File Handling in Python

**Introduction**

File handling in Python refers to the process of interacting with files stored on the system. Python provides several built-in functions and methods to **read** from, **write** to, and **manage** files of different types (e.g., text files, binary files, CSV, JSON, etc.). This functionality is essential for tasks that involve persistent data storage, such as saving user data, reading configuration files, or processing large datasets.

**Opening a File**

The open() function is used to open a file in Python, which returns a file object. This file object is used to interact with the file (e.g., read from, write to, etc.).

**Syntax:**

file = open("filename", "mode")

**Parameters:**

* **filename**: The name of the file (can include the path).
* **mode**: The mode in which the file should be opened (e.g., 'r', 'w', 'a', 'b', etc.).

**Common File Modes:**

* **'r'**: Read (default). Opens the file for reading.
* **'w'**: Write. Creates the file if it doesn't exist or truncates it to zero length if it does.
* **'a'**: Append. Adds data to the end of the file if it exists; creates a new file if it doesn't.
* **'x'**: Exclusive creation. Fails if the file already exists.
* **'b'**: Binary mode (e.g., for images or audio files).
* **'t'**: Text mode (default).

**Example:**

file = open("example.txt", "r") # Open file in read mode

**Reading from a File**

After opening the file, we can read its contents using various methods depending on the requirement.

* **read()**: Reads the entire content of the file as a string.
* content = file.read()
* **readline()**: Reads a single line from the file. The cursor moves to the next line after reading.
* line = file.readline()
* **readlines()**: Reads all lines of the file and returns them as a list of strings.
* lines = file.readlines()

**Example:**

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

content = file.read()

print(content)

**Writing to a File**

To write data to a file, you need to open the file in write ('w'), append ('a'), or exclusive creation ('x') mode.

* **write()**: Writes a string to the file. If the file exists, the content will be overwritten.
* file.write("Hello, World!")
* **writelines()**: Writes a list of strings to the file.
* lines = ["Hello", "World"]
* file.writelines(lines)

**Example:**

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

file.write("This is a new file with new content.")

**Closing a File**

After performing file operations, it is crucial to close the file using the close() method to release system resources. Failure to close files may lead to memory or resource leaks.

file.close()

However, Python provides the **with** statement, which ensures that the file is automatically closed once the block is executed, even in case of errors.

**Using with Statement (Context Manager)**

The with statement is a context manager that simplifies file handling by automatically closing the file after the operations are completed, reducing the risk of errors.

**Syntax:**

with open("filename", "mode") as file:

# perform file operations here

**Example:**

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

content = file.read()

print(content)

# File is automatically closed after the block ends

**File Operations**

* **seek(offset, whence)**: Moves the cursor to a specified position in the file. The offset is the number of bytes, and whence is the reference point.
  + whence = 0: Starting from the beginning of the file (default).
  + whence = 1: Starting from the current position.
  + whence = 2: Starting from the end of the file.
* file.seek(0) # Move to the start of the file
* **tell()**: Returns the current position of the file cursor.
* position = file.tell()

**Error Handling in File Operations**

It's important to handle errors during file operations, such as missing files or permission issues, using try-except blocks.

**Example:**

try:

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

content = file.read()

except FileNotFoundError:

print("File not found.")

except IOError:

print("Error reading the file.")