Task 1: Read and Display Contents of a File

Objective: Learn how to open, read, and display the contents of a text file.

Steps:

1. Create a text file named `sample.txt` and write a few lines of text in it.
2. Write a Python script to open the file in read mode (`'r'`).
3. Use the `read()` or `readline()` function to read the content of the file.
4. Print the content to the console.
5. Close the file after reading.

Question:

1. What function is used to open a file in Python?
2. What does the `read()` method do?
3. What happens if you try to read a file that doesn't exist? How can you handle this situation in your program?

Task 2: Write Data to a New File

Objective: Learn how to write data to a new file using Python.

Steps:

1. Create a Python script that creates a new file named `report.txt`.
2. Open the file in write mode (`'w'`).
3. Write a few lines of text to the file using the `write()` or `writelines()` method.
4. Save and close the file after writing.

Question:

1. What is the difference between the `'w'` and `'a'` file modes in Python?
2. How can you ensure that your file contains the correct data after writing to it?

Task 3: Append Data to an Existing File

Objective: Understand how to add new content to an existing file without overwriting its current content.

Steps:

1. Open an existing file (`data.txt`) in append mode (`'a'`).
2. Write additional content to the file using the `write()` method.
3. Save and close the file.
4. Verify that the new content is appended to the end of the file.

Question:

1. What happens if you open a file in `'w'` mode that already exists?
2. How can you check if the file exists before attempting to append data to it?

Task 4: Read a File Line by Line

Objective: Learn how to process large files line by line without consuming too much memory.

Steps:

1. Open a large text file (`large_file.txt`) in read mode (`'r'`).
2. Use a loop to read the file line by line.
3. Print each line as it is read.
4. After processing, close the file.

Question:

1. What is the advantage of reading a file line by line using a loop instead of reading it all at once?
2. How would you modify the program to count the number of lines in the file?

Task 5: Modify Content in a File

Objective: Learn how to modify the content of a file, such as replacing a word or updating a line.

Steps:

1. Open the file `text_file.txt` in read mode (`'r'`) and read its content.
2. Replace a specific word in the content using the `replace()` method.
3. Open the same file in write mode (`'w'`).
4. Write the modified content back to the file.
5. Close the file after writing.

Question:

1. How does the `replace()` method work in Python? What does it return?
2. What precautions should you take when modifying files directly, such as creating backups?

Task 6: Handle File Not Found Error

Objective: Learn how to handle errors related to non-existing files.

Steps:

1. Write a Python program that attempts to open a file named `missing_file.txt` in read mode (`'r'`).
2. Use a `try-except` block to handle the case where the file does not exist.
3. Print an appropriate message to the user when the file is not found.

Question:

1. What type of error is raised when attempting to open a non-existing file?
2. How can you handle this error gracefully to improve the user experience?

Task 7: Copy the Content of One File to Another

Objective: Learn how to copy content from one file to another.

Steps:

1. Open the source file `source.txt` in read mode (`'r'`).
2. Open the destination file `destination.txt` in write mode (`'w'`).
3. Read the content of the source file and write it to the destination file.
4. Close both files after the operation.

Question:

1. What function do you use to read the entire content of a file in Python?
2. How would you handle the situation where the destination file already exists?

Task 8: Create a Simple Log File

Objective: Learn how to create and append logs to a file.

Steps:

1. Open a log file (`logfile.txt`) in append mode (`'a'`).
2. Write a timestamped log message to the file indicating an event, such as "Process started" or "Error encountered."
3. Close the file after logging the message.

Question:

1. How would you format the timestamp for logging purposes in Python?
2. What considerations should be made when writing log files, such as managing file size or creating log rotations?

Task 9: Read and Write Binary Files

Objective: Learn how to handle binary files such as images or audio files.

Steps:

1. Open an image file (`image.png`) in binary read mode (`'rb'`).
2. Read the binary data from the file.
3. Open a new file (`copy_image.png`) in binary write mode (`'wb'`).
4. Write the binary data to the new file.
5. Close both files.

Question:

1. What does the `'rb'` mode do when opening a file?
2. How is handling binary data different from handling text data?

Task 10: Use Context Manager for File Operations

Objective: Understand how to use the `with` statement to automatically manage file resources.

Steps:

1. Open a file (`notes.txt`) using the `with` statement in read mode.
2. Read and print the contents of the file.
3. The file will automatically close after the block of code finishes executing.

Question:

1. Why is it better to use the `with` statement rather than manually opening and closing files?
2. How does the `with` statement ensure that a file is closed, even if an error occurs during file operations?