# File Handling in Python

# File Handling

- **File handling** refers to the process of managing files in a computer system using programming techniques. It involves performing operations like creating, opening, reading, writing, appending, and deleting files.
- Python provides built-in functions and methods to create, read, write, and delete files.

# Importance of File handling

- **Data Storage**: It allows programs to store and retrieve data persistently, beyond the runtime of the program.
- **Data Sharing**: Files enable easy sharing of data between different programs or systems.
- **Data Manipulation**: It facilitates processing and manipulation of large datasets, such as logs, configurations, or user-generated content.
- **Backup**: Files are often used to backup important information, ensuring data is not lost during system failures.
- **Resource Management**: Proper file handling ensures efficient management of system resources, avoiding issues like data corruption or loss.

# Open()

- Before performing any operation on a file, you need to open it using Python's built-in **open( )** function. The **open( )** function returns a file object, which is used to read, write, or manipulate the file.

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

# File Modes

- 1. **Read Mode (`'r'`)**
- **Purpose**: Opens the file for reading only.
- **Behavior**:
  - The file pointer is placed at the beginning of the file.
  - If the file does not exist, a `FileNotFoundError` is raised.
- **Use Case**: When you want to read the contents of an existing file without modifying it
- file = open('example.txt', 'r')

- **Write Mode (`'w'`)**
- **Purpose**: Opens the file for writing.
- **Behavior**:
  - The file is truncated (i.e., all existing content is deleted) if it already exists.
  - If the file does not exist, it creates a new file.
- **Use Case**: When you want to write data to a file, starting with a clean slate.

- file = open('example.txt', 'w')

- **Append Mode (`'a'`)**
- **Purpose**: Opens the file for appending data.
- **Behavior**:
  - The file pointer is placed at the end of the file.
  - If the file does not exist, it creates a new file.
  - The existing content is preserved, and new data is added to the end.
- **Use Case**: When you want to add data to an existing file without deleting its current content.
- file = open('example.txt', 'a')

- **Exclusive Creation Mode (`'x'`)**
- **Purpose**: Opens the file for exclusive creation.
- **Behavior**:
  - If the file exists, the operation fails and raises a `FileExistsError`.
  - If the file does not exist, it creates a new file.
- **Use Case**: When you want to ensure that a new file is created and avoid overwriting an existing file.
- file = open('example.txt', 'x')

- **Binary Mode (`'b'`)**
- **Purpose**: Opens the file in binary mode.
- **Behavior**:
  - The file is read or written in binary format (i.e., bytes) rather than text.
  - Often used with other modes like `'rb'`, `'wb'`, `'ab'`, `'xb'`.
- **Use Case**: When working with non-text files, such as images, audio, or video files.
- file = open('image.png', 'rb')  # Read in binary mode

- **Text Mode (`'t'`)**
- **Purpose**: Opens the file in text mode (default).
- **Behavior**:
  - The file is read or written as text (i.e., strings).
  - Often used with other modes like `'rt'`, `'wt'`, `'at'`, `'xt'`.
- **Use Case**: When working with text files like `.txt`, `.csv`, or `.log` files
- file = open('example.txt', 'rt')  # Read in text mode

- **Read and Write Mode (`'r+'`)**
- **Purpose**: Opens the file for both reading and writing.
- **Behavior**:
  - The file pointer is placed at the beginning of the file.
  - If the file does not exist, a **`FileNotFoundError`** is raised.
  - Allows both reading and writing to the file, but does not truncate the file.
- **Use Case**: When you want to read and update the content of an existing file.
- file = open('example.txt', 'r+')

- **Write and Read Mode (`'w+'`)**
- **Purpose**: Opens the file for both writing and reading.
- **Behavior**:
  - The file is truncated (i.e., all existing content is deleted) if it already exists.
  - If the file does not exist, it creates a new file.
- **Use Case**: When you want to write new data to a file and then read it.
- file = open('example.txt', 'w+')

- **Append and Read Mode ('a+')**
- **Purpose**: Opens the file for appending and reading.
- **Behavior**:
  - The file pointer is placed at the end of the file.
  - If the file does not exist, it creates a new file.
  - Allows you to add data to the file and read its content without truncating it.
- **Use Case**: When you want to add to a file and also read its existing content.
- file = open('example.txt', 'a+')

- **Binary Read and Write Mode (`'r+b'` or `'rb+'`)**
- **Purpose**: Opens the file for both reading and writing in binary mode.
- **Behavior**:
  - The file pointer is placed at the beginning of the file.
  - If the file does not exist, a `FileNotFoundError` is raised.
- **Use Case**: When you need to read and update binary files, like images or executables.
- file = open('image.png', 'r+b')

- **Binary Write and Read Mode (`'w+b'` or `'wb+'`)**
- **Purpose**: Opens the file for both writing and reading in binary mode.
- **Behavior**:
  - The file is truncated if it already exists.
  - If the file does not exist, it creates a new file.
- **Use Case**: When you want to write to and then read from a binary file.
- file = open('image.png', 'w+b')

- **Binary Append and Read Mode (`'a+b'` or `'ab+'`)**
- **Purpose**: Opens the file for appending and reading in binary mode.
- **Behavior**:
  - The file pointer is placed at the end of the file.
  - If the file does not exist, it creates a new file.
- **Use Case**: When you need to append to and read from a binary file.
- file = open('image.png', 'a+b')

# Reading from a File

- The `read()` method reads the entire content of the file into a single string.
- Reads the entire file at once.
- Suitable for smaller files where the content can be stored in memory.
- Returns a string containing the file's content.
- If the file is large, this method may consume a lot of memory.

Example:

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

## Reading Line by Line with `readline()`

- The `readline()` method reads a single line from the file at a time.
- **Behavior:**
- Reads one line at a time.
- Suitable for processing files line by line, especially useful for large files.
- Returns a string representing the current line, including the newline character (`\n`).

## Reading All Lines into a List with `readlines()`


- **Reading All Lines into a List with `readlines()`**

- The `readlines()` method reads all the lines of the file and returns them as a list of strings.
- **Behavior:**
- Reads the entire file into a list, where each line is an element.
- Useful when you need to manipulate or access lines individually after reading.
- Returns a list where each element is a line from the file, including newline characters.

## Reading a Specified Number of Characters with `read(size)`

- The `read(size)` method reads a specified number of characters from the file.

**Behavior:**

- Reads up to `size` characters from the file.
- Useful when you need to process a specific amount of data at a time.
- Returns a string containing the characters read.

# Writing to a File

- Writing to a file can be done using the `write()` or `writelines()` methods.

- **Writing a String**: Use `write()` to write a string to a file.

- **Writing Multiple Lines**: Use `writelines()` to write multiple lines to a file.

# Checking File Existence

- Sometimes, you might want to check if a file exists before performing any operation. You can use the **os** module.
- Os.path.exists('exmple.txt')

- **Deleting a File**
- You can delete a file using the `remove()` method from the `os` module.
- If Os.path.exists('exmple.txt'):
  - Os.remove('exmple.txt')
- Else:
  - Print("file does not exist")

# File Position and Seeking

- You can manage the position of the file pointer using `tell()` and `seek()` methods.
- **`tell()`**: Returns the current position of the file pointer.
- **`seek(offset, whence)`**: Moves the file pointer to a specific position.