MainFlow Task -1 Project Documentation

Name: Sana Perween
Intern ID: 15665
Domain: Python Programming

1. Write a program to calculate the sum of two numbers.

Objective The primary objective of this task is to create a Python program that calculates and outputs the sum of two integers. The program demonstrates basic input/output operations and arithmetic in Python.

Approach The program is designed to follow these steps:

1. **Input Handling**:
   - The program can prompt the user to enter two integers or use predefined values for testing.
   - The numbers are stored in variables number1 and number2.
2. **Computation**:
   - The sum of the two integers is calculated using the + operator.
   - The result is stored in a variable called result.
3. **Output**:
   - The result is displayed to the user in a formatted string for clarity.
4. **Testing**:
   - Example input values (number1 = 3 and number2 = 5) are used for testing.
   - The program prints the correct sum: 8.

Key Challenges

- **User Input Handling**:
  - When accepting input from the user, ensuring that only integers are entered can be a challenge. Input validation must be implemented to handle invalid inputs like strings or special characters.
- **Error Handling**:
  - Handling runtime errors, such as entering non-numeric input when the program expects integers, is crucial for program robustness.

Solutions

- **Input Validation**:
  - Use a try-except block to validate user input.
- **Code**:

```python
# 1. Objective: Write a program to calculate the sum of two numbers.

try:
    number1 = int(input("Enter the first number: "))
    number2 = int(input("Enter the second number: "))
except ValueError:
    print("Invalid input. Please enter integers only.")

# number1 = 3
# number2 = 5

result = number1 + number2

print(f'The sum of {number1} and {number2} is:', result)
```

2. Write a program to determine whether a number is odd or even.

Objective The primary objective of this task is to create a Python program that determines whether a given number is odd or even. The program uses basic input/output operations and conditional statements to achieve this goal.

Approach The program follows these steps:

1. **Input Handling**:
   - The program can prompt the user to enter a number or use predefined values for testing.
   - The input is stored in a variable named number.
2. **Computation**:
   - The program checks if number is divisible by 2 using modulus (%) operator
   - If number % 2 == 0, the number is even; otherwise, it is odd.
3. **Output**:
   - Based on the computation, the program prints whether the number is "Even" or "Odd".

Key Challenges

- **User Input Handling**:
  - Ensuring the user enters a valid integer. If the input is not a number, the program should handle the error gracefully.
- **Negative Numbers and Zero**:
  - Ensuring the program handles negative numbers and zero correctly without additional modifications.

Solutions

- **Input Validation**:
  - Use a try-except block to validate the user input. For example:
- **Code**:

```python
# 2. Determine whether a number is odd or even..

# number = int(input("Enter the number: "))
try:
    number = int(input("Enter the number: "))
except ValueError:
    print("Invalid input. Please enter an integer.")
    exit()

# number = 34

if number % 2 == 0:
    print("Even")
else:
    print("Odd")
```

3. Write a program to Compute the factorial of a given number nnn.

Objective The primary goal of this task is to create a Python program that calculates the factorial of a number using two methods:

1. Iterative approach (using a loop).
2. Recursive approach (using a function).

Approach The program implements two different methods to compute the factorial of a number:

Method 1: Iterative Approach

- **Input Handling**:
  - The user provides an integer input, which is stored in the variable num.
- **Computation**:
  - Initialize a variable factorial to 1.
  - Use a for loop to iterate from 1 to num (inclusive), multiplying factorial by each number in the range.

Method 2: Recursive Approach

- **Input Handling**:
  - The user provides an integer input, which is stored in the variable n.
- **Computation**:
  - Define a recursive function fact(n).
  - The base case is when n is 0 or 1, returning 1.
  - For all other values, the function returns n * fact(n-1).

Key Challenges

- **Input Validation**:
  - Ensuring the user enters a non-negative integer.
- **Handling Edge Cases**:
  - Correctly handling 0!, which is defined as 1.

Solutions

- **Input Validation**:
  - Use a try-except block to validate that the input is a non-negative integer.
- **Code:**

```python
# 3. Compute the factorial of a given number

num = int(input("Enter the number: "))

factorial = 1

for i in range(1, num + 1):
    factorial *= i

print(f'The factorial of {num} is', factorial)

# using recursive function

n = int(input("Enter the number: "))

def fact(n):
    if n==1 or n==0:
        return 1
    else:
        return n*fact(n-1)

print(f'Factorial of {n} using recursive method is', fact(n))
```

4. Write a program to Generate the first nnn numbers in the Fibonacci sequence.

Objective: This task aims to create a Python program that generates the Fibonacci series up to a specified number of terms. The Fibonacci series is a sequence of numbers where each term is the sum of the two preceding ones, starting from 0 and 1.

Approach The program follows these steps:

1. **Input Handling**:
   - The user inputs the number of terms (num) to generate in the Fibonacci series.
   - Input validation is performed to ensure the value is non-negative.
2. **Initialization**:
   - Two variables n1 and n2 are initialized to 0 and 1, representing the first two terms of the series.
   - A counter variable count is initialized to 0 to track the number of terms generated.
3. **Computation**:
   - If the input num is less than 0, the program prints an error message.
   - If num equals 1, the program outputs the first term of the Fibonacci series (0).
   - For inputs greater than 1, a while loop generates and prints the Fibonacci terms until the desired number of terms is reached. Each new term is calculated as the sum of the previous two terms.

Key Challenges

- **Input Validation**:
  - Ensuring the user inputs a valid non-negative integer.
- **Edge Cases**:
  - Handling inputs like 0 and 1 correctly.
- **Series Termination**:
  - Ensuring the loop terminates after the specified number of terms is generated.

---

4. Solutions

- **Input Validation**:
  - Use a conditional statement to check if the input is non-negative. If the input is invalid, print an appropriate error message and terminate the program.
- **Edge Case Handling**:
  - Include specific conditions for inputs 0 and 1 to handle them separately.
- **Code:**

```python
# 4.Generate the first nnn numbers in the Fibonacci sequence.

num = int(input("Enter the number of terms: "))

n1, n2 = 0, 1
count = 0

if num < 0:
    print("Invalid Input !!")
elif num == 1:
    print("Fibonacci Series: ", n1)
else:
    while count < num:
        print(n1)
        term = n1 + n2
        n1 = n2
        n2 = term
        count += 1
```

5. Write a program to reverse a string.

Objective: The objective of this program is to reverse a given string using two different methods:

1. String slicing.
2. A loop-based approach.

Approach The program demonstrates two approaches to reversing a string:

Method 1: Using String Slicing

1. **Input**:
   - A predefined string word is used: 'All that glitters is not gold.'
2. **Reversal Logic**:
   - String slicing with the syntax word[::-1] reverses the string efficiently by stepping through it in reverse order.

Method 2: Using a Loop

1. **Input**:
   - A predefined string new_word is used: 'All that glitters is not gold.'
2. **Reversal Logic**:
   - An empty string rev_word is initialized to store the reversed string.
   - A for loop iterates through each character in the original string.
   - Each character is added to the beginning of rev_word to construct the reversed string.

Key Challenges:

- **Efficiency**:
  - The slicing method is more efficient as it leverages Python's optimized string operations.
  - The loop-based method is less efficient due to repeated concatenations.

Code:

```
1   # 5. Reverse the characters in a string.
2
3   word = 'All that glitters is not gold.'
4
5   reversed_word = word[::-1]
6
7   print(reversed_word)
8
9   # <--Using loop-->
10
11  new_word = 'All that glitters is not gold.'
12
13  rev_word = " "
14
15  for char in new_word:
16      rev_word = char + rev_word
17
18  print(rev_word)
```

6. Write a program to Check if a string reads the same backward as forward.

Objective The objective of this program is to determine whether a given word is a palindrome. A palindrome is a word, phrase, or sequence that reads the same backward as forward (e.g., "radar", "level").

Approach The program uses the following steps to check if the input string is a palindrome:

1. **Input**:
   - The user is prompted to enter a word, which is stored in the variable s.
2. **Palindrome Check**:
   - A function palindrome(s) is defined that checks if the string s is equal to its reverse.
   - String slicing (s[::-1]) is used to reverse the string efficiently.
   - The function returns True if the input string matches its reverse, otherwise False.

Key Challenges

- **Case Sensitivity**:
  - By default, the program is case-sensitive (e.g., "Madam" will not match "madam").
- **Handling Non-Alphabetic Characters**:
  - Special characters, spaces, or numbers in the input will not affect the palindrome check.

Solution:

```
1   # 6. Check if a string reads the same backward as forward.
2
3   s = input("Enter the word: ")
4
5   def palindrome(s):
6       return s == s[::-1]
7
8   result = palindrome(s)
9
10  if result:
11      print("True, It's Palindrome")
12  else:
13      print("False, it's not Palindrome")
14
```

7. Write a program to determine whether a year is a leap year.

Objective The objective of this program is to determine whether a given year is a leap year. A leap year occurs every four years, with the exception of years divisible by 100 but not divisible by 400.

Approach The program uses the following steps to check if a given year is a leap year:

1. **Input**:
   - The user is prompted to enter a year, which is stored in the variable year.
2. **Leap Year Check**:
   - A function is_leap_year(year) is defined to determine if the input year satisfies the leap year conditions:
     - A year is a leap year if it is divisible by 4 and not divisible by 100.
     - Alternatively, a year is a leap year if it is divisible by 400.
   - The function returns True if the year is a leap year, otherwise False.
3. **Result Evaluation**:
   - The result of the function is stored in the variable leap_year.
   - The program prints whether the input year is a leap year.

Key Challenges

- **Understanding Leap Year Rules**:
  - Leap years occur under specific conditions that need to be carefully implemented in the program logic.

Solutions:

```python
# 7. Determine whether a year is a leap year.

#divivsible by 4 and  not divisible by 100 , divisible by 400

year = int(input("Enter the year to check: "))

def is_leap_year(year):
    return (year %4 ==0 and year %100 != 0) or (year %400 == 0)

leap_year = is_leap_year(year)
print(f'{year} is a leap year ? : ', leap_year)

```

8. Write a program Check if a number is an Armstrong number.

Objective: The primary objective of this task is to create a Python program that checks whether a given number is an Armstrong number. An Armstrong number is a number that is equal to the sum of its digits each raised to the power of the number of digits. This program demonstrates basic input/output operations, handling of loops, and arithmetic calculations in Python.

Approach

1. **Input Handling:**
   - The program prompts the user to enter a number. The user input is captured and stored in the variable number.
   - The length of the number (i.e., the number of digits) is calculated by converting the number to a string and using len() function.
2. **Computation:**

- The program initializes a temporary variable temp to the value of the number and sets a variable sum to zero.
- It enters a loop where the last digit of the number is extracted, raised to the power of the length of the number, and added to the sum.
- The last digit is then removed from the number by performing integer division by 10.

3. **Comparison:**
   - After the loop finishes, the program checks if the sum is equal to the original number. If they are equal, the number is an Armstrong number; otherwise, it is not.

Key Challenges

1. **User Input Handling:**
   - Ensuring that the user inputs a valid number without any non-numeric characters can be challenging.
   - The program assumes the user will enter a valid number, but it does not handle invalid input in the current implementation.
2. **Correctness of Armstrong Condition:**
   - The main challenge lies in correctly raising each digit to the power of the length of the number and summing them to check if they match the original number.

Solutions

```python
# 8.  Check if a number equals the sum of its digits raised to the powe

number = int(input("Enter the number: "))
length = len(str(number))

temp = number
sum = 0

while temp > 0:
    digit = temp % 10 # only remainder
    sum += digit ** length
    temp = temp // 10 # no decimal

if sum == number:
    print(f"{number} is an Armstrong number!!")
else:
    print(f"{number} is not an Armstrong number!!")

```

TASK : CUSTOM ENCRYPTION AND DECRYPTION PROGRAM REPORT

Objective

The primary objective of this program is to implement a custom encryption and decryption mechanism using two techniques:

1. **Substitution Cipher**: Each character in the message is substituted with another character based on a predefined mapping.
2. **Matrix Transformation**: The message is encrypted by swapping every pair of characters, making it more complex for decryption.

This program demonstrates encryption and decryption methods that can be used for basic text security and manipulation. It also shows how to use both character substitution and matrix transformations to enhance encryption.

Approach

1. **Substitution Cipher**:
    - A substitution cipher replaces each character in the original message with a corresponding character from a predefined mapping.
    - The program uses two strings: one (original) for the standard character set and another (substitution) for the mapped set.
    - **Encryption**: The program iterates over the message, replacing each character with its counterpart in the substitution string.
    - **Decryption**: The reverse process is applied by mapping the characters back to the original set.
2. **Matrix Transformation**:
    - This technique involves swapping every pair of adjacent characters in the string, introducing a second layer of encryption.
    - **Encryption**: Characters in the message are swapped in pairs.
    - **Decryption**: The same transformation is applied to reverse the process, restoring the original order of characters.

Key Functions

1. **create_substitution_cipher()**:
    - **Purpose**: This function initializes two strings: one representing the original character set and another representing the substituted set.
    - **Original Set**: "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789 .,!?;:"
    - **Substitution Set**: "QWERTYUIOPASDFGHJKLZXCVBNMmnbvcxzlkjhgfdsapoiuytrewq0987654321.,!?;:"
    - **Output**: The function returns these two sets to be used in encryption and decryption.
2. **substitution_encrypt(message, original, substitution)**:
    - **Purpose**: This function encrypts the input message using the substitution cipher technique.
    - **Process**:
        - It iterates over each character in the message.
        - If the character is found in the original string, it is replaced with the corresponding character from the substitution string.

        - If the character is not in the original string (like punctuation or spaces), it is retained as is.
    - **Output**: The function returns the encrypted message.
3. **substitution_decrypt(encrypted_message, original, substitution)**:
    - **Purpose**: This function decrypts the encrypted message using the substitution cipher technique.
    - **Process**:
        - It iterates over each character in the encrypted message.
        - If the character is found in the substitution string, it is replaced with the corresponding character from the original string.
        - If the character is not in the substitution string, it remains unchanged.
    - **Output**: The function returns the decrypted message.
4. **matrix_transform_encrypt(message)**:
    - **Purpose**: This function applies a matrix transformation to the encrypted message by swapping every pair of adjacent characters.
    - **Process**:
        - It iterates over the message in steps of two characters.
        - If a pair is present, the characters are swapped and added to the encrypted string.
        - If there's an odd number of characters, the last character remains in place.
    - **Output**: The function returns the transformed encrypted message.
5. **matrix_transform_decrypt(encrypted_message)**:
    - **Purpose**: This function reverses the matrix transformation by swapping adjacent characters back to their original positions.
    - **Process**:
        - It iterates over the encrypted message in steps of two characters.
        - If a pair is present, the characters are swapped back to their original positions and added to the decrypted string.
        - If there's an odd number of characters, the last character remains unchanged.
    - **Output**: The function returns the decrypted message after matrix transformation.
6. **main()**:
    - **Purpose**: This is the main function that ties together the entire encryption and decryption process.
    - **Process**:
        - The user is prompted to input a message.
        - The message is first encrypted using the substitution cipher.
        - After that, matrix transformation is applied to the already encrypted message.
        - The program then prints the final encrypted message.
        - For decryption, the program reverses the matrix transformation and then the substitution cipher decryption.
        - The decrypted message is printed.

Key Challenges

1. **Character Mapping**:
    - Ensuring that all characters in the message have corresponding counterparts in the original and substitution sets is crucial.

- Non-alphanumeric characters must be handled gracefully to avoid unexpected behavior during encryption or decryption.

2. **Efficient Reversal of Matrix Transformation**:
   - The matrix transformation is a simple swap, but it must be applied consistently in both encryption and decryption to maintain symmetry.
3. **Handling Special Characters**:
   - The program is designed to handle a wide variety of characters, such as punctuation, spaces, and digits, without modifying them during the encryption/decryption processes.

Solutions

1. **Character Mapping**:
   - The program checks whether each character exists in the original and substitution sets before applying transformations. This ensures that only characters that need to be encrypted/decrypted are processed.
2. **Efficient Reversal of Matrix Transformation**:
   - Since the matrix transformation is reversible, applying the transformation twice (once for encryption and once for decryption) ensures the correct outcome.
3. **Handling Special Characters**:
   - The program retains non-alphabetic characters (like spaces, punctuation, and numbers) as they are during both encryption and decryption. This preserves the original structure of the message.

Output: