**Basic File Operations in Python**

1. **Opening a file**: You can open a file using the open() function. It takes two arguments: the file name and the mode in which you want to open it.
   * 'r': Read mode (default) - opens the file for reading.
   * 'w': Write mode - opens the file for writing (and creates a new file if it doesn't exist).
   * 'a': Append mode - opens the file for appending new content.
   * 'r+': Read and Write mode - opens the file for both reading and writing.
2. **Reading from a file**: You can use the read() method to read the entire content, or readline() and readlines() to read lines from a file.
3. **Writing to a file**: You can use the write() method to write a string to a file.
4. **Closing a file**: It is important to close the file after you're done with it using close() to free up system resources.

**Example: File Handling in Python**

# Opening a file in write mode and writing to it

file = open("example.txt", "w")

file.write("Hello, this is a text file.\nWelcome to Python file handling!")

file.close() # Always close the file

# Opening a file in read mode and reading from it

file = open("example.txt", "r")

content = file.read() # Reading the entire file

print(content)

file.close()

# Opening a file in append mode

file = open("example.txt", "a")

file.write("\nAdding a new line to the file.")

file.close()

# Reading line by line

file = open("example.txt", "r")

for line in file:

print(line.strip()) # strip() removes trailing newline characters

file.close()

**Handling Files Safely with with Statement**

Instead of manually closing the file using close(), you can use the with statement. It automatically handles closing the file, even if an error occurs during the operation.

# Using the 'with' statement for file handling

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

content = file.read()

print(content)

**File Modes Summary:**

* 'r': Read (default mode) - the file must exist.
* 'w': Write - creates a new file or truncates an existing file.
* 'a': Append - creates a file if it doesn't exist, adds content at the end.
* 'r+': Read and write - the file must exist.

**What is File Handling?**

File handling refers to the process of managing files using Python. Python provides several built-in functions and methods to interact with files. The primary operations for file handling are:

* **Opening a file**
* **Reading from a file**
* **Writing to a file**
* **Closing a file**

We'll cover these concepts in detail, including the use of different file modes and working with text and binary files.

**1. Opening a File in Python**

To open a file, Python provides the open() function. This function takes two main arguments:

1. **File name (or path)**: The name of the file or the complete path to the file.
2. **Mode**: Specifies the mode in which the file should be opened. Common modes are:
   * 'r': Read mode (default) – Opens the file for reading.
   * 'w': Write mode – Opens the file for writing (creates the file if it doesn’t exist, or truncates the file if it exists).
   * 'a': Append mode – Opens the file for appending (adds data to the end of the file without truncating it).
   * 'r+': Read and Write mode – Opens the file for both reading and writing.

**Example: Opening a file**

python

Copy code

# Open a file for reading

file = open("example.txt", "r") # 'r' is the default mode

**2. Reading from a File**

Once a file is opened in read mode ('r'), you can read its content using various methods.

* **read()**: Reads the entire content of the file.
* **readline()**: Reads a single line from the file.
* **readlines()**: Reads all lines and returns them as a list.

**Example: Reading from a file**

python

Copy code

# Open the file in read mode

file = open("example.txt", "r")

# Read the entire file

content = file.read()

print(content)

# Reading line by line

file.seek(0) # Move the file pointer back to the start

for line in file:

print(line.strip()) # strip() removes trailing newline characters

file.close() # Always close the file after reading

**3. Writing to a File**

To write to a file, open the file in 'w' (write) or 'a' (append) mode. If you use 'w', it will overwrite the existing content. If you use 'a', it will add new content to the end of the file.

* **write()**: Writes a string to the file.
* **writelines()**: Writes a list of strings to the file.

**Example: Writing to a file**

python

Copy code

# Open the file in write mode

file = open("example.txt", "w")

# Write content to the file

file.write("Hello, World!\n")

file.write("This is a new line.")

file.close() # Always close the file after writing

# Now append more content to the file

file = open("example.txt", "a")

file.write("\nAppended another line.")

file.close()

**4. Closing a File**

You should always close a file after performing operations to free up system resources. Use the close() method to close a file.

**Example:**

python

Copy code

file = open("example.txt", "r")

content = file.read()

print(content)

file.close()

**5. Using with Statement for File Handling**

Instead of manually closing a file using close(), Python provides a more efficient way to handle files using the with statement. The with statement ensures that the file is closed automatically after the block of code is executed.

**Example: Using with for file handling**

python

Copy code

# Using 'with' statement for opening and closing a file automatically

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

content = file.read()

print(content)

You don't need to call file.close() when using with – it automatically closes the file when the block of code is done.

**6. File Modes in Detail**

| **Mode** | **Description** |
| --- | --- |
| 'r' | Opens the file for reading (default mode). |
| 'w' | Opens the file for writing. Creates the file if it does not exist, or truncates the file if it exists. |
| 'a' | Opens the file for appending. Data is written at the end of the file. |
| 'r+' | Opens the file for both reading and writing. The file must exist. |
| 'w+' | Opens the file for both reading and writing. Creates the file if it does not exist, or truncates if it does. |
| 'a+' | Opens the file for both reading and appending. If the file doesn’t exist, it creates a new one. |
| 'b' | Binary mode (e.g., 'rb', 'wb'). Used for working with non-text files like images or videos. |

**7. Working with Binary Files**

Binary files (like images or videos) can also be handled using the open() function by adding 'b' to the mode (e.g., 'rb' for reading a binary file or 'wb' for writing a binary file).

**Example: Reading and Writing Binary Files**

python

Copy code

# Open an image file in binary mode

with open("image.jpg", "rb") as image\_file:

content = image\_file.read() # Read the binary content

# Writing binary content to a new file

with open("copy\_image.jpg", "wb") as new\_image\_file:

new\_image\_file.write(content)

**8. File Pointer (seek() and tell() methods)**

When you open a file, a file pointer is positioned at the beginning. You can move the file pointer using the seek() method and check the current position using the tell() method.

* **seek(offset, from\_what)**: Moves the pointer to a specific position in the file.
* **tell()**: Returns the current position of the file pointer.

**Example:**

python

Copy code

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

print(file.tell()) # Output: 0 (pointer at the start)

file.read(5) # Read first 5 characters

print(file.tell()) # Output: 5 (pointer after the 5th character)

file.seek(0) # Move pointer back to the start

print(file.read(5)) # Output: First 5 characters

**9. Handling File Exceptions**

You can handle exceptions during file operations using try-except blocks. For instance, if a file doesn't exist and you try to open it in read mode, Python will raise a FileNotFoundError.

**Example: Handling exceptions**

python

Copy code

try:

with open("non\_existent\_file.txt", "r") as file:

content = file.read()

except FileNotFoundError:

print("File not found. Please check the file name or path.")

**Summary**

1. **Opening a file**: open()
2. **Reading from a file**: read(), readline(), readlines()
3. **Writing to a file**: write(), writelines()
4. **Appending to a file**: Open in 'a' mode
5. **Closing a file**: close(), or use with for automatic closing
6. **Handling binary files**: Open in 'b' mode
7. **Moving the file pointer**: seek(), tell()
8. **Handling exceptions**: Use try-except to handle file-related errors

**Conclusion**

File handling in Python is simple yet powerful. It allows you to perform various file operations like reading, writing, and appending in both text and binary formats. By understanding the different file modes and methods, you can effectively manage files in Python.