Assignmnet-1 : Reading and Writing Text Files

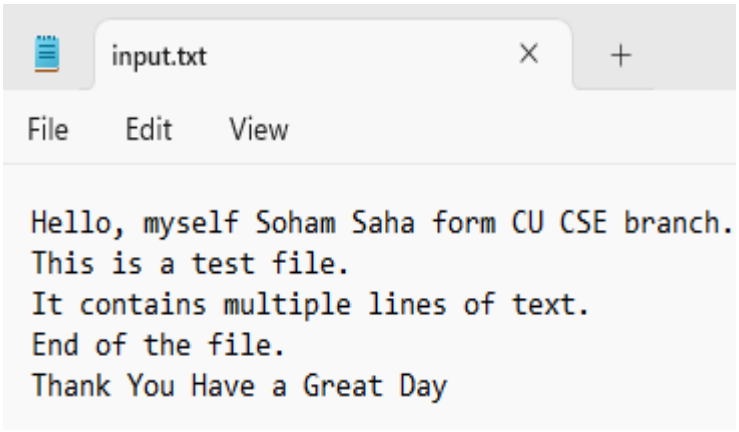
- **Problem**: Write a program to read from one text file and write its contents to another file.
- **Example**: Given a text file `input.txt`, create a new file `output.txt` with the same content.
- **Objective**: Read from a file, process the data if needed, and write it to a new file.

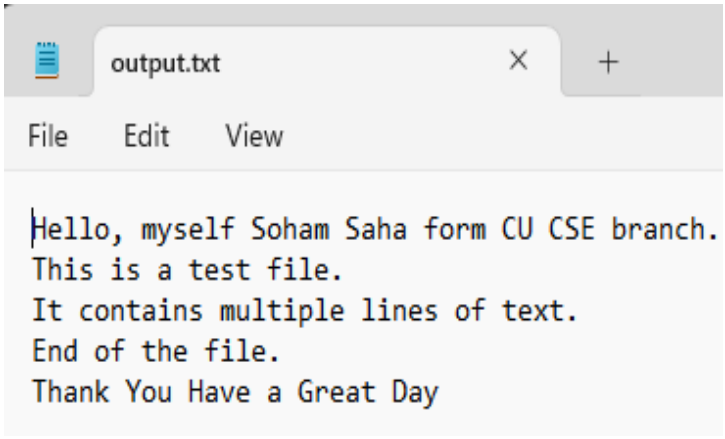
Program code :

```
with open('input.txt', 'r') as input_file:
    # Read the contents of the input file
    content = input_file.read()

# Open the output file in write mode
with open('output.txt', 'w') as output_file:
    # Write the content to the output file
    output_file.write(content)
print("Contents of 'input.txt' have been successfully copied to 'output.txt'.")
```

Output :

$ python "d:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\tempCodeRunnerFile.py"
Contents of 'input.txt' have been successfully copied to 'output.txt'.





Assignmnet-2 : Count Word Frequency in a Text File

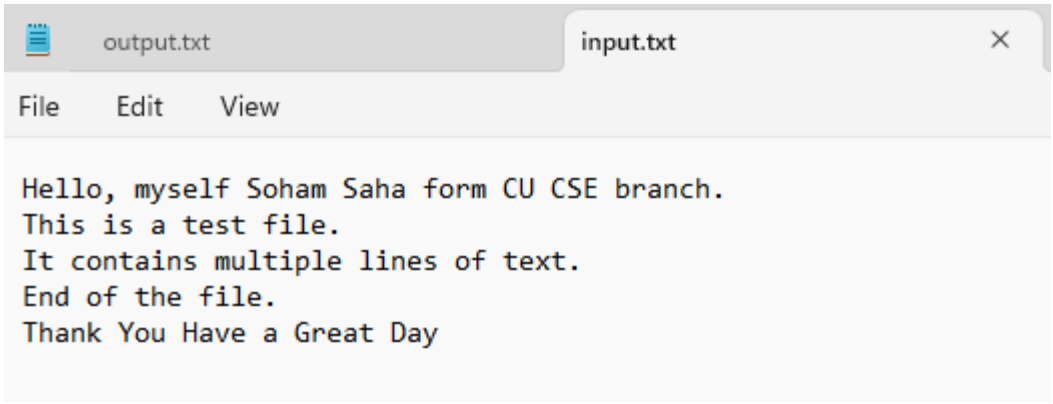
- **Problem**: Write a Python program to read a text file and calculate the frequency of each word in the file.
- **Example**: Given a text file, count how many times each word appears and display the results in descending order of frequency.
- **Objective**: Read file contents, split the text into words, and count word frequencies using dictionaries.

Program code :

```
from collections import Counter
# Open the input file in read mode
with open('input.txt', 'r') as file:
    # Read the contents of the file
    text = file.read()
# Split the text into words (removing punctuation and converting to lowercase)
words = text.lower().split()
# Use Counter to count the frequency of each word
word_counts = Counter(words)
# Sort the words by frequency in descending order
sorted_word_counts = sorted(word_counts.items(), key=lambda x: x[1], reverse=True)
# Display the word frequencies
print("Word Frequencies (Descending Order):")
for word, count in sorted_word_counts:
    print(f"{word}: {count}")
```

Output :

```
$ python "d:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\assg2.py"
Word Frequencies (Descending Order):
a: 2
file.: 2
of: 2
hello,: 1
myself: 1
soham: 1
saha: 1
form: 1
cu: 1
cse: 1
branch.: 1
this: 1
is: 1
test: 1
it: 1
contains: 1
multiple: 1
lines: 1
text.: 1
end: 1
the: 1
thank: 1
you: 1
have: 1
great: 1
day: 1
```



Assignmnet-3 : Merging Multiple Text Files into One

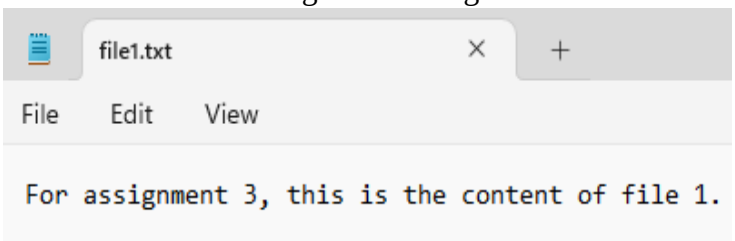
- **Problem**: Merge the contents of multiple text files into a single file.
- **Example**: Given three text files (`file1.txt`, `file2.txt`, `file3.txt`), merge their contents into a new file `merged.txt`.
- **Objective**: Open and read multiple files, then write their contents into a single file.

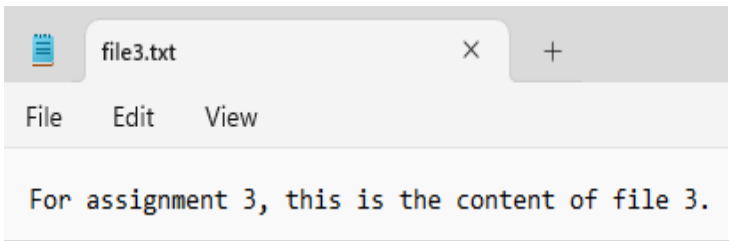
Program Code :

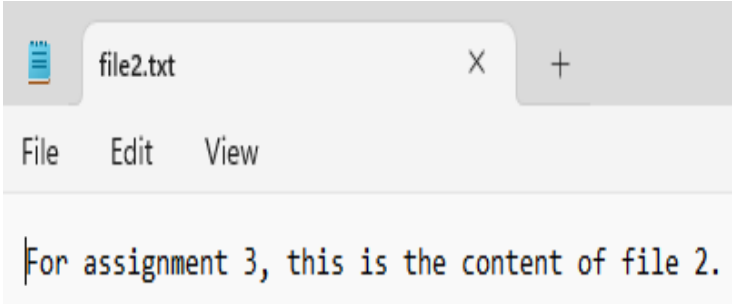
```
# List of input files to merge
input_files = ['file1.txt', 'file2.txt', 'file3.txt']
# Name of the output file
output_file = 'merged.txt'
# Open the output file in write mode
with open(output_file, 'w') as outfile:
    # Loop through each input file
    for file in input_files:
        try:
            # Open the input file in read mode
            with open(file, 'r') as infile:
                # Read and write the content to the output file
                content = infile.read()
                outfile.write(content)
                # Add a newline to separate files (optional)
                outfile.write('\n')
            print(f"Successfully merged {file}")
        except FileNotFoundError:
            print(f"File not found: {file}")

print(f"All files have been merged into {output_file}.")
```

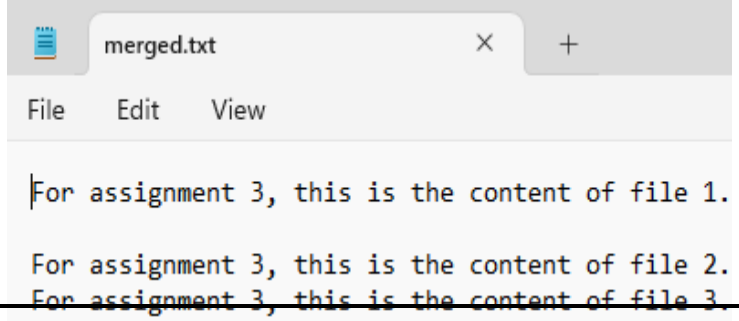
Output :

```
$ python "d:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\tempCodeRunnerFile.py"
Successfully merged file1.txt
Successfully merged file2.txt
Successfully merged file3.txt
All files have been merged into merged.txt.
```









Assignmnet-4 : File Comparison

- **Problem**: Write a Python program to compare two text files and highlight the differences between them.
- **Example**: Compare `file1.txt` and `file2.txt` line by line and print the differences.
- **Objective**: Read both files, compare their contents, and identify the differences (line by line or word by word).

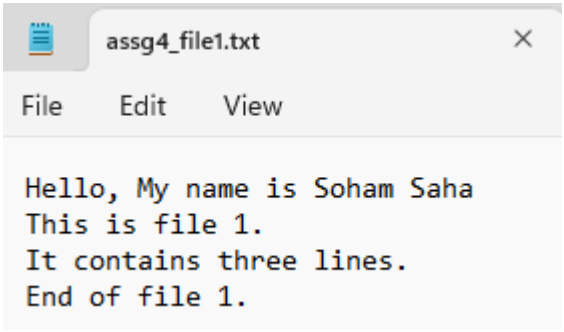
Program code :

```
from difflib import Differ
# Function to compare two files line by line
def compare_files(file1, file2):
    try:
        # Open both files in read mode
        with open(file1, 'r') as f1, open(file2, 'r') as f2:
            # Read all lines from both files
            lines1 = f1.readlines()
            lines2 = f2.readlines()

        print(f"Comparing '{file1}' and '{file2}':\n")

        # Get the maximum number of lines from both files
        max_lines = max(len(lines1), len(lines2))

        # Compare files line by line
        print("Line-by-Line Comparison:\n")
        for i in range(max_lines):
            # Get lines from each file, defaulting to an empty string if the line doesn't exist
            line1 = lines1[i].strip() if i < len(lines1) else "(no line)"
            line2 = lines2[i].strip() if i < len(lines2) else "(no line)"

            # Highlight differences
            if line1 != line2:
                print(f"Line {i + 1}:")
                print(f"  File 1: {line1}")
                print(f"  File 2: {line2}")
                print()

        # Word-by-word comparison for differing lines
        print("\nWord-by-Word Comparison (for differing lines):\n")
        differ = Differ()
        for i in range(max_lines):
            line1 = lines1[i].strip() if i < len(lines1) else ""
            line2 = lines2[i].strip() if i < len(lines2) else ""
            if line1 != line2:
                print(f"Line {i + 1}:")
                diff = list(differ.compare(line1.split(), line2.split()))
                for word in diff:
                    if word.startswith('- '):
                        print(f"  Missing in File 2: {word[2:]}")
                    elif word.startswith('+ '):
                        print(f"  Missing in File 1: {word[2:]}")
                    elif word.startswith('? '):
                        pass  # Ignore alignment markers
                    else:
```

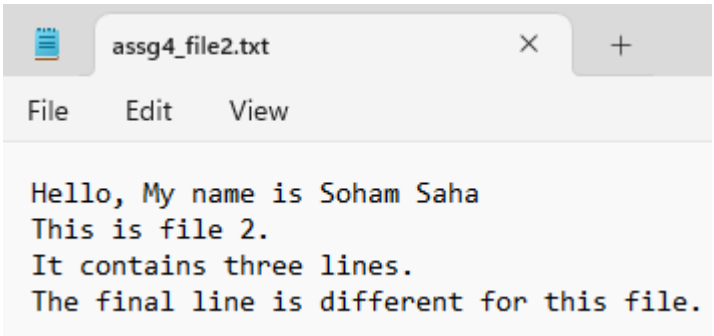
```
                print(f"  Common: {word[2:]}")
            print()
        print("Comparison complete.")
    except FileNotFoundError as e:
        print(f"Error: {e}")

# File names to compare
file1 = 'assg4_file1.txt'
file2 = 'assg4_file2.txt'

# Compare the files
compare_files(file1, file2)
```

Output :

```
$ python "d:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\tempCodeRunnerFile.py"
Comparing 'assg4_file1.txt' and 'assg4_file2.txt':

Line-by-Line Comparison:

Line 2:
  File 1: This is file 1.
  File 2: This is file 2.

Line 4:
  File 1: End of file 1.
  File 2: The final line is different for this file.
```




```
Word-by-Word Comparison (for differing lines):

Line 2:
  Common: This
  Common: is
  Common: file
  Missing in File 2: 1.
  Missing in File 1: 2.

Line 4:
  Missing in File 2: End
  Missing in File 2: of
  Missing in File 1: The
  Missing in File 1: final
  Missing in File 1: line
  Missing in File 1: is
  Missing in File 1: different
  Missing in File 1: for
  Missing in File 1: this
  Missing in File 2: file
  Missing in File 1: file.
  Missing in File 2: 1.

Comparison complete.
```



Assignmnet-5 : Log File Analyzer

- **Problem**: Write a program to analyze a log file and extract useful information such as error counts and timestamps.
- **Example**: Given a server log file, count the number of error occurrences and list the corresponding timestamps.
- **Objective**: Parse and process log files to extract patterns like error messages, IP addresses, or other critical information.

Program Code :

```
import re
from collections import Counter

def analyze_log_file(log_file):
    try:
        # Open the log file in read mode
        with open(log_file, 'r') as file:
            lines = file.readlines()

        # Initialize variables
        error_count = Counter()  # To count occurrences of each error
        error_details = []  # To store timestamps and error messages

        # Regex patterns for timestamp and error messages
        timestamp_pattern = r'\[(.*?)\]'  # Matches timestamps like [2025-01-25 10:00:00]
        error_pattern = r'(ERROR|WARNING|CRITICAL): (.+)'  # Matches error messages with their
severity

        # Process each line in the log file
        for line in lines:
            timestamp_match = re.search(timestamp_pattern, line)
            error_match = re.search(error_pattern, line)

            if error_match:
                # Extract timestamp and error message
                timestamp = timestamp_match.group(1) if timestamp_match else "No timestamp"
                error_message = error_match.group(2)
                severity = error_match.group(1)

                # Increment error count and store details
                error_count[error_message] += 1
                error_details.append((timestamp, severity, error_message))

        # Display results
        print("Error Counts:")
        for error, count in error_count.items():
            print(f"{error}: {count} occurrences")

        print("\nError Details:")
        for timestamp, severity, message in error_details:
            print(f"[{timestamp}] {severity}: {message}")
```

```
    except FileNotFoundError:
        print(f"Error: File '{log_file}' not found.")

# Specify the log file name
log_file = 'server.log'

# Analyze the log file
analyze_log_file(log_file)
```

Output :

server.log

```
[2025-01-25 10:00:00] INFO: Server started successfully.
[2025-01-25 10:05:23] ERROR: Database connection failed.
[2025-01-25 10:10:45] WARNING: High memory usage detected.
[2025-01-25 10:15:30] ERROR: Database connection failed.
[2025-01-25 10:20:15] CRITICAL: Disk space exhausted.
[2025-01-25 10:25:50] ERROR: Failed to authenticate user.
[2025-01-25 10:30:00] INFO: Server shut down.
```

$ python "d:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\tempCodeRunnerFile.py"
Error Counts:
Database connection failed.: 2 occurrences
High memory usage detected.: 1 occurrences
Disk space exhausted.: 1 occurrences
Failed to authenticate user.: 1 occurrences

Error Details:
[2025-01-25 10:05:23] ERROR: Database connection failed.
[2025-01-25 10:10:45] WARNING: High memory usage detected.
[2025-01-25 10:15:30] ERROR: Database connection failed.
[2025-01-25 10:20:15] CRITICAL: Disk space exhausted.
[2025-01-25 10:25:50] ERROR: Failed to authenticate user.

Assignmnet-6 : CSV File Reader and Writer

- **Problem**: Write a program to read data from a CSV file, modify the data, and write it to another CSV file.
- **Example**: Given a CSV file containing user information, modify one of the fields (e.g., change email addresses) and save the changes to a new CSV file.
- **Objective**: Work with CSV files, reading and writing data using Python's `csv` module.

Program Code:

```
import csv

def modify_csv(input_file, output_file, field_to_modify, modify_function):
    try:
        # Read the data from the input CSV file
        with open(input_file, 'r', newline='') as infile:
            reader = csv.DictReader(infile)
            rows = list(reader)  # Store all rows in a list
            fieldnames = reader.fieldnames  # Get field names (headers)

        # Check if the field to modify exists
        if field_to_modify not in fieldnames:
            print(f"Error: Field '{field_to_modify}' not found in the CSV file.")
            return

        # Modify the specified field using the provided function
        for row in rows:
            row[field_to_modify] = modify_function(row[field_to_modify])

        # Write the modified data to the output CSV file
        with open(output_file, 'w', newline='') as outfile:
            writer = csv.DictWriter(outfile, fieldnames=fieldnames)
            writer.writeheader()  # Write the header
            writer.writerows(rows)  # Write the modified rows

        print(f"CSV file modified successfully and saved as '{output_file}'.")

    except FileNotFoundError:
        print(f"Error: File '{input_file}' not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

# Example modify function to change email addresses
def modify_email(email):
    if '@' in email:
        username, domain = email.split('@')
        return f"{username}@newdomain.com"  # Change domain
    return email

# File paths
input_file = 'users.csv'  # Input CSV file
output_file = 'updated_users.csv'  # Output CSV file

# Field to modify
field_to_modify = 'Email'
```

```
# Modify the CSV file
modify_csv(input_file, output_file, field_to_modify, modify_email)
```

Output :

user.csv

	A	B	C	D
1	ID	Name	Email	Age
2	1	Alice	alice@exa	25
3	2	Bob	bob@exai	30
4	3	Charlie	charlie@e	35

updated_user.csv

	A	B	C	D
1	ID	Name	Email	Age
2	1	Alice	alice@newdomain.com	25
3	2	Bob	bob@newdomain.com	30
4	3	Charlie	charlie@newdomain.com	35

```
$ python "d:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\assg6.py"
CSV file modified successfully and saved as 'updated_users.csv'.
```

Assignmnet-7 : Parsing JSON Files

- **Problem**: Write a program to read a JSON file, parse the data, and print it in a human-readable format.
- **Example**: Given a JSON file with nested structures, extract specific pieces of data (e.g., user details) and display them.
- **Objective**: Read and parse JSON files using Python's `json` module and extract specific data fields.

Program Code :

```
import json

def parse_json_file(json_file, fields_to_extract=None):
    try:
        # Open and read the JSON file
        with open(json_file, 'r') as file:
            data = json.load(file)  # Parse JSON into a Python dictionary or list

        # Print the JSON data in a human-readable format
        print("Full JSON Data (Formatted):")
        print(json.dumps(data, indent=4))  # Pretty print JSON

        # Extract and print specific fields if requested
        if fields_to_extract:
            print("\nExtracted Data:")
            for field in fields_to_extract:
                extracted_value = extract_field(data, field)
                print(f"{field}: {extracted_value if extracted_value else 'Field not found'}")
        else:
            print("\nNo specific fields to extract.")
    except FileNotFoundError:
        print(f"Error: File '{json_file}' not found.")
    except json.JSONDecodeError:
        print("Error: Invalid JSON format.")
    except Exception as e:
        print(f"An error occurred: {e}")

def extract_field(data, field):
    # Recursively extract data from nested JSON structures
    keys = field.split('.')  # Support nested fields with dot notation
    value = data
    for key in keys:
        if isinstance(value, dict) and key in value:
            value = value[key]
        else:
            return None  # Field not found
    return value

# File path
json_file = 'data.json'

# Fields to extract (supports dot notation for nested fields)
fields_to_extract = ['user.name', 'user.email', 'user.details.age']
```

```
# Parse and process the JSON file
parse_json_file(json_file, fields_to_extract)
```

Output :

```
$ python "d:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\assg7.py"
Full JSON Data (Formatted):
{
    "user": {
        "name": "Alice",
        "email": "alice@example.com",
        "details": {
            "age": 25,
            "location": "New York"
        }
    },
    "settings": {
        "theme": "dark",
        "notifications": true
    }
}

Extracted Data:
user.name: Alice
user.email: alice@example.com
user.details.age: 25
```

Assignmnet-8 : Binary File Reader and Writer

- **Problem**: Write a program to read binary data from a file, manipulate it, and write the modified data back to a new file.
- **Example**: Read a binary image file, modify some of the pixel values, and save it as a new image.
- **Objective**: Work with binary files using Python's file I/O methods like `rb` and `wb` modes.

Assignmnet-9 : File Encryption and Decryption

- **Problem**: Implement a program that encrypts the content of a text file and decrypts it back to its original form.
- **Example**: Given a text file, use a simple encryption algorithm (like Caesar Cipher or XOR) to encrypt the file content and then decrypt it.
- **Objective**: Read from a file, encrypt the data, write it to a new file, and implement the reverse process to decrypt.

Assignmnet-10 : File Compression and Decompression

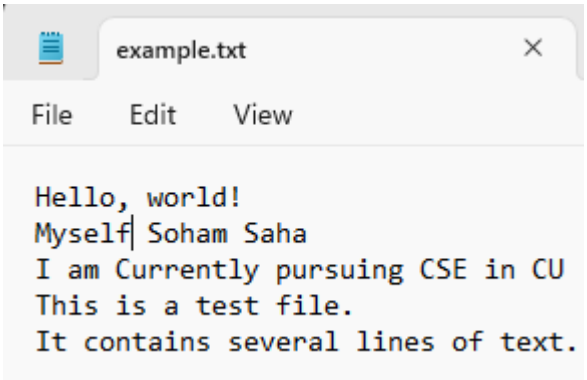
- **Problem**: Write a program to compress a file using gzip or zip and decompress it back to its original form.
- **Example**: Compress a text file `data.txt` into `data.zip` and then decompress it.
- **Objective**: Work with file compression and decompression using Python's `gzip` or `zipfile` module.

Assignmnet-11 : Counting Lines, Words, and Characters in a File

- **Problem**: Write a program to count the number of lines, words, and characters in a text file.
- **Example**: Given a text file, calculate the number of lines, words, and characters, and print the results.
- **Objective**: Read the file and perform basic text analysis to count lines, words, and characters.

Program Code :

```
def count_file_content(file_path):
    try:
        # Open the file in read mode
        with open(file_path, 'r') as file:
            lines = file.readlines()  # Read all lines
            lines = [line for line in lines if line.strip()] # Remove empty lines
        # Count the number of lines
        line_count = len(lines)
        # Count the number of words and characters
        word_count = 0
        char_count = 0
        for line in lines:
            words = line.split()  # Split the line into words
            word_count += len(words)  # Add word count of this line
            char_count += len(line)  # Add character count of this line
        # Print the results
        print(f"File: {file_path}")
        print(f"Number of lines: {line_count}")
        print(f"Number of words: {word_count}")
        print(f"Number of characters: {char_count}")

    except FileNotFoundError:
        print(f"Error: File '{file_path}' not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

# File path
file_path = 'example.txt'  # Replace with your text file's path

# Count lines, words, and characters in the file
count_file_content(file_path)
```



Output :

```
$ python "d:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\tempCodeRunnerFile.py"
File: example.txt
Number of lines: 5
Number of words: 23
Number of characters: 122
```

Assignmnet-12 : Finding Duplicate Files in a Directory

- **Problem**: Write a program to scan a directory and identify duplicate files based on file content (not file names).
- **Example**: Compare all files in a given directory and identify any files that have the same content.
- **Objective**: Use hashing or byte-by-byte comparison to identify duplicate files in a directory.

Program Code :

```
import os
import hashlib

def hash_file(file_path):
    """Generate a hash for the file content using SHA-256."""
    hash_obj = hashlib.sha256()
    try:
        with open(file_path, 'rb') as file:
            while chunk := file.read(4096):  # Read the file in chunks to handle large files
                hash_obj.update(chunk)
        return hash_obj.hexdigest()
    except Exception as e:
        print(f"Error hashing file {file_path}: {e}")
        return None

def find_duplicate_files(directory):
    """Find duplicate files in the given directory based on content."""
    file_hashes = {}  # Dictionary to store hash: file paths
    duplicates = []  # List to store duplicate file pairs

    # Walk through the directory and its subdirectories
    for root, _, files in os.walk(directory):
        for file in files:
            file_path = os.path.join(root, file)
            file_hash = hash_file(file_path)

            if file_hash:
                if file_hash in file_hashes:
                    duplicates.append((file_hashes[file_hash], file_path))
                else:
                    file_hashes[file_hash] = file_path

    return duplicates

def display_duplicates(duplicates):
    """Display duplicate file information."""
    if duplicates:
        print("Duplicate files found:")
        for original, duplicate in duplicates:
            print(f"Original: {original}")
            print(f"Duplicate: {duplicate}\n")
    else:
        print("No duplicate files found.")
```

```
# Directory to scan
directory = input("Enter the directory to scan for duplicates: ")

# Find and display duplicate files
duplicates = find_duplicate_files(directory)
display_duplicates(duplicates)
```

Output :

```
$ python "d:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\assg12.py"
Enter the directory to scan for duplicates: D:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\example_dir
Duplicate files found:
Original: D:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\example_dir\old_duplicate_of_file1.txt
Duplicate: D:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\example_dir\old_file1.txt

Original: D:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\example_dir\old_file2.txt
Duplicate: D:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\example_dir\sub_dir\2nd_duplicate.txt
```

Assignmnet-13 : Directory Walker (Recursive File Listing)

- **Problem**: Write a Python program to recursively list all files and directories in a given directory.
- **Example**: Given a directory, output a tree-like structure of all files and subdirectories within it.
- **Objective**: Use `os` or `os.path` to traverse directories and print their structure.

Program Code :

```
import os
def list_directory(directory, indent=0):
    """Recursively list all files and directories in a given directory."""
    try:
        # Get all items in the directory
        items = os.listdir(directory)
    except PermissionError:
        print(" " * indent + f"[Permission Denied]: {directory}")
        return
    except FileNotFoundError:
        print(f"Error: Directory '{directory}' not found.")
        return
    # Sort items to display directories first
    items.sort(key=lambda x: (not os.path.isdir(os.path.join(directory, x)), x.lower()))
    for item in items:
        item_path = os.path.join(directory, item)
        # Print the item with indentation
        if os.path.isdir(item_path):
            print(" " * indent + f"[DIR] {item}")
            # Recursively list subdirectories
            list_directory(item_path, indent + 4)
        else:
            print(" " * indent + f"[FILE] {item}")
# Get the directory path from the user
directory = input("Enter the directory to list: ")
# Start listing
print(f"Listing files and directories in: {directory}")
list_directory(directory)
```

Output :

```
$ python "d:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\assg13.py"
Enter the directory to list: D:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\example_dir
Listing files and directories in: D:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\example_dir
[DIR] sub_dir
    [FILE] 2nd_duplicate.txt
[FILE] old_duplicate_of_file1.txt
[FILE] old_file1.txt
[FILE] old_file2.txt
```

Assignmnet-14 : Renaming Files in a Directory

- **Problem**: Write a program to rename all files in a directory according to a specific pattern.
- **Example**: Rename all `.txt` files in a directory by adding a prefix like `old_` to their names.
- **Objective**: Use Python's `os` or `os.rename()` function to batch rename files based on specific criteria.

Program Code :

```
import os

def rename_files_in_directory(directory, prefix='', suffix='', extension_filter=None):
    try:
        # Get a list of all files in the directory
        files = os.listdir(directory)

        for file in files:
            file_path = os.path.join(directory, file)

            # Skip directories
            if not os.path.isfile(file_path):
                continue

            # Filter by file extension if specified
            if extension_filter and not file.endswith(extension_filter):
                continue

            # Extract file name and extension
            file_name, file_ext = os.path.splitext(file)

            # Construct the new file name
            new_file_name = f"{prefix}{file_name}{suffix}{file_ext}"
            new_file_path = os.path.join(directory, new_file_name)

            # Rename the file
            os.rename(file_path, new_file_path)
            print(f"Renamed: {file} -> {new_file_name}")

        print("\nRenaming complete!")

    except FileNotFoundError:
        print(f"Error: Directory '{directory}' not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

# Get user inputs
directory = input("Enter the directory path: ")
prefix = input("Enter a prefix to add (leave blank if not needed): ")
suffix = input("Enter a suffix to add (leave blank if not needed): ")
extension_filter = input("Enter the file extension to filter by (e.g., .txt, leave blank for all files): ")

# Rename files in the directory
rename_files_in_directory(directory, prefix, suffix, extension_filter)
```

Output :

```
$ python "d:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\assg14.py"
Enter the directory path: D:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\example_dir
Enter a prefix to add (leave blank if not needed): New_
Enter a suffix to add (leave blank if not needed):
Enter the file extension to filter by (e.g., .txt, leave blank for all files):
Renamed: old_duplicate_of_file1.txt -> New_old_duplicate_of_file1.txt
Renamed: old_file1.txt -> New_old_file1.txt
Renamed: old_file2.txt -> New_old_file2.txt

Renaming complete!
```

Assignmnet-15 : File Metadata Extraction

- **Problem**: Write a program to extract and display metadata (such as file size, creation date, and modification date) of files in a directory.
- **Example**: Given a directory of files, print each file's size, creation date, and last modification date.
- **Objective**: Use Python's os or os.stat() methods to retrieve and display file metadata.

Program Code :

```
import os
import time

def get_file_metadata(directory):
    try:
        # Check if the directory exists
        if not os.path.isdir(directory):
            print("Error: The specified directory does not exist.")
            return

        print(f"\nMetadata for files in directory: {directory}\n")
        print(f"{'File Name':<30} {'Size (Bytes)':<15} {'Creation Date':<25} {'Modification Date':<25}")
        print("-" * 100)

        # Iterate over files in the directory
        for file_name in os.listdir(directory):
            file_path = os.path.join(directory, file_name)

            # Skip subdirectories
            if os.path.isfile(file_path):
                # Get file metadata
                file_stat = os.stat(file_path)
                file_size = file_stat.st_size
                creation_time = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(file_stat.st_ctime))
                modification_time = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(file_stat.st_mtime))

                # Print metadata
                print(f"{file_name:<30} {file_size:<15} {creation_time:<25} {modification_time:<25}")
    except Exception as e:
        print(f"An error occurred: {e}")

def main():
    # Input directory path
    directory = input("Enter the directory path: ")
    get_file_metadata(directory)

if __name__ == "__main__":
    main()
```

Output :

```
sohamsaha@LAPTOP-IEU86UEB MINGW64 /d/sohamsaha/Documents/sohamsaha/B.Tech CSE/CSE SEM-3/Programing with Python and C
$ python "d:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\assg15.py"
Enter the directory path: D:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\example_dir

Metadata for files in directory: D:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\example_dir

File Name                      Size (Bytes)    Creation Date            Modification Date
New_old_file2.txt              15              2025-01-26 01:10:03      2025-01-26 01:10:20
```

Assignmnet-16 : Log File Rotation

- **Problem**: Implement a program that simulates **log file rotation** by renaming old log files and creating a new log file when the current one reaches a certain size.
- **Example**: When the current log file exceeds 1MB, rename it and start a new log file.
- **Objective**: Work with file sizes, renaming, and writing to new files to implement file rotation behavior.

Assignmnet-17 : Finding and Replacing Text in Files

- **Problem**: Write a program that reads a file, searches for a specific text pattern, and replaces it with another text.
- **Example**: Search for the word “error” in a log file and replace it with “warning.”
- **Objective**: Use Python’s file reading and writing capabilities to perform search-and-replace operations in text files.

Program code :

```
def find_and_replace(file_path, search_text, replace_text, output_file=None):
    try:
        # Open and read the original file
        with open(file_path, 'r') as file:
            content = file.read()
        # Perform the search and replace operation
        modified_content = content.replace(search_text, replace_text)
        # Determine the file to save the changes
        save_path = output_file if output_file else file_path
        # Write the modified content to the output file
        with open(save_path, 'w') as file:
            file.write(modified_content)

        print(f"'{search_text}' has been replaced with '{replace_text}' in '{save_path}'.")

    except FileNotFoundError:
        print(f"Error: File '{file_path}' not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

# User inputs
file_path = input("Enter the path of the file to modify: ")
search_text = input("Enter the text to search for: ")
replace_text = input("Enter the text to replace it with: ")
output_file = input("Enter the output file path (leave blank to overwrite the original file): ")

# Call the function
find_and_replace(file_path, search_text, replace_text, output_file if output_file.strip() else None)
```

Output :

```
$ python "d:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\tempCodeRunnerFile.py"
Enter the path of the file to modify: D:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\example_dir
Enter the text to search for: The
Enter the text to replace it with: the
Enter the output file path (leave blank to overwrite the original file):

An error occurred: [Errno 13] Permission denied: 'D:\\sohamsaha\\Documents\\sohamsaha\\B.Tech CSE\\CSE SEM-3\\Programing with Python and C\\example_dir'
```

Assignmnet-18 : Generating and Writing Large Data Files

- **Problem**: Write a program to generate large data files, such as files containing random numbers, text, or CSV data.
- **Example**: Generate a file with 10 million random integers and write it to a file.
- **Objective**: Use Python to generate large datasets and write them efficiently to disk.

Program code :

```
def generate_large_data_file(filename, rows, data_type="numbers", delimiter=","):
    try:
        with open(filename, "w", newline="") as file:
            if data_type == "numbers":
                for _ in range(rows):
                    number = random.randint(1, 10**6)
                    file.write(f"{number}\n")

            elif data_type == "text":
                words = ["apple", "banana", "cherry", "date", "elderberry"]
                for _ in range(rows):
                    sentence = " ".join(random.choices(words, k=10))
                    file.write(f"{sentence}\n")

            elif data_type == "csv":
                writer = csv.writer(file, delimiter=delimiter)
                headers = ["ID", "Name", "Score"]
                writer.writerow(headers)
                for i in range(1, rows + 1):
                    row = [i, f"Name_{i}", random.randint(0, 100)]
                    writer.writerow(row)

            else:
                print("Invalid data type. Please choose 'numbers', 'text', or 'csv'.")
                return

        print(f"File '{filename}' with {rows} rows of {data_type} data created successfully.")

    except Exception as e:
        print(f"An error occurred: {e}")

# User inputs
filename = input("Enter the output file name (e.g., data.txt or data.csv): ")
rows = int(input("Enter the number of rows to generate: "))
data_type = input("Enter the type of data to generate ('numbers', 'text', 'csv'): ").lower()
delimiter = input("Enter a delimiter for CSV files (default is ','): ") or ","

# Generate the file
generate_large_data_file(filename, rows, data_type, delimiter)
```

Assignmnet-19 : Checksum Calculation for Files

- **Problem**: Write a program to calculate a file's checksum (e.g., MD5 or SHA256) to ensure its integrity.
- **Example**: Compute the MD5 hash of a file and verify its integrity by comparing it to a known hash.
- **Objective**: Use Python's `hashlib` module to compute and verify file checksums.

Program code :

Output :

Assignmnet-20 : File Permissions Checker

- **Problem**: Write a Python program that checks the permissions of files in a directory and identifies files with insecure permissions.
- **Example**: Scan a directory and identify files that are world-writable or have other risky permissions.
- **Objective**: Use Python's `os` and `stat` modules to check file permissions and highlight any potential security risks.

Program code :

```
import os
import stat

def check_file_permissions(directory):
    try:
        print(f"Scanning directory: {directory}\n")
        print(f"{'File Name':<40} {'Permissions':<15} {'Insecure?':<10}")
        print("-" * 65)
        # Walk through files in the directory
        for root, _, files in os.walk(directory):
            for file in files:
                file_path = os.path.join(root, file)
                # Get file permissions
                file_stat = os.stat(file_path)
                permissions = stat.filemode(file_stat.st_mode)
                # Check for insecure permissions (world-writable)
                is_insecure = False
                if (file_stat.st_mode & stat.S_IWOTH) != 0:  # World-writable
                    is_insecure = True
                # Check for other risky permissions
                insecure_label = "Yes" if is_insecure else "No"

                print(f"{file:<40} {permissions:<15} {insecure_label:<10}")

        print("\nScan complete.")

    except FileNotFoundError:
        print(f"Error: Directory '{directory}' not found.")
    except PermissionError:
        print("Error: Permission denied while accessing files.")
    except Exception as e:
        print(f"An error occurred: {e}")
```

```
# Get user input for the directory to scan
directory = input("Enter the directory to scan for file permissions: ")

# Perform the file permissions check
check_file_permissions(directory)
```

Output :

```
$ python "d:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\tempCodeRunnerFile.py"

Enter the directory to scan for file permissions: D:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\example_dir

Scanning directory: D:\sohamsaha\Documents\sohamsaha\B.Tech CSE\CSE SEM-3\Programing with Python and C\example_dir


File Name                                Permissions     Insecure?

--------------------------------------------------------------------------------

New_old_duplicate_of_file1.txt          -rw-rw-rw-      Yes

New_old_file1.txt                       -rw-rw-rw-      Yes

New_old_file2.txt                       -rw-rw-rw-      Yes

2nd_duplicate.txt                       -rw-rw-rw-      Yes


Scan complete.
```