1 Read and Print the Contents of a Text File

Method 1: Using `open()` and `read()`

In this method, we'll open the file in read mode (`'r'`). Then, we'll read the entire content using the `read()` function. We'll count the lines, words, and characters within the content.

```
def count_lines_words_chars(file_path):
    try:
        with open(file_path, 'r') as file:
            file_content = file.read()
            num_lines = len(file_content.split('\n'))
            num_words = len(file_content.split())
            num_chars = len(file_content)
            return num_lines, num_words, num_chars
    except FileNotFoundError:
        return None

# Example usage:
file_path = 'sample.txt'
result = count_lines_words_chars(file_path)
if result:
    num_lines, num_words, num_chars = result
    print(f"Number of lines: {num_lines}")
    print(f"Number of words: {num_words}")
    print(f"Number of characters: {num_chars}")
else:
    print(f"File '{file_path}' not found.")
```

Method 2: Using `readlines()`

In this alternative method, we'll read the file line by line using `readlines()`. We'll count the lines and words within each line.

```
def count_lines_words(file_path):
    try:
        with open(file_path, 'r') as file:
            num_lines = 0
            num_words = 0
            for line in file:
                words = line.split()
                num_lines += 1
                num_words += len(words)
            return num_lines, num_words
    except FileNotFoundError:
        return None

# Example usage:
file_path = 'sample.txt'
result = count_lines_words(file_path)
if result:
    num_lines, num_words = result
    print(f"Number of lines: {num_lines}")
    print(f"Number of words: {num_words}")
else:
    print(f"File '{file_path}' not found.")
```

2 Count the Number of Lines, Words, and Characters in a Text File

Method 1: Using open() and read()

```
def count_lines_words_chars(file_path):
    try:
        with open(file_path, 'r') as file:
            file_content = file.read()
            num_lines = len(file_content.split('\n'))
            num_words = len(file_content.split())
            num_chars = len(file_content)
            return num_lines, num_words, num_chars
    except FileNotFoundError:
        return None

# Example usage:
file_path = 'sample.txt'
result = count_lines_words_chars(file_path)
if result:
    num_lines, num_words, num_chars = result
    print(f"Number of lines: {num_lines}")
    print(f"Number of words: {num_words}")
    print(f"Number of characters: {num_chars}")
else:
    print(f"File '{file_path}' not found.")
```

Method 2: Using readlines()
In this alternative method, we'll read the file line by line using readlines(). We'll count the lines and words within each line. (this is the method I have done)

3 . Search for a String in a Text File

Method 1: Using open() and read()
In this method, we'll open the file in read mode ('r'). Then, we'll read the entire content using the read() function. We'll count the lines, words, and characters within the content.

```
def search_string_in_file(file_path, target_string):
    try:
        with open(file_path, 'r') as file:
            file_content = file.read()
            if target_string in file_content:
                return f"'{target_string}' exists in the file."
            else:
                return f"'{target_string}' does not exist in the file."
    except FileNotFoundError:
        return f"File '{file_path}' not found."

# Example usage:
file_path = 'sample.txt'
target = 'Line 3'
result = search_string_in_file(file_path, target)
print(result)
```

Method 2: Using readlines()
In this alternative method, we'll read the file line by line using readlines(). We'll check if the target string exists in any of the lines.

```
def search_string_by_line(file_path, target_string):
    try:
        with open(file_path, 'r') as file:
            for line in file:
                if target_string in line:
                    return f"'{target_string}' found in the file."
            return f"'{target_string}' does not exist in the file."
    except FileNotFoundError:
        return f"File '{file_path}' not found."

# Example usage:
file_path = 'sample.txt'
target = 'Line 8'
result = search_string_by_line(file_path, target)
print(result)
```

Method 3: Using enumerate()

We can also find the string using the enumerate() function. It checks whether the target string is present in the file.

```
def search_string_with_enumerate(file_path, target_string):
    try:
        with open(file_path, 'r') as file:
            for index, line in enumerate(file):
                if target_string in line:
                    return f"'{target_string}' found in line {index + 1}."
            return f"'{target_string}' does not exist in the file."
    except FileNotFoundError:
        return f"File '{file_path}' not found."

# Example usage:
file_path = 'sample.txt'
target = 'Line 3y'
result = search_string_with_enumerate(file_path, target)
print(result)
```

4. Merge Multiple Text Files into One

Using file handling with open() and write (the method I have done)

Using os module:

```
import os
input_files = ['file1.txt', 'file2.txt', 'file3.txt']
output_file = 'merged_file.txt'

with open(output_file, 'w') as outfile:
    for file_name in input_files:
        with open(file_name, 'r') as infile:
            outfile.write(infile.read())
            outfile.write('\n')
```

Using shutil module:

```
import shutil
input_files = ['file1.txt', 'file2.txt', 'file3.txt']
output_file = 'merged_file.txt'

with open(output_file, 'wb') as outfile:
    for file_name in input_files:
        with open(file_name, 'rb') as infile:
            shutil.copyfileobj(infile, outfile)
            outfile.write(b'\n')
```

5.Implement a program that reads a text file and counts the occurrences of each word, ignoring case sensitivity.

Method 1: Using file handling with dictionary (this is the method that i done)

Method 2: Using file handling with defaultdict

```
from collections import defaultdict
def count_word_occurrences(file_name):
    word_count = defaultdict(int)
    with open(file_name, 'r') as file:
        content = file.read()
        words = content.lower().split()
        for word in words:
            word_count[word] += 1

    return word_count

file_name = input("Enter the text file: ")
word_count = count_word_occurrences(file_name)

for word, count in word_count.items():
    print(f"'{word}': {count}")
```

Method 3 : Using file handling with Counter

```
from collections import Counter

def count_word_occurrences(file_name):
    word_count = Counter()


    with open(file_name, 'r') as file:
        content = file.read()
        words = content.lower().split()
        word_count.update(words)

    return word_count

file_name = input("Enter the text file: ")
word_count = count_word_occurrences(file_name)
for word, count in word_count.items():
    print(f"'{word}': {count}")
```

6. Write a Python function that takes a list of strings as input and returns a new list with the strings sorted in descending order of their lengths.

Method which wrote is the only method

7. Write a function that takes a list of numbers as input and returns the second-largest number. also show diffrent methods

Method 1: Using sorting

```
def second_largest(numbers):
  sorted_numbers = sorted(numbers)
  return sorted_numbers[-2]

# Example usage:
numbers = [10, 20, 30, 40, 50]
print("Second largest number:", second_largest(numbers))
```

Method 2: Using max() and remove()

I have used this method

Method 3: Using heapq

```
import heapq
def second_largest(numbers):
  return heapq.nlargest(2, set(numbers))[-1]
# Example usage:
numbers = [10, 20, 30, 40, 50]
print("Second largest number:", second_largest(numbers))
```

Method 4: Using loop with max() and remove()

```
def second_largest(numbers):
  max_num = max(numbers)
  second_max = float('-inf')
  for num in numbers:
    if num != max_num and num > second_max:
      second_max = num
  return second_max
# Example usage:
numbers = [10, 20, 30, 40, 50]
print("Second largest number:", second_largest(numbers))
```

Method 5: Using sorting with set()

```
def second_largest(numbers):
  unique_numbers = list(set(numbers))
  unique_numbers.sort()
  return unique_numbers[-2]

# Example usage:
numbers = [10, 20, 30, 40, 50]
print("Second largest number:", second_largest(numbers))
```

8. Write a Python program that takes a list of integers as input and returns a new list with only the numbers that are prime.

Method 1: Using a function to check for prime numbers

```
def is_prime(n):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True

def filter_primes(numbers):
    return [num for num in numbers if is_prime(num)]

# Example usage:
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
prime_numbers = filter_primes(numbers)
print("Prime numbers:", prime_numbers)
```

Method 2: Using a function with a loop(method I use)

Method 3: Using the Sieve of Eratosthenes algorithm

```
def sieve_of_eratosthenes(n):
    primes = [True] * (n + 1)
    primes[0] = primes[1] = False
    p = 2
    while p * p <= n:
        if primes[p]:
            for i in range(p * p, n + 1, p):
                primes[i] = False
        p += 1
    return [i for i in range(n + 1) if primes[i]]

def filter_primes(numbers):
    max_num = max(numbers)
    primes = sieve_of_eratosthenes(max_num)
    return [num for num in numbers if num in primes]

# Example usage:
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
prime_numbers = filter_primes(numbers)
print("Prime numbers:", prime_numbers)
```

9 Write a Python function that takes a list of integers as input and returns a new list with only the numbers that are perfect squares.

Method 1: Using a function to check perfect squares

```
import math

def is_perfect_square(num):
    sqrt = math.isqrt(num)
    return sqrt * sqrt == num

def filter_perfect_squares(numbers):
    return [num for num in numbers if is_perfect_square(num)]

# Example usage:
input_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
perfect_squares = filter_perfect_squares(input_list)
print("Perfect square numbers:", perfect_squares)
```

Method 2: Using a loop to check perfect squares (I use this method)

Method 3: Using math.isqrt() function

```
import math

def filter_perfect_squares(numbers):
    return [num for num in numbers if math.isqrt(num) ** 2 == num]

# Example usage:
input_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
perfect_squares = filter_perfect_squares(input_list)
print("Perfect square numbers:", perfect_squares)
```

Method 4: Using a generator expression

```
def filter_perfect_squares(numbers):
    return [num for num in numbers if any(num == i * i for i in range(int(num ** 0.5) + 1))]

# Example usage:
```

```
input_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
perfect_squares = filter_perfect_squares(input_list)
print("Perfect square numbers:", perfect_squares)
```

Method 5: Using numpy library

```
import numpy as np

def filter_perfect_squares(numbers):
    return [num for num in numbers if np.sqrt(num) % 1 == 0]

# Example usage:
input_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
perfect_squares = filter_perfect_squares(input_list)
print("Perfect square numbers:", perfect_squares)
```

10. Write a Python function that takes a list of numbers as input and returns the sum of all the numbers divisible by 3 or 5.

Method 1 Using a Loop
I used this method.

Method 2: Using list comprehension with sum()

```
def sum_divisible_by_3_or_5(numbers):
    return sum(num for num in numbers if num % 3 == 0 or num % 5 == 0)

# Example usage:
input_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
result = sum_divisible_by_3_or_5(input_list)
print("Sum of numbers divisible by 3 or 5:", result)
```

Method 3: Using filter() and sum()

```
def sum_divisible_by_3_or_5(numbers):
    divisible_numbers = filter(lambda x: x % 3 == 0 or x % 5 == 0, numbers)
    return sum(divisible_numbers)

# Example usage:
input_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
result = sum_divisible_by_3_or_5(input_list)
print("Sum of numbers divisible by 3 or 5:", result)
```

Method 4: Using numpy library

```
import numpy as np

def sum_divisible_by_3_or_5(numbers):
    divisible_numbers = [num for num in numbers if num % 3 == 0 or num % 5 == 0]
    return np.sum(divisible_numbers)

# Example usage:
input_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
result = sum_divisible_by_3_or_5(input_list)
print("Sum of numbers divisible by 3 or 5:", result)
```

11. Write a Python function called calculate_discounted_price that takes the original price of an item and a discount percentage as input. The function should return the discounted price after applying the discount. Ensure that the function handles the case where the discount percentage is negative and raises a custom exception called InvalidDiscountError with an appropriate error message.

Method 1: Using if-else statement to check for negative discount percentage
Method I used.

Method 2: Using assert statement to validate the discount percentage

```
class InvalidDiscountError(Exception):
    pass

def calculate_discounted_price(original_price, discount_percentage):
    assert discount_percentage >= 0, "Discount percentage cannot be negative"
    discounted_price = original_price - (original_price * discount_percentage / 100)
    return discounted_price

# Example usage:
try:
    original_price = float(input("Enter the original price: "))
    discount_percentage = float(input("Enter the discount percentage: "))
    discounted_price = calculate_discounted_price(original_price, discount_percentage)
    print("Discounted price:", discounted_price)
except AssertionError as e:
    print("Error:", e)
```

Method 3: Using a custom if-condition to check for negative discount percentage

python
Copy code
```
class InvalidDiscountError(Exception):
    pass

def calculate_discounted_price(original_price, discount_percentage):
    if discount_percentage < 0:
        raise InvalidDiscountError("Discount percentage cannot be negative")
    discounted_price = original_price - (original_price * discount_percentage / 100)
    return discounted_price

# Example usage:
try:
    original_price = float(input("Enter the original price: "))
    discount_percentage = float(input("Enter the discount percentage: "))
    discounted_price = calculate_discounted_price(original_price, discount_percentage)
    print("Discounted price:", discounted_price)
except InvalidDiscountError as e:
    print("Error:", e)
```

12. Write a function that takes a sentence as input and returns a new sentence with the words reversed, while keeping the order of the words in the sentence.

Method 1: Using split() and join()

I use this method

Method 2: Using split() and reversed()

```
def reverse_words(sentence):
  words = sentence.split()
  reversed_words = [word[::-1] for word in reversed(words)]
  return ' '.join(reversed_words)

# Example usage:
sentence = "Hello world, how are you?"
reversed_sentence = reverse_words(sentence)
print("Reversed sentence:", reversed_sentence)
```

Method 3: Using split() and reversed() with map()

```
def reverse_words(sentence):
  words = sentence.split()
  reversed_words = list(map(lambda x: x[::-1], reversed(words)))
  return ' '.join(reversed_words)

# Example usage:
sentence = "Hello world, how are you?"
reversed_sentence = reverse_words(sentence)
print("Reversed sentence:", reversed_sentence)
```

Method 4: Using list comprehension with reversed()

```
def reverse_words(sentence):
  words = sentence.split()
  reversed_words = [''.join(reversed(word)) for word in reversed(words)]
  return ' '.join(reversed_words)

# Example usage:
sentence = "Hello world, how are you?"
reversed_sentence = reverse_words(sentence)
print("Reversed sentence:", reversed_sentence)
```

13. Implement a program that simulates a basic calculator, allowing users to perform arithmetic operations (addition, subtraction, multiplication, division) on two numbers.

Method 1: Using if-elif-else statements to handle different operations

I have used this method

Method 2: Using a dictionary to map operations to functions

```
def add(num1, num2):
  return num1 + num2

def subtract(num1, num2):
  return num1 - num2
```

```
def multiply(num1, num2):
    return num1 * num2

def divide(num1, num2):
    if num2 != 0:
        return num1 / num2
    else:
        return "Error: Division by zero!"

operations = {'+': add, '-': subtract, '*': multiply, '/': divide}

# Example usage:
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))
operation = input("Enter the operation (+, -, *, /): ")

if operation in operations:
    result = operations[operation](num1, num2)
    print("Result:", result)
else:
    print("Error: Invalid operation!")
```

Method 3: Using eval() function (Note: Using eval can be risky, especially if the input is not properly sanitized)

```
expression = input("Enter an arithmetic expression: ")

try:
    result = eval(expression)
    print("Result:", result)
except Exception as e:
    print("Error:", e)
```

14. Create a class named Notes for handling text-based file operations. Class should contain methods "write", "read" and then "append" as instance methods or class methods. (Can contain any other methods if you wish) Use a single file for saving the notes. You can set the file name as a constant somewhere in the program (Or as a class variable). write method should create the if it doesn't exist, Then it should overwrite the older contents with the user input if the user plans to overwrite the file. read method should read the whole file contents and return it. If the file doesn't exist, then it should return "No notes found" append method should take the user input value and it must add the value to the end of the file. It must not overwrite the file. Now create a program to utilize this class. The program should repeatedly ask the user for these 4 choices : 1 - Write Note (Overwrite existing). 2 - Add more Notes (Append). 3 - Read Notes. 4 – Exit

Method 1: Using instance methods

```
class Notes:
    FILE_NAME = "notes.txt"  # Class variable for file name

    def write(self):
        with open(self.FILE_NAME, 'w') as file:
            content = input("Enter your note: ")
            file.write(content)
            print("Note written successfully!")

    def read(self):
        try:
            with open(self.FILE_NAME, 'r') as file:
                return file.read()
        except FileNotFoundError:
            return "No notes found."

    def append(self):
        with open(self.FILE_NAME, 'a') as file:
            content = input("Enter additional note: ")
            file.write("\n" + content)
            print("Note appended successfully!")

def main():
    notes = Notes()
    while True:
        print("\nMenu:")
        print("1 - Write Note (Overwrite existing)")
        print("2 - Add more Notes (Append)")
        print("3 - Read Notes")
        print("4 - Exit")

        choice = input("Enter your choice: ")

        if choice == '1':
            notes.write()
        elif choice == '2':
            notes.append()
        elif choice == '3':
            print(notes.read())
        elif choice == '4':
            print("Exiting the program. Goodbye!")
            break
        else:
            print("Invalid choice. Please choose again.")

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

Method 2: Using class methods

This is the method that I used

15 Copy odd lines of one file to another file in Python

Method 1: Using a loop to iterate over lines and a counter to track odd lines

```
def copy_odd_lines(input_file, output_file):
    with open(input_file, 'r') as infile, open(output_file, 'w') as outfile:
        line_number = 1
        for line in infile:
            if line_number % 2 != 0:
                outfile.write(line)
            line_number += 1

# Example usage:
copy_odd_lines("input.txt", "output.txt")
```

Method 2: Using list slicing to filter odd lines
I have used this method

Method 3: Using enumerate() function to iterate over lines with a step

```
def copy_odd_lines(input_file, output_file):
    with open(input_file, 'r') as infile, open(output_file, 'w') as outfile:
        for index, line in enumerate(infile):
            if index % 2 == 0:
                outfile.write(line)

# Example usage:
copy_odd_lines("input.txt", "output.txt")
```

Method 4: Using itertools.islice() to filter odd lines

```
import itertools

def copy_odd_lines(input_file, output_file):
    with open(input_file, 'r') as infile, open(output_file, 'w') as outfile:
        odd_lines = itertools.islice(infile, None, None, 2)
        outfile.writelines(odd_lines)

# Example usage:
copy_odd_lines("input.txt", "output.txt")
```

16 Count the total number of uppercase characters in a file in Python

Method 1: Using a loop to iterate over each character in the file and checking for uppercase characters.

```
def count_uppercase_chars(filename):
    count = 0
    with open(filename, 'r') as file:
        for line in file:
            for char in line:
                if char.isupper():
                    count += 1
    return count

# Example usage:
filename = "example.txt"
uppercase_count = count_uppercase_chars(filename)
print("Total number of uppercase characters:", uppercase_count)
```

Method 2: Using list comprehension and sum() function to count uppercase characters.

I use this method

Method 3: Using the re module to count uppercase characters with a regular expression.

```
import re

def count_uppercase_chars(filename):
    with open(filename, 'r') as file:
        text = file.read()
        return len(re.findall(r'[A-Z]', text))

# Example usage:
filename = "example.txt"
uppercase_count = count_uppercase_chars(filename)
print("Total number of uppercase characters:", uppercase_count)
```

Method 4: Using file.readlines() and str.count() to count uppercase characters.

```
def count_uppercase_chars(filename):
    with open(filename, 'r') as file:
        lines = file.readlines()
        text = ''.join(lines)
        return text.count(c for c in text if c.isupper())

# Example usage:
filename = "example.txt"
uppercase_count = count_uppercase_chars(filename)
print("Total number of uppercase characters:", uppercase_count)
```

17. Python program to delay printing of line from a file using sleep function

Method 1: Using a loop to iterate over each line and sleep between print statements.

```
import time

def delay_print_file(filename, delay):
    try:
        with open(filename, 'r') as file:
            for line in file:
                print(line.strip())
                time.sleep(delay)
    except FileNotFoundError:
        print(f"Error: File '{filename}' not found.")

# Example usage:
filename = "example.txt"
delay = 1  # Delay in seconds
delay_print_file(filename, delay)
```

Method 2: Reading all lines into a list and using a loop to print with delay.

This method I have used

Method 3: Using the read() method to read the entire file content and splitting lines, then printing with delay.

```
import time

def delay_print_file(filename, delay):
  try:
    with open(filename, 'r') as file:
      content = file.read()
      lines = content.split('\n')
      for line in lines:
        print(line)
        time.sleep(delay)
  except FileNotFoundError:
    print(f"Error: File '{filename}' not found.")

# Example usage:
filename = "example.txt"
delay = 1  # Delay in seconds
delay_print_file(filename, delay)
```

18. Python program to for student height record for a school using Class and Objects.

Method 1: Using a class to represent students and storing heights in a list attribute.

```
class School:
  def __init__(self):
    self.student_heights = []

  def add_student_height(self, height):
    self.student_heights.append(height)

  def display_heights(self):
    for i, height in enumerate(self.student_heights, start=1):
      print(f"Student {i}: {height} cm")

# Example usage:
school = School()
school.add_student_height(160)
school.add_student_height(155)
school.add_student_height(170)

school.display_heights()
```

Method 2: Using a class to represent students and storing heights as individual attributes for each student.
I use this method.

21. Write a Python program that executes an operation on a list and handles an
# IndexError exception if the index is out of range.

Method 2: Using if statement to check index validity before accessing the element:

```
python
Copy code
def perform_operation(data, index):
  if 0 <= index < len(data):  # Check if the index is within the valid range
    result = data[index]
    print("Operation result:", result)
  else:
    print("Error: Index is out of range.")

# Example usage:
my_list = [1, 2, 3, 4, 5]
index = 6  # Index out of range
perform_operation(my_list, index)
```

Method 1: Using try-except block to catch the IndexError:

I use this method

24 Implement the game rock, paper, scissors Rock smashes scissors. Paper covers rock. Scissors cut paper.

```
import random

def play_game(player_choice):
  choices = ['rock', 'paper', 'scissors']
  computer_choice = random.choice(choices)

  print(f"Your choice: {player_choice}")
  print(f"Computer's choice: {computer_choice}")

  if player_choice == computer_choice:
    print("It's a tie!")
  elif (player_choice == 'rock' and computer_choice == 'scissors') or \
     (player_choice == 'paper' and computer_choice == 'rock') or \
     (player_choice == 'scissors' and computer_choice == 'paper'):
    print("You win!")
  else:
    print("Computer wins!")

# Main program
while True:
  print("\nRock, Paper, Scissors Game")
  print("Enter your choice ('rock', 'paper', or 'scissors') or 'quit' to exit:")

  user_input = input().lower()
```

```
if user_input == 'quit':
   print("Exiting the game...")
   break
elif user_input in ['rock', 'paper', 'scissors']:
   play_game(user_input)
else:
   print("Invalid choice! Please enter 'rock', 'paper', 'scissors', or 'quit'.")
```