File Handling

1. Introduction

File handling in Python allows reading, writing, and modifying files efficiently.

2. Opening a File

- Default mode is 'r' (read mode).
- A file must be opened before performing any operations on it.

3. File Opening Modes

- **Read ('r')** – Opens a file for reading (default mode).
- **Write ('w')** – Opens a file for writing, overwriting existing content.
- **Append ('a')** – Opens a file for writing, appending content without removing existing data.
- **Text ('t') or Binary ('b')** – Specifies file type (text or binary).

4. Writing to a File

- Using 'w' mode replaces existing content.
- 'a' mode adds new content without deleting previous data.

5. Reading from a File

- .read() retrieves the entire content.
- .readline() reads one line at a time.
- .readlines() returns all lines as a list.

6. Closing a File

- Always close files using .close() to free system resources.
- Using with automatically closes the file after execution.

7. Seeking and Cursor Positioning

- .seek(n) moves the cursor to the nth byte in the file.
- .tell() returns the current cursor position.

8. Truncating a File

- .truncate(n) reduces the file size to n bytes.

9. Binary File Handling

- 'rb' mode reads binary files like images and videos.
- 'wb' mode writes binary data.

10. Checking and Deleting Files

- os.path.exists() checks if a file exists.
- os.remove() deletes a file if it exists.

11. Copying Files

- shutil.copy() duplicates a file.

12. Exception Handling in File Operations

- Using try-except prevents crashes due to missing files or permission issues.

CODE

```
import os

import shutil


# 1. Creating and Writing to a File ('w' Mode - Overwrites Content)

with open("example.txt", "w") as f:

    f.write("Hello, this is the first line.\n")

    f.write("File handling in Python is easy.\n")


# 2. Reading a File ('r' Mode - Reads Entire Content)

with open("example.txt", "r") as f:

    print("Reading Full File:\n", f.read())


# 3. Appending to a File ('a' Mode - Adds Content)

with open("example.txt", "a") as f:

    f.write("Appending a new line.\n")


# 4. Reading Line by Line

with open("example.txt", "r") as f:

    print("\nReading Line by Line:")

    for line in f:

        print(line.strip())


# 5. Using Readline() in a Loop

with open("example.txt", "r") as f:

    print("\nUsing readline() method:")

    while True:
        line = f.readline()
        if not line:
            break
        print(line.strip())

# 6. Writing Multiple Lines Using writelines()
lines = ["Python is powerful.\n", "File handling is useful.\n"]
with open("example.txt", "w") as f:
    f.writelines(lines)

# 7. Using seek() and tell()
with open("example.txt", "r") as f:
    f.seek(10)  # Move cursor to 10th byte
    print("\nReading from Position 10:", f.read(10))
    print("Current Position:", f.tell())

# 8. Truncating a File
with open("example.txt", "w") as f:
    f.write("Hello, World!")  # Writing new content
    f.truncate(5)  # Truncates to first 5 characters

with open("example.txt", "r") as f:
    print("\nAfter Truncation:", f.read())

# 9. Handling Binary Files
with open("image.jpg", "rb") as f:  # Reading binary data
    data = f.read()

with open("copy.jpg", "wb") as f:  # Writing binary data
    f.write(data)
```

```
# 10. Checking If a File Exists
if os.path.exists("example.txt"):
    print("\nFile exists!")

# 11. Copying a File
shutil.copy("example.txt", "backup.txt")
print("File copied successfully!")

# 12. Deleting a File
if os.path.exists("backup.txt"):
    os.remove("backup.txt")
    print("Backup file deleted.")

# 13. Exception Handling in File Operations
try:
    with open("nonexistent.txt", "r") as f:
        print(f.read())
except FileNotFoundError:
    print("\nError: File not found!")

print("\nFile Handling Operations Completed Successfully! 🚀 ")
```