Day 12: File Handling (Material)

File handling in Python allows you to read from and write to files on your system. It is a core aspect of many applications, as it enables data persistence.

---

1. Reading from and Writing to Files

Opening a File

Python uses the built-in open() function to work with files. The syntax is:

open(file, mode)

- **file**: The name or path of the file.
- **mode**: The mode in which the file is opened.
    - 'r': Read (default mode).
    - 'w': Write (overwrites if file exists, creates if it doesn't).
    - 'a': Append (adds to the file, if it exists).
    - 'b': Binary mode.
    - 'x': Create (fails if the file exists).

Reading Files

1. **Reading the Entire File**
2. with open("example.txt", "r") as file:
3. content = file.read()
4. print(content)
    - **file.read()**: Reads the entire file content as a string.
5. **Reading Line by Line**
6. with open("example.txt", "r") as file:
7. for line in file:
8. print(line.strip()) # Removes any trailing newline or whitespace

9. **Reading a Specific Number of Characters**

```
with open("example.txt", "r") as file:
    part = file.read(10)  # Reads the first 10 characters
    print(part)
```

13. **Reading All Lines into a List**

```
with open("example.txt", "r") as file:
    lines = file.readlines()
    print(lines)  # Each line is stored as an item in a list
```

Writing to Files

1. **Overwriting the File**

```
with open("example.txt", "w") as file:
    file.write("This will overwrite the file content.\n")
```

4. **Appending to a File**

```
with open("example.txt", "a") as file:
    file.write("This will be added to the file.\n")
```

7. **Writing Multiple Lines**

```
with open("example.txt", "w") as file:
    lines = ["Line 1\n", "Line 2\n", "Line 3\n"]
    file.writelines(lines)
```

---

2. Context Manager (with Statement)

What is a Context Manager?

A context manager automatically handles opening and closing resources, like files, ensuring there are no memory leaks or issues due to forgetting to close a file.

Using with for File Handling

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

```
content = file.read()
print(content)
```

- **Advantages**:
  - Automatically closes the file after the block executes.
  - Ensures proper resource management.

How it Differs from open() Without with

Without with:

```
file = open("example.txt", "r")
try:
    content = file.read()
    print(content)
finally:
    file.close()
```

With with, this is handled implicitly, making your code cleaner and less error-prone.

Examples

1. **Copying File Content**

```
with open("source.txt", "r") as source, open("destination.txt", "w") as dest:
    for line in source:
        dest.write(line)
```

2. **Counting Words in a File**

```
with open("example.txt", "r") as file:
    content = file.read()
    words = content.split()
    print(f"Number of words: {len(words)}")
```

3. **Logging with Append Mode**

```
with open("log.txt", "a") as log:
log.write("This is a new log entry.\n")
```

4. **Reading a Binary File**

```
with open("image.jpg", "rb") as file:
data = file.read()
print(f"Read {len(data)} bytes.")
```

Important Points

1. **Error Handling**
   - Always use try-except blocks when working with file operations outside with to handle errors gracefully.

```
try:
    file = open("nonexistent.txt", "r")
except FileNotFoundError:
    print("File not found.")
finally:
    if file:
        file.close()
```

2. **File Modes**

Mode	Description
r	Read (default). File must exist.
w	Write. Overwrites the file or creates a new one.
a	Append. Adds data to the end of the file.
x	Create. Fails if the file exists.

Mode	Description
b	Binary mode. Used for non-text files.

3. **Efficient Resource Management** Always prefer the **with** statement for file handling as it minimizes the risk of resource leaks.

Real-World Use Cases

1. **Log Files**: Record user activity in a system log.
2. **Data Processing**: Read and process CSV or text data for analytics.
3. **File Backups**: Create automated scripts to copy or backup files.
4. **Configuration Management**: Store application settings in text files and read them during runtime.