File Handling in Python

File handling in Python is used to store and manipulate data in files. Python has several functions and modules that make file handling easy. Python also supports various file types like text files (.txt), binary files (.bin), and many more.

File Operations:

There are several basic file operations in Python, including:

1. Opening a file
2. Reading from a file
3. Writing to a file
4. Closing a file
5. Appending to a file

1. Opening a File

In Python, the `open()` function is used to open a file. It returns a file object, and it's commonly used with two arguments:

```
file_object = open(filename, mode)
```

filename: The name or path of the file.

mode: The mode in which the file should be opened.

File Modes:

- "r": Read (default mode). Opens the file for reading. Raises an error if the file does not exist.
- "w": Write. Opens the file for writing. Creates a new file if it doesn't exist or truncates the file if it exists.
- "a": Append. Opens the file for appending. Creates a new file if it doesn't exist.
- "x": Create. Creates a new file. Raises an error if the file already exists.
- "b": Binary mode. Opens the file in binary mode (for non-text files like images).
- "t": Text mode (default mode). Opens the file in text mode (for text files).
- "+": Read and write. Opens the file for both reading and writing.

```
# Opening a file in read mode
file = open("example.txt", "r")

# Opening a file in write mode
file = open("example.txt", "w")
```

Reading from a File

Python provides various methods to read from a file:

1. **read()**: Reads the entire content of the file as a string.
2. **readline()**: Reads one line from the file.
3. **readlines()**: Reads all lines of the file and returns them as a list of strings.

```
# Reading the entire file
file = open("example.txt", "r")
content = file.read()
print(content)
file.close()

# Reading one line at a time
file = open("example.txt", "r")
line = file.readline()
print(line)
file.close()

# Reading all lines into a list
file = open("example.txt", "r")
lines = file.readlines()
print(lines)
file.close()
```

Writing to a File

There are two primary methods for writing to a file:

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

```
# Writing to a file
file = open("example.txt", "w")
file.write("Hello, World!\n")
file.write("This is a new line.")
file.close()

# Writing multiple lines
lines = ["Line 1\n", "Line 2\n", "Line 3\n"]
file = open("example.txt", "w")
file.writelines(lines)
file.close()
```

Appending to a File

To append new content to an existing file without overwriting it, use the "a" mode.

```
file = open("example.txt", "a")
file.write("This line is appended to the file.\n")
file.close()
```

Closing a File

It's important to close a file after you're done working with it. This ensures that resources are freed and changes are saved.

Alternatively, Python provides a context manager (`with` statement) to automatically handle closing the file.

Example:

python Copy code

```python
with open("example.txt", "r") as file:
    content = file.read()
    print(content)
# No need to explicitly call file.close()
```

```python
file = open("example.txt", "r")
# Perform operations
file.close()  # Always close the file after use
```

File Pointer

The file pointer indicates the current position where reading or writing will happen in the file. Some methods to manipulate the file pointer include:

1. **seek(offset, from_what)**: Moves the file pointer to a specific position.
    - offset: Number of bytes to move.
    - from_what: Starting point (0 = start of the file, 1 = current position, 2 = end of the file).
2. **tell()**: Returns the current position of the file pointer.

```python
file = open("example.txt", "r")
print(file.tell())  # Outputs the current position (0)
file.seek(5)  # Move the pointer to the 5th byte
print(file.read())  # Read from the new position
file.close()
```

Binary File Handling

For non-text files like images or audio files, use binary mode "b". File operations on binary files work similarly to text files.

```
# Reading a binary file (image)
with open("image.jpg", "rb") as file:
    content = file.read()

# Writing to a binary file
with open("new_image.jpg", "wb") as file:
    file.write(content)
```

File Handling Functions

1. **os.remove()**: Deletes a file.
2. **os.rename()**: Renames a file.
3. **os.path.exists()**: Checks if a file exists.

```
import os

# Deleting a file
os.remove("example.txt")

# Renaming a file
os.rename("old_name.txt", "new_name.txt")

# Checking if a file exists
if os.path.exists("example.txt"):
    print("File exists")
else:
    print("File does not exist")
```

Common File Handling Methods

Method	Description
`open()`	Opens a file and returns a file object.
`read()`	Reads the entire file content.
`readline()`	Reads one line at a time from the file.
`readlines()`	Reads all lines into a list.
`write()`	Writes a string to the file.
`writelines()`	Writes multiple lines to the file.
`close()`	Closes the file object.
`seek()`	Moves the file pointer to a specific position.
`tell()`	Returns the current position of the file pointer.
`os.remove()`	Deletes a file.
`os.rename()`	Renames a file.
`os.path.exists()`	Checks if a file exists.

Practical Use Cases

1. **Logging System**:
    - Append log entries to a file using file handling to track application activity.
2. **Reading Configuration Files**:
    - Read configuration settings for an application from a text file.
3. **CSV File Processing**:
    - Handle comma-separated values (CSV) files for data storage and manipulation.
4. **Data Persistence**:
    - Save user preferences, application states, or temporary data to a file.
5. **File-based Data Storage**:

- Storing data in text or binary files when using a full-fledged database isn't necessary.