Writing and Parsing Text Files

October 15, 2025

Writing and Appending to a Text File

Objective: To learn how to create and write content to a text file in Python, and how to add new data to it without overwriting existing content.

Theory: File handling is a fundamental part of programming. When working with text files, you use specific modes to tell Python how to interact with the file.

- **'w' (Write Mode):** Opens a file for writing. If the file already exists, its content is completely overwritten. If it doesn't exist, a new file is created.
- **'a' (Append Mode):** Opens a file for writing. If the file exists, new content is added to the end of the file. If it doesn't exist, a new file is created. The open() function is used to get a file object, and the with statement is the recommended way to handle files, as it ensures the file is automatically closed, even if errors occur.

Program:

```
# --- Writing to a new file ---
# Open the file in 'write' mode ('w').
# 'with open(...)' ensures the file is closed automatically.
try:
    with open("notes.txt", "w") as file:
        print("Writing to 'notes.txt'...")
        file.write("Python Programming Notes\n")
        file.write("1. Functions\n")
        file.write("2. Classes\n")
    print("Content written successfully.")
except IOError as e:
    print(f"Error writing to file: {e}")


# --- Appending to the same file ---
# Open the file in 'append' mode ('a').
try:
    with open("notes.txt", "a") as file:
```

```
        print("\nAppending to 'notes.txt'...")
        file.write("3. File Handling\n")
    print("Content appended successfully.")
except IOError as e:
    print(f"Error appending to file: {e}")
```

Output:

```
Writing to 'notes.txt'...
Content written successfully.


Appending to 'notes.txt'...
Content appended successfully.
```

Reading and Parsing a Text File

Objective: To learn how to read data from a text file and parse its content for use in a program.

Theory: To read a file, you open it in **read mode ('r')**. You can read the entire content at once or process it line by line. Processing line by line is generally more memory-efficient for very large files.

- **read():** Reads the entire file content into a single string.
- **readline():** Reads a single line from the file.
- **readlines():** Reads all lines from the file into a list of strings.
- **Looping over the file object:** A common and efficient way to read a file line by line.

Parsing involves processing the read data to extract meaningful information, often by splitting strings or converting data types.

Program:

```
# The 'notes.txt' file from Experiment 1 now contains all the notes.
# We will read this file and parse its contents.
# --- Reading and parsing line by line ---
try:
    print("Reading and parsing 'notes.txt' line by line:")
```

```
    with open("notes.txt", "r") as file:
        # Looping over the file object is an efficient way to read line by line.
        for line_number, line in enumerate(file, 1):
            # The 'line' variable includes a newline character, so we use .strip().
            clean_line = line.strip()

            # Check if the line is not empty before parsing.
            if clean_line:
                if clean_line.startswith("Python"):
                    print(f"Header: {clean_line}")
                elif clean_line.startswith("3."):
                    # Parsing the line to extract the topic.
                    parts = clean_line.split(". ")
                    print(f"Topic {parts[0]}: {parts[1]}")
                else:
                    print(f"Normal line: {clean_line}")

except FileNotFoundError:
    print("The file 'notes.txt' was not found.")
except IOError as e:
    print(f"Error reading the file: {e}")
```

Output:

```
Reading and parsing 'notes.txt' line by line:
Header: Python Programming Notes
Normal line: 1. Functions
Normal line: 2. Classes
Topic 3: File Handling
```