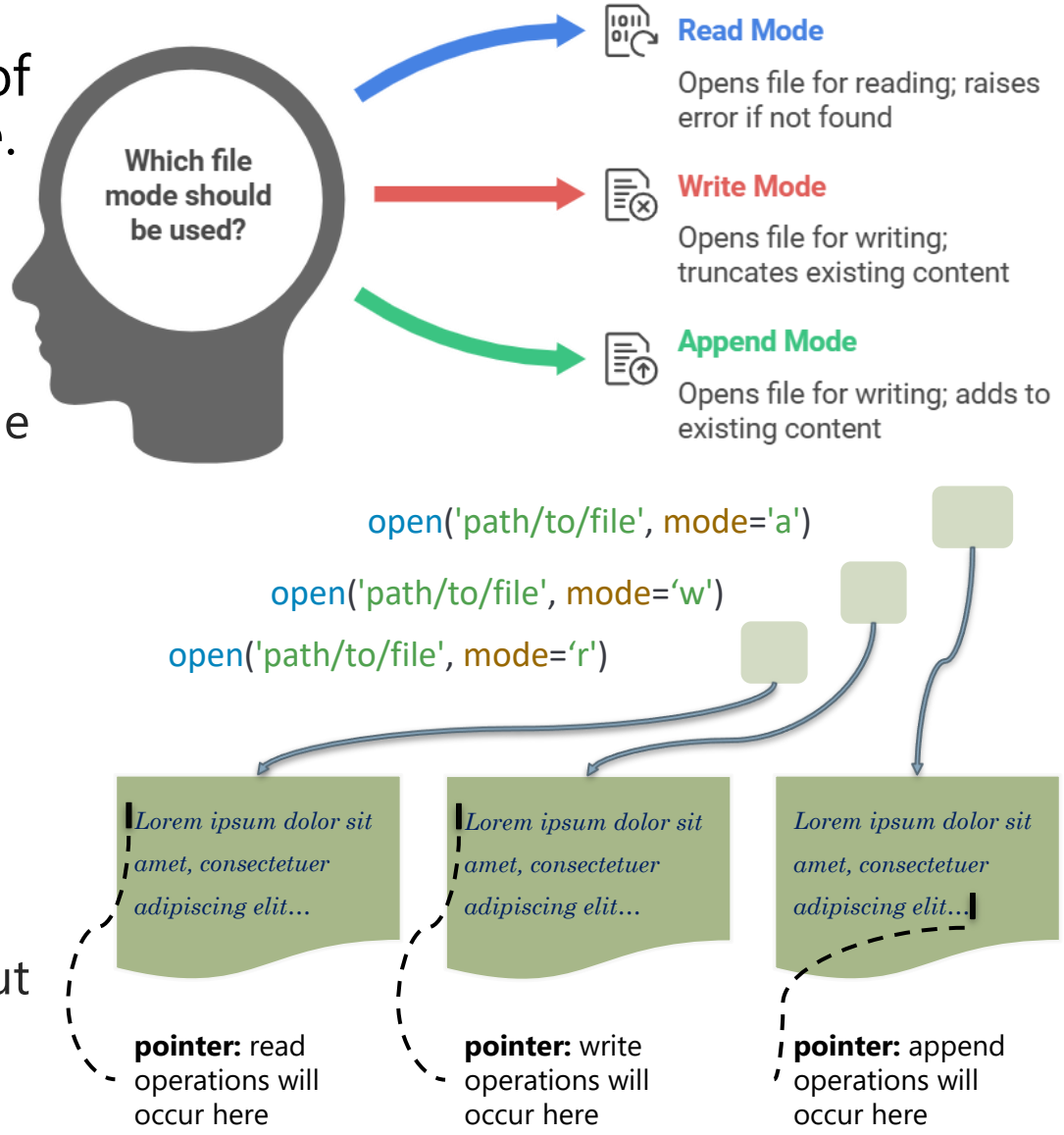
```
# Opening and closing a file
file = open('path/to/file', 'mode')
file.write('Hello World')
file.close()
```





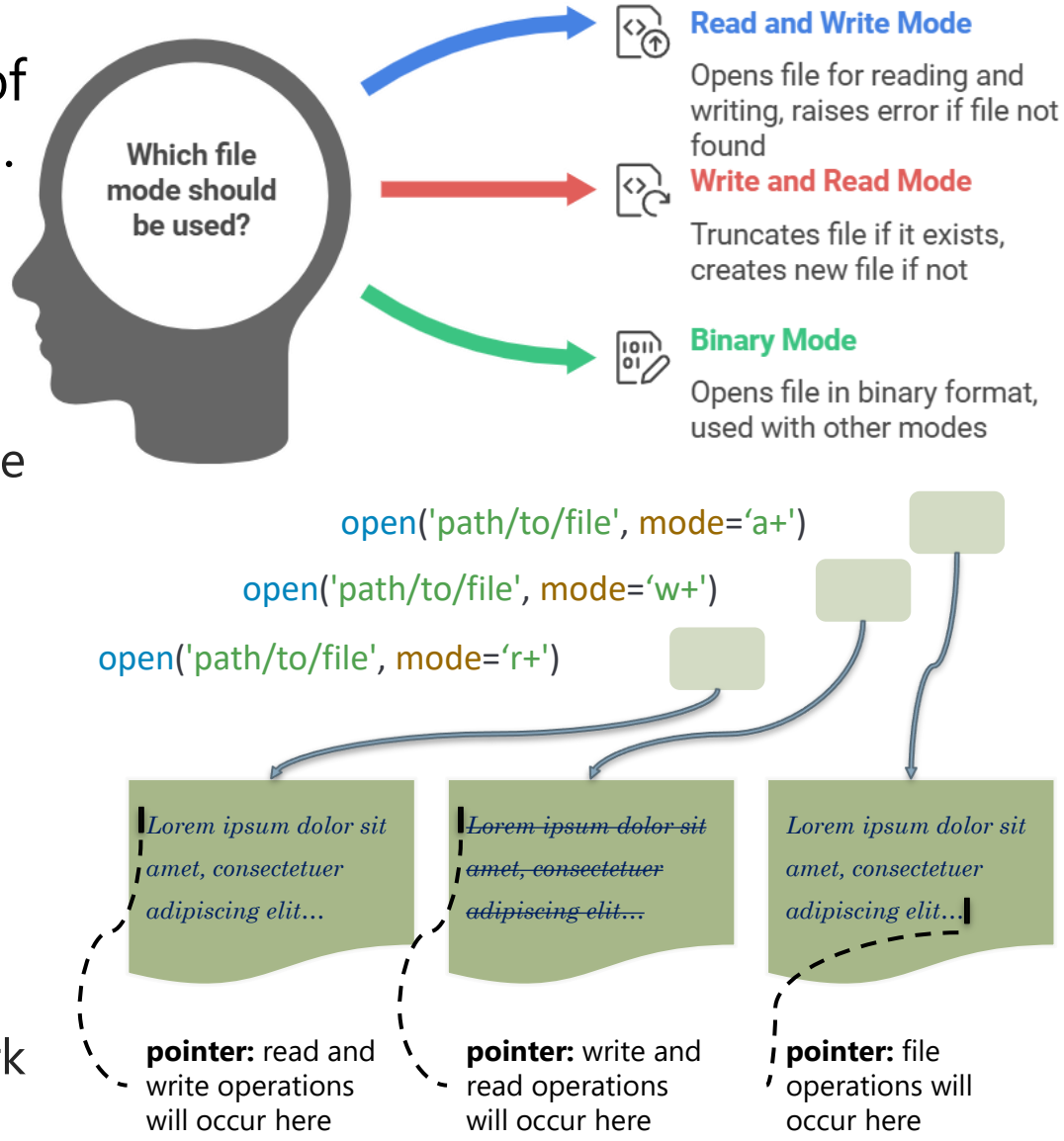
Different File Modes

- A file mode determines the kind of operations that can be performed on the file.
- **Read mode – `'r'`:**
  - Opens the file for reading only
  - File pointer is placed at the beginning of the file
  - `FileNotFoundError` is raised, when the file does not exist
- **Write mode – `'w'`:**
  - Opens the file for writing only
  - If the file exists, its content is truncated (deleted)
  - A new file is created, when the file does not exist
- **Append mode – `'a'`:**
  - Opens the file for writing
  - File pointer is placed at the end of the file without modifying existing content
  - A new file is created, when the file does not exist



Different File Modes

- A file mode determines the kind of operations that can be performed on the file.
- **Read and write mode – ‘r+’:**
  - Opens the file for both reading and writing
  - File pointer is placed at the beginning of the file
  - `FileNotFoundError` is raised, when the file does not exist
- **Write and read mode – ‘w+’:**
  - Opens the file for both writing and reading
  - If the file exists, its content is truncated (deleted)
  - A new file is created, when the file does not exist
- **Binary mode – ‘b’:**
  - Opens the file in binary model.
  - Used in combination with other modes to work with the content of the file in binary format.



Reading Files

- Reading from files is fundamental operation in file handling.
- To read the contents of a file, it must be opened in **'r'**, **'r+'**, or **'rb'** mode.
- The **read()** method reads the entire content of the file as a single string.
- The **readline()** method reads a single line from the file.
- The **readlines()** methods reads all lines in the file and returns them as a list of strings.
- ***Note:*** *In read mode,* ***file pointer*** *is placed at the beginning of the file initially.*


```
file = open('path/to/file', mode='r')
entire_file = file.read()
single_line = file.readline()
list_of_lines = file.readlines()
file.close()
```

Writing to Files

- Writing to files is necessary to store data in the files.
- To write some data to a file, it must be opened in **'w'**, **'w+**, **'a'**, **'a+'**, or **'wb'** mode.
- The **`write()`** method writes a string to a file but doesn't automatically add a newline at the end.
- The **`writelines()`** method writes a list of strings to the file, each on a new line.
- ***Note:*** *In write mode, initially the* ***file pointer*** *is placed at the beginning of the file while in append mode it is placed at the end.*


```
file = open('path/to/file', mode='w')
# write a line to the file
file.write('Hello, World!')
# write lines to the file
file.writelines(['I', 'Love', 'Python'])
file.close()
```

File Context Manager

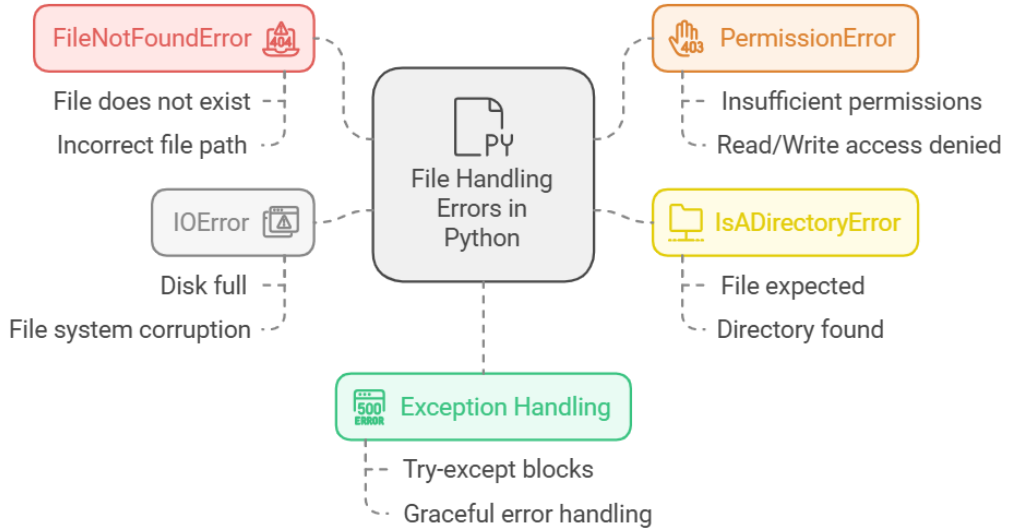
- File context manager is a way to handle files using the **`with`** statement.
- It ensures that the file is closed automatically after the block of code is executed.
- Even if an exception occurs within the block, the file will still be closed properly.
- It makes the code more readable and concise.
- File cannot be accessed outside of this context.

```
with open('path/to/file', mode='a') as file:
    # perform file operations
    ...
    # no need to close the file

# cannot perform operations with file here
```

Error Handling in File Operations

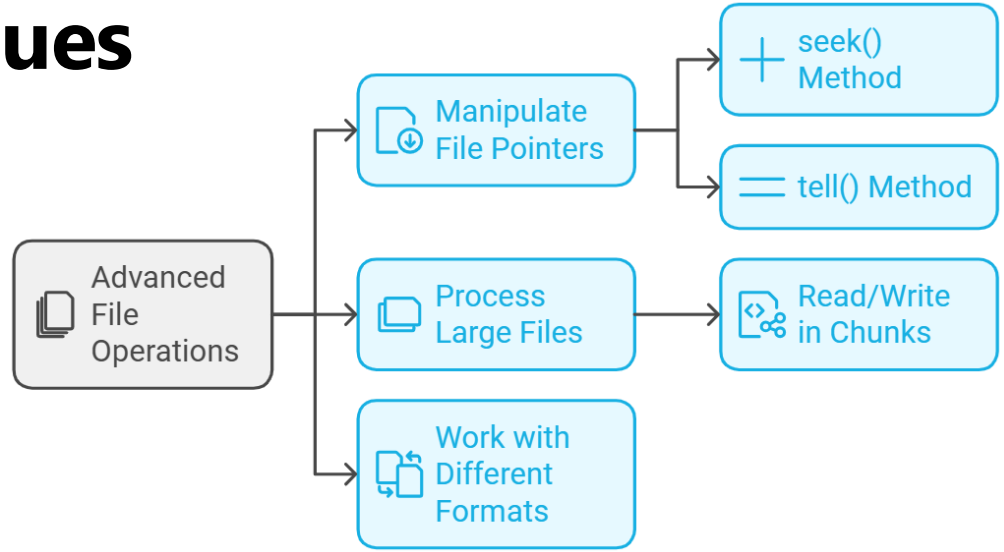
- Error handling is used when working with files to ensure that your program can gracefully handle unexpected situations.
- **`FileNotFoundError:`** Raised when trying to open a file that doesn't exist.
- **`PermissionError:`** Raised when the program does not have the necessary permissions to access the file.
- **`IsADirectoryError:`** Raised when a file is expected but a directory is found, or vice versa.
- **`IOError:`** Raised for various I/O related errors, such as disk full, file system corruption, etc.
- Python's exception handling mechanism is used to handle these situations using **`try-except`** blocks




```python
try:
    file = open("random.txt", "r")
    content = file.read()
    file.close()
except FileNotFoundError:
    print("Error: 'example.txt' was not found.")
except PermissionError:
    print("Error: no permission to read 'example.txt'.")
except Exception as e:
    print(f"Error: Unexpected error occurred. {e}")
finally:
    file.close()
    print("File closed.")
```

Advanced File Handling Techniques

- Python provides advanced operations like manipulating file pointers, reading large files, and working with different formats.
- We can manipulate file pointer to read and write at specific positions:
  - The seek() method allows us to control the file pointer position.
  - The tell() method returns the current position of the file pointer.
- Reading or writing the entire file at once can be inefficient and may cause memory leaks.
- Therefore, large files are processed in chunks.

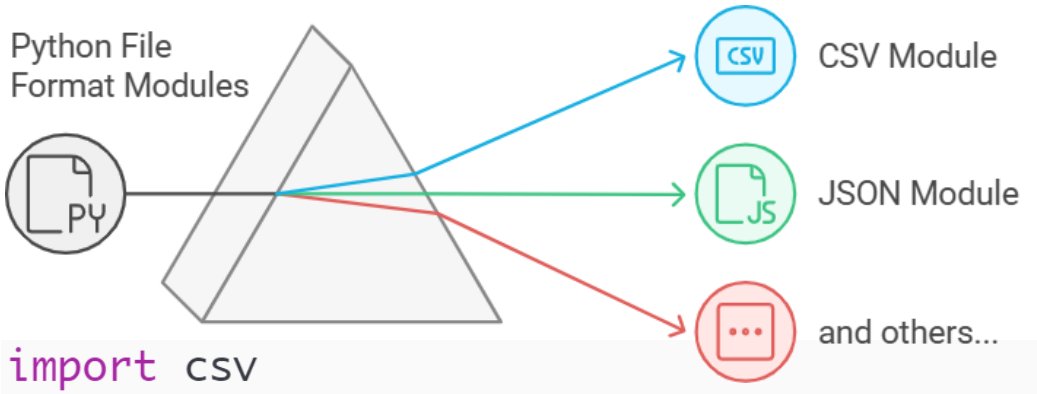



```python
with open('temp.txt', 'w+') as f:
    # gets the current file pointer position in the file
    f.tell()
    # sets the file pointer to new position in the file
    f.seek(0)
```

```python
with open(file_path, 'r') as file:
    while True:
        chunk = file.read(chunk_size)
        if not chunk:
            break
```

Advanced File Handling Techniques

- Python standard library has several modules to work with different file formats.
- The **`csv`** module allows working with CSV (Comma Separated Values) files which is a simple format for storing tabular data.
- Each line in the file represents a row and columns are typically separated by commas.
- The **`json`** module is used to work with JSON (JavaScript Object Notation) files, which is a lightweight data-interchange format.
- It is used to store structured data in a readable way.
- Its syntax is similar to python's dictionary.




```python
import csv

# Reading rows of a CSV file as
# lists using csv.reader
with open('example.csv', 'r') as csvfile:
    csvreader = csv.reader(csvfile)
    for row in csvreader:
        print(row)
```

```python
import json

# Reading from a JSON file
with open('example.json', 'r') as json_file:
    # json.load() reads from a file
    data = json.load(json_file)
    print("Content of example.json:\n", data)
```

File and Directory Management

- Python standard library also offers tools for managing files and directories through modules like **`os`** and **`shutil`**.
- These tools allow programmers to create, rename, delete, and move files or directories.
- It is essential for organizing resources in applications, automating file operations, and maintaining a clean project structure.
- Such operations are foundational for developing scripts and managing storage effectively.