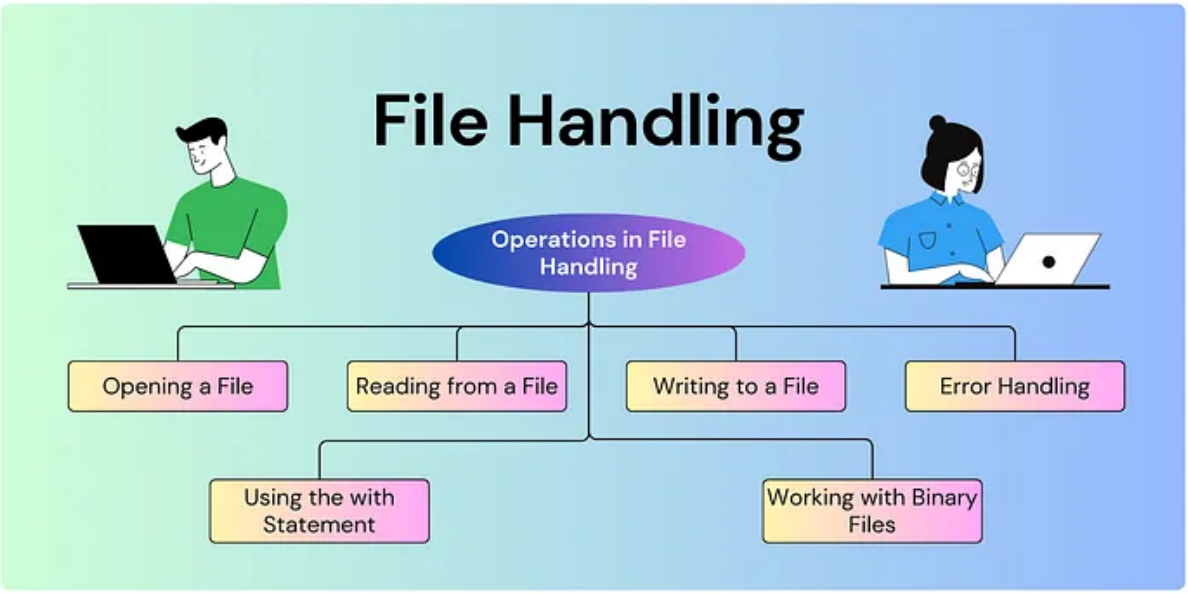
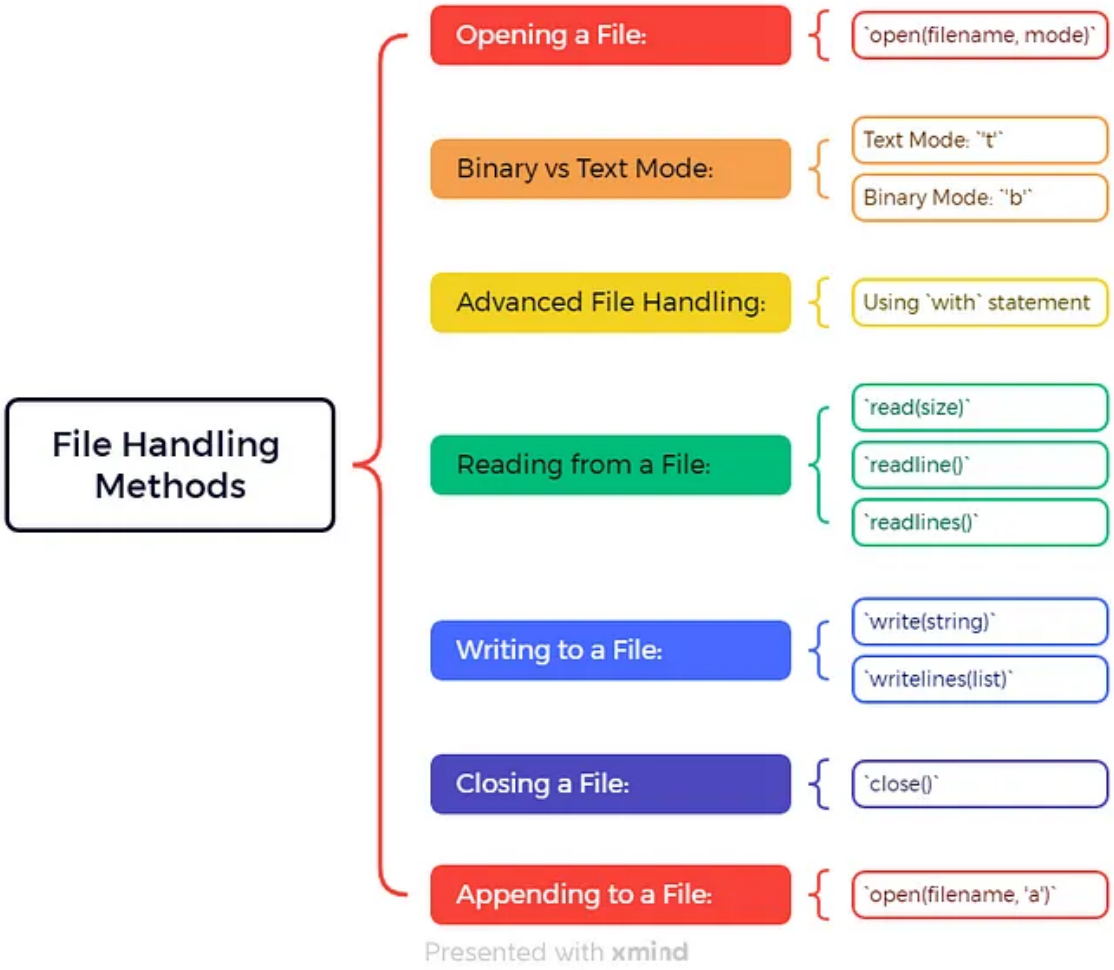
File Handling:





`open()` and `close()` in File Handling

`open()`

Description: The `open()` function is used to open a file and returns a file object, which provides methods for performing file operations such as reading, writing,

and closing.

Syntax:

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

- `'filename'`: The name of the file you want to open. This can be a relative or absolute path.
- `'mode'`: The mode in which the file is opened. Common modes include:
  - `'r'`: Read mode (default). Opens the file for reading. The file must exist.
  - `'w'`: Write mode. Opens the file for writing. Creates a new file or truncates an existing file.
  - `'a'`: Append mode. Opens the file for writing. Creates a new file if it doesn't exist and appends data to the end of the file.
  - `'b'`: Binary mode. Used with other modes to handle binary files.
  - `'t'`: Text mode (default). Used with other modes to handle text files.

Example:

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

# Open a file in write mode
file = open('example.txt', 'w')
print("File opened in write mode.")

# Open a file in append mode
file = open('example.txt', 'a')
print("File opened in append mode.")
```

```
# Open a binary file
file = open('example.bin', 'wb')
print("Binary file opened for writing.")
```

`close()`

Description: The `close()` method is used to close the file. Closing a file is important to free up system resources and to ensure that all data is properly written to the file.

Syntax:

```
pythonCopy code
file.close()
```

Writing to Files

1. Write Mode (`'w'`)

Overwrites the file if it exists, otherwise creates a new one.

```
pythonCopy code
file = open('example.txt', 'w')
file.write("This is written in write ('w') mode.\n")
file.close()
```

2. Append Mode (`'a'`)

Appends content to the end of the file without overwriting it.

```
pythonCopy code
file = open('example.txt', 'a')
file.write("This is appended in append ('a') mode.\n")
file.close()
```

3. Write Binary Mode (`'wb'`)

Writes binary content to a file.

```
pythonCopy code
file = open('example.bin', 'wb')
file.write(b"This is binary content written in 'wb' mode.\n")
file.close()
```

4. Writing Multiple Lines

Use `.writelines()` to write multiple lines at once from a list of strings.

```
pythonCopy code
file = open('example.txt', 'w')
lines = ["First line\n", "Second line\n", "Third line\n"]
file.writelines(lines)
file.close()
```

`read()`

The `read()` method reads the entire content of a file and returns it as a single string.

```
pythonCopy code
# Open the file in read mode
```

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

# Read the entire content of the file
content = file.read()

print("Content read using read():")
print(content)

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

`readline()`

The `readline()` method reads a single line from the file. If you call it repeatedly, it will read each line one by one.

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

# Read the first line
line1 = file.readline()
print("First line read using readline():")
print(line1)

# Read the second line
line2 = file.readline()
print("Second line read using readline():")
print(line2)

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

`readlines()`

The `readlines()` method reads all lines of the file and returns them as a list of strings.

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

# Read all lines into a list
lines = file.readlines()

print("Lines read using readlines():")
for line in lines:
    print(line, end='')  # end='' to avoid adding extra newli
nes

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

Summary

- `read()`: Reads the entire file content into a single string.
- `readline()`: Reads one line at a time from the file.
- `readlines()`: Reads all lines into a list, where each element is a line from the file.

Context Managers

Description: Context managers in Python are used to handle resources efficiently and ensure proper cleanup. The `with` statement is commonly used with file handling to automatically close the file after the block of code is executed.

Syntax:

```
with open('filename', 'mode') as file:
    # Perform file operations
    pass
# File is automatically closed here
```

Examples:

Example 1: Basic File Handling with Context Manager

Description: Open a file, read its content, and ensure it is closed properly after operations.

```
with open('example.txt', 'r') as file:
    content = file.read()
    print(content)
# File is automatically closed here
```

Example 2: Writing to a File with Context Manager

Description: Open a file in write mode, write some content, and ensure it is closed properly.

```
with open('example.txt', 'w') as file:
    file.write("Hello, World!\n")
    file.write("This is a new line.\n")
# File is automatically closed here
```

Example 3: Appending to a File with Context Manager

Description: Open a file in append mode, add new content, and ensure it is closed properly.

```
with open('example.txt', 'a') as file:
    file.write("Appending this line.\n")
# File is automatically closed here
```

Example 4: Reading Large Files in Chunks with Context Manager

Description: Efficiently read large files in chunks to manage memory usage.

```
with open('large_file.txt', 'r') as file:
    while True:
        chunk = file.read(1024)  # Read 1024 bytes at a time
        if not chunk:
            break
        print(chunk)
# File is automatically closed here
```

6. Working with Binary Files

Example 1: Writing Binary Data

Writing a byte sequence to a binary file:

```
pythonCopy code
# Writing binary data to a file
with open('example.bin', 'wb') as file:
    file.write(b'\xDE\xAD\xBE\xEF')  # Write hexadecimal byte
s to the file
```

```
print("Binary data written to example.bin")
```

Explanation: This code opens `example.bin` in binary write mode (`'wb'`) and writes a sequence of bytes to it. The byte sequence `b'\xDE\xAD\xBE\xEF'` is written to the file.

Example 2: Reading Binary Data

Reading binary data from a file:

```
pythonCopy code
# Reading binary data from a file
with open('example.bin', 'rb') as file:
    content = file.read()  # Read the entire content of the f
ile
    print(content)         # Output the content (in binary fo
rmat)

print("Binary data read from example.bin")
```

Explanation: This code opens `example.bin` in binary read mode (`'rb'`) and reads the entire content of the file, printing it out in binary format.

7. Handling CSV Files

Example 1: Reading CSV Files

Reading CSV data and printing each row:

```
pythonCopy code
import csv

# Reading data from a CSV file
with open('example.csv', 'r') as file:
    reader = csv.reader(file)
```

```
    for row in reader:
        print(row)  # Print each row from the CSV file

print("CSV data read from example.csv")
```

Explanation: This code opens `example.csv` in read mode (`'r'`) and uses `csv.reader` to read the file. It then prints each row of the CSV file.

Example 2: Writing to CSV Files

Writing a list of data to a CSV file:

```
pythonCopy code
import csv

# Data to be written to the CSV file
data = [
    ["Name", "Age", "City"],
    ["Alice", 30, "New York"],
    ["Bob", 25, "San Francisco"]
]

# Writing data to a CSV file
with open('example.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(data)  # Write multiple rows to the CSV
file

print("Data written to example.csv")
```

Explanation: This code opens `example.csv` in write mode (`'w'`) and uses `csv.writer` to write a list of lists (rows) to the CSV file.

8. Handling JSON Files

Example 1: Reading JSON Files

Reading JSON data and printing it:

```
pythonCopy code
import json

# Reading data from a JSON file
with open('example.json', 'r') as file:
    data = json.load(file)  # Load JSON data from the file
    print(data)             # Print the JSON data

print("JSON data read from example.json")
```

Explanation: This code opens `example.json` in read mode (`'r'`) and uses `json.load` to read and parse the JSON data from the file.

Example 2: Writing JSON Files

Writing a Python dictionary to a JSON file:

```
pythonCopy code
import json

# Data to be written to the JSON file
data = {
    "name": "Alice",
    "age": 30,
    "city": "New York"
}

# Writing data to a JSON file
with open('example.json', 'w') as file:
    json.dump(data, file, indent=4)  # Dump JSON data to the
file with pretty printing
```

```
print("Data written to example.json")
```

Pickling and Unpickling in Python

Pickling is the process of converting a Python object into a byte stream, while **unpickling** is the reverse process: converting a byte stream back into a Python object. This is useful for saving and loading Python objects.

Pickling

To pickle an object, use the `pickle` module:

```
pythonCopy code
import pickle

# Data to pickle
data = {'name': 'Alice', 'age': 30, 'city': 'New York'}

# Pickle the data
with open('data.pkl', 'wb') as file:
    pickle.dump(data, file)
```

Unpickling

To unpickle (load) the object:

```
pythonCopy code
import pickle

# Unpickle the data
with open('data.pkl', 'rb') as file:
    data = pickle.load(file)
```

```
print(data)
```

Pickling and Unpickling Example

Pickling a Set

Example: Pickling a Set

```
pythonCopy code
import pickle

# Data to be pickled (a set of integers)
data = {1, 2, 3, 4, 5}

# Serialize the set to a binary file
with open('set_data.pkl', 'wb') as file:
    pickle.dump(data, file)

print("Set pickled to binary file.")
```

Unpickling a Set

```
pythonCopy code
import pickle

# Deserialize the set from the binary file
with open('set_data.pkl', 'rb') as file:
    data = pickle.load(file)
```

```
print("Set unpickled from binary file:", data)
```

In these examples, a set of integers is pickled to a binary file and then unpickled to retrieve

Serialization and Deserialization in Python

Serialization is the process of converting an object into a format that can be easily stored or transmitted (e.g., JSON, XML, binary). **Deserialization** is the process of converting that format back into an object.

JSON Serialization and Deserialization

JSON is a common format for serialization and deserialization due to its readability and wide support.

Serialization to JSON:

```
pythonCopy code
import json

# Data to serialize
data = {
    "name": "Alice",
    "age": 30,
    "city": "New York"
}

# Serialize to JSON
with open('data.json', 'w') as file:
    json.dump(data, file, indent=4)
```

Deserialization from JSON:

```
pythonCopy code
import json

# Deserialize from JSON
with open('data.json', 'r') as file:
    data = json.load(file)

print(data)
```

1. CSV Serialization and Deserialization

CSV (Comma Separated Values) is a simple and commonly used format for tabular data.

Example 1: Serialization to CSV

```
pythonCopy code
import csv

data = [
    ["Name", "Age", "City"],
    ["Eve", 28, "Boston"],
    ["Frank", 33, "Chicago"]
]

# Serialize data to a CSV file
with open('data.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(data)

print("Data serialized to CSV.")
```

Example 2: Deserialization from CSV

```
pythonCopy code
import csv

# Read CSV data from a file and deserialize it
with open('data.csv', 'r') as file:
    reader = csv.reader(file)
    data = [row for row in reader]

print("Data deserialized from CSV:", data)
```

`os` Module Functions

1. `os.path.exists(path)`: Checks if a path exists.

   ```
   pythonCopy code
   exists = os.path.exists('filename')
   ```

2. `os.path.isfile(path)`: Checks if a path is a file.

   ```
   pythonCopy code
   is_file = os.path.isfile('filename')
   ```

3. `os.path.isdir(path)`: Checks if a path is a directory.

   ```
   pythonCopy code
   is_dir = os.path.isdir('directory')
   ```

4. `os.path.join(path, *paths)`: Joins one or more path components.

```
full_path = os.path.join('folder', 'subfolder', 'file.tx
t')
```

5. `os.path.abspath(path)`: Returns the absolute path of a file or directory.

```
abs_path = os.path.abspath('filename')
```

6. `os.mkdir(path)`: Creates a directory.

```
pythonCopy code
os.mkdir('new_directory')
```

7. `os.makedirs(path)`: Creates directories recursively.

```
os.makedirs('new_directory/subdirectory')
```

8. `os.rmdir(path)`: Removes a directory (must be empty).

```
pythonCopy code
os.rmdir('empty_directory'
```

9. `os.removedirs(path)`: Removes directories recursively (if empty).

```
pythonCopy code
os.removedirs('new_directory/subdirectory')
```

10. `os.rename(src, dst)`: Renames a file or directory.

```
pythonCopy code
os.rename('old_name.txt', 'new_name.txt')
```

11. `os.listdir(path)`: Lists files and directories in a directory.

```
files = os.listdir('directory')
```

`pathlib` Module Methods

1. `Path(path)`: Creates a `Path` object.

```
pythonCopy code
from pathlib import Path
path = Path('filename')
```

2. `Path.exists()`: Checks if a path exists.

```
pythonCopy code
exists = path.exists()
```

3. `Path.is_file()`: Checks if the path is a file.

```
pythonCopy code
is_file = path.is_file()
```

4. `Path.is_dir()`: Checks if the path is a directory.

```
pythonCopy code
is_dir = path.is_dir()
```

5. `Path.mkdir(parents=False, exist_ok=False)`: Creates a directory.

```
pythonCopy code
path.mkdir()
```

6. `Path.rmdir()`: Removes a directory (must be empty).

```
pythonCopy code
path.rmdir()
```

7. `Path.rename(target)`: Renames or moves a file or directory.

```
pythonCopy code
path.rename('new_name.txt')
```

Common File Handling Errors and Examples

1. FileNotFoundError

This error occurs when trying to open a file that does not exist.

Example:

```
pythonCopy code
try:
    with open('nonexistent_file.txt', 'r') as file:
```

```
        content = file.read()
except FileNotFoundError:
    print("Error: The file does not exist.")
```

2. PermissionError

This error occurs when the program does not have the necessary permissions to access or modify the file.

Example:

```
pythonCopy code
try:
    # Attempt to open a file with read-only permission for wr
iting
    with open('readonly_file.txt', 'w') as file:
        file.write("Some text")
except PermissionError:
    print("Error: You do not have permission to modify this f
ile.")
```

4. IsADirectoryError

This error occurs when attempting to open a directory as a file.

Example:

```
import os

# Ensure 'example_dir' is a directory for this test
os.makedirs('example_dir', exist_ok=True)

# Attempt to open the directory 'example_dir' as if it were a fi
```

```
with open('example_dir', 'r') as file:
    content = file.read()
```

5. ValueError

This error can occur if an invalid file mode is specified.

Example:

```
try:
    # Attempt to open a file with an invalid mode
    with open('example.txt', 'invalid_mode') as file:
        content = file.read()
except ValueError:
    print("Error: Invalid file mode specified.")
```

6. EOFError

This error occurs when reading beyond the end of a file or when an unexpected end of file is reached.

Example:

```
try:
    with open('example.txt', 'r') as file:
        # Attempt to read beyond the end of the file
        content = file.read()
        extra = file.read()  # This may raise EOFError if no
more data
except EOFError:
    print("Error: Unexpected end of file.")
```

7. FileExistsError

This error occurs when trying to create a file that already exists when using exclusive creation mode.

Example:

```
pythonCopy code
try:
    # Attempt to create a file that already exists with 'x' m
ode
    with open('existing_file.txt', 'x') as file:
        file.write("Some text")
except FileExistsError:
    print("Error: The file already exists.")
```

Why Use `'x'` Mode

- **Prevent Overwriting**: If you want to ensure that your program does not overwrite any existing files, use `'x'`. This can be useful in situations where the integrity of pre-existing data is important.
- **Safety in File Creation**: It is especially useful in scenarios where file creation should only happen if the file does not already exist, such as logging or generating reports.

When to Use `'x'` Mode:

- You should use `'x'` mode when you need to ensure that a file is created only if it does not exist, and you do not want to overwrite any existing file by mistake.