File Handling - Reading and Writing Binary Data

October 14, 2025

Writing and Reading Binary Data

Objective: To understand how to open, write, and read data from a file in binary mode.

Theory: Binary files store data in the form of bytes, not human-readable text. Examples include images, audio files, and executables. When working with binary data, you must open the file in **binary mode** by adding a 'b' to the mode string (e.g., 'wb' for write-binary, 'rb' for read-binary).

- **Writing Binary Data:** You must convert the data (like text strings or numbers) into bytes before writing to the file. Python's built-in bytes() function or a string's encode() method can be used for this.
- **Reading Binary Data:** When you read from a binary file, the data comes back as a bytes object. You must then decode it back into a string using the decode() method if it contains text.

Program:

```
# Create a sample text string and an integer.
text_data = "This is a secret message."
integer_data = 12345

# --- Writing to a Binary File ---
# Open the file in 'write binary' mode ('wb').
# 'wb' creates a new file or overwrites an existing one.
try:
   with open("binary_file.bin", "wb") as file:
      print("Writing data to 'binary_file.bin'...")
      # Encode the string into bytes using UTF-8.
      file.write(text_data.encode('utf-8'))

      # Binary data is often structured. For simplicity, we'll encode
      # the integer as a string and then as bytes.
      file.write(str(integer_data).encode('utf-8'))
```

```
    print("Data written successfully.")


except IOError as e:
    print(f"Error writing to file: {e}")



# --- Reading from the Binary File ---
# Open the file in 'read binary' mode ('rb').
try:
    with open("binary_file.bin", "rb") as file:
        print("Reading data from 'binary_file.bin'...")
        # Read all bytes from the file.
        read_data = file.read()

        # Decode the bytes back to a string using UTF-8.
        decoded_data = read_data.decode('utf-8')

    print("Data read successfully.")
    print(f"Content read from file: {decoded_data}")

except IOError as e:
    print(f"Error reading from file: {e}")
```

Output:

```
Writing data to 'binary_file.bin'...
Data written successfully.
Reading data from 'binary_file.bin'...
Data read successfully.
Content read from file: This is a secret message.12345
```

Appending to a Binary File

Objective: To learn how to add new data to an existing binary file without overwriting its content.

Theory: To add new data to the end of a binary file, you must open it in **append-binary mode** ('ab'). This mode moves the file pointer to the end of the file before writing, so any new data is simply added after the existing content.

Program:

```
# Assume 'binary_file.bin' from Experiment 1 exists.
new_data = " -- A new message appended."


# --- Appending to a Binary File ---
# Open the file in 'append binary' mode ('ab').
try:
    with open("binary_file.bin", "ab") as file:
        print("Appending new data to 'binary_file.bin'...")
        # Encode and write the new data.
        file.write(new_data.encode('utf-8'))

    print("Data appended successfully.")

except IOError as e:
    print(f"Error appending to file: {e}")



# --- Reading the updated Binary File ---
# Open in 'read binary' mode to see the combined content.
try:
    with open("binary_file.bin", "rb") as file:
        read_data = file.read()
        decoded_data = read_data.decode('utf-8')

    print("\nReading the entire file again:")
```

```
    print(f"Updated content: {decoded_data}")


except IOError as e:
    print(f"Error reading from file: {e}")
```

Output:

```
Appending new data to 'binary_file.bin'...
Data appended successfully.


Reading the entire file again:
Updated content: This is a secret message.12345 -- A new message appended.
```