File Handling in Python

File handling in Python is essential for reading, writing, and manipulating files. Python provides built-in functions to handle files and offers a simple way to work with different types of files, such as text files (`.txt`), binary files, and more.

File Modes

When working with files, you need to specify how you want to interact with the file. Python uses file modes to define the purpose of opening the file. Below are the most commonly used modes:

- `'r'`: Read mode. Opens a file for reading. If the file doesn't exist, it raises an error.
- `'w'`: Write mode. Opens a file for writing. If the file exists, its contents are truncated (deleted). If the file doesn't exist, a new file is created.
- `'a'`: Append mode. Opens a file for appending data at the end. If the file doesn't exist, a new file is created.
- `'x'`: Exclusive creation mode. Opens a file for exclusive creation. If the file already exists, the operation fails.
- `'b'`: Binary mode. Used for working with binary files (e.g., images, videos). Can be combined with other modes.
- `'t'`: Text mode. The default mode for reading or writing text files. Can be combined with other modes.
- `'+'`: Update mode. Opens a file for both reading and writing. Can be combined with other modes (`r+`, `w+`, etc.).

Syntax for Opening a File

```
file = open('filename.txt', 'mode')
```

Example of Opening a File

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

Basic File Operations

1. **Opening and Closing Files**

- To open a file, use the `open()` function with the appropriate mode.
- After working with the file, you should close it using `file.close()` to free up system resources.

Example:

```python
# Open a file
file = open('example.txt', 'r')

# Close the file
file.close()
```

2. **Reading from a File**

- `read()`: Reads the entire file or a specific number of characters.
- `readline()`: Reads a single line from the file.
- `readlines()`: Reads all lines from the file and returns a list.

Example:

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

# Reading line by line
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()
for line in lines:
    print(line)
file.close()
```

3. **Writing to a File**

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

Example:

```python
# Writing to a file (overwrites if exists)
file = open('example.txt', 'w')
file.write("This is a new line.\n")
file.write("Writing another line.")
file.close()

# Appending to a file
```

```
file = open('example.txt', 'a')
file.write("\nAppending a line at the end.")
file.close()
```

4. **Using `with` Statement**

The `with` statement is used for managing file resources efficiently. It automatically closes the file after the block of code is executed, even if exceptions occur.

Example:

```
# Using 'with' to open and close file automatically
with open('example.txt', 'r') as file:
    content = file.read()
    print(content)
# No need to call file.close(), it is done automatically
```

5. **File Object Methods**

- `file.tell()`: Returns the current file pointer position.
- `file.seek(offset)`: Changes the file pointer to the given position.

Example:

```
file = open('example.txt', 'r')

# Get current file pointer position
position = file.tell()
print(f"Current position: {position}")

# Move the file pointer to the start
file.seek(0)
print(f"After seeking, position: {file.tell()}")

file.close()
```

Handling Binary Files

Binary files (e.g., images, videos) are handled differently. You can open them in binary mode using `'rb'`, `'wb'`, or `'ab'` for reading, writing, or appending.

Example:

```
# Reading a binary file (like an image)
with open('image.png', 'rb') as file:
    binary_data = file.read()
    print(binary_data)
```

Checking if a File Exists

You can check if a file exists using the `os` module or the `pathlib` module.

Using `os` module:

```
import os

if os.path.exists('example.txt'):
    print("File exists")
else:
    print("File does not exist")
```

Using `pathlib` module:

```
from pathlib import Path

file = Path('example.txt')
if file.is_file():
    print("File exists")
else:
    print("File does not exist")
```

File Handling Modes Comparison

Mode	Description
`'r'`	Read-only mode (default). Raises error if file doesn't exist.
`'w'`	Write mode. Creates a new file or truncates an existing file.
`'a'`	Append mode. Data is added to the end of the file.
`'x'`	Exclusive creation mode. Fails if file exists.
`'b'`	Binary mode. Used for non-text files.
`'t'`	Text mode (default). Used for text files.
`'+'`	Open file for updating (both read and write).