Jainam Jagani
Intern ID: 17696

TASK-1

1. Sum of Two Numbers

Description of Problem: This problem involves adding two integer numbers. The user provides two integers as input, and the task is to calculate and display their sum.

Approach:

- Understand the core requirement of the problem
- Break down the steps logically
- Write and test simple working code

Key Challenges:

- Handling invalid input or unexpected behavior
- Ensuring accuracy of logic with edge cases

Solutions:

- Use input validation (if needed)
- Test with multiple edge cases to confirm output

Logic Used:

- Accept two integers using the `input()` function.
- Convert the input strings to integers using `int()`.
- Add the two integers using the + operator.
- Print the result using `print()`.

```
num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))
print("The sum is:", num1 + num2)
```

What I Learned Through This Problem:

- Taking input from users
- Type conversion from string to integer
- Using arithmetic operators in Python

Edge Cases Handled:

- Non-integer input will cause an error unless validated
- Negative numbers and zero are correctly supported

Output Example:

```
Enter first number: 4
Enter second number: 5
The sum is: 9
```

---

2. Odd or Even

Description of Problem: This problem checks whether a given integer is odd or even. An even number is divisible by 2, whereas an odd number is not.

Approach:

- Understand the core requirement of the problem
- Break down the steps logically
- Write and test simple working code

Key Challenges:

- Handling invalid input or unexpected behavior
- Ensuring accuracy of logic with edge cases

Solutions:

- Use input validation (if needed)
- Test with multiple edge cases to confirm output

Logic Used:

- Take an integer input from the user.
- Use the modulus operator % to find the remainder when divided by 2.
- If remainder is 0, it's even; otherwise, it's odd.

```python
num = int(input("Enter a number: "))
if num % 2 == 0:
    print("Even")
else:
    print("Odd")
```

What I Learned Through This Problem:

- Use of conditional statements (`if-else`)
- Modulo operator for divisibility checks

Edge Cases Handled:

- Works for negative integers and zero

Output Example:

```
Enter a number: 7
Odd
```

---

3. Factorial Calculation

Description of Problem: Factorial of a number n (denoted as n!) is the product of all positive integers less than or equal to n. For example, 5! = 5 × 4 × 3 × 2 × 1 = 120.

Approach:

- Understand the core requirement of the problem
- Break down the steps logically
- Write and test simple working code

Key Challenges:

- Handling invalid input or unexpected behavior
- Ensuring accuracy of logic with edge cases

Solutions:

- Use input validation (if needed)
- Test with multiple edge cases to confirm output

Logic Used:

- Take an integer input from the user.
- Use a loop from 1 to n, multiplying each number.
- Store the running product in a variable.

```python
n = int(input("Enter a number: "))
factorial = 1
for i in range(1, n + 1):
    factorial *= i
print("Factorial is:", factorial)
```

What I Learned Through This Problem:

- Looping constructs in Python
- Multiplicative accumulation
- Mathematical concept of factorial

Edge Cases Handled:

- Factorial of 0 is 1 (handled by default loop)
- Negative input should be validated as invalid

Output Example:

```
Enter a number: 4
Factorial is: 24
```

---

4. Fibonacci Sequence

Description of Problem: The Fibonacci sequence is a series where each number is the sum of the two preceding ones, starting from 0 and 1. For example: 0, 1, 1, 2, 3, 5, 8...

Approach:

- Understand the core requirement of the problem
- Break down the steps logically
- Write and test simple working code

Key Challenges:

- Handling invalid input or unexpected behavior
- Ensuring accuracy of logic with edge cases

Solutions:

- Use input validation (if needed)
- Test with multiple edge cases to confirm output

Logic Used:

- Input how many terms to generate.
- Initialize the first two terms (0 and 1).
- Use a loop to generate the next term by summing the last two terms.

```
n = int(input("Enter number of Fibonacci terms: "))
a, b = 0, 1
for _ in range(n):
    print(a, end=' ')
    a, b = b, a + b
```

What I Learned Through This Problem:

- Sequence generation

- Tuple unpacking for swapping values
- Loop-based logic

Edge Cases Handled:

- Input = 0 should print nothing
- Input = 1 prints only the first number

Output Example:

```
Enter number of Fibonacci terms: 6
0 1 1 2 3 5
```

---

5. Reverse a String

Description of Problem: This problem involves reversing the characters of a string. For example, the reverse of “hello” is “olleh”.

Approach:

- Understand the core requirement of the problem
- Break down the steps logically
- Write and test simple working code

Key Challenges:

- Handling invalid input or unexpected behavior
- Ensuring accuracy of logic with edge cases

Solutions:

- Use input validation (if needed)
- Test with multiple edge cases to confirm output

Logic Used:

- Use Python’s slicing syntax `[::-1]` to reverse the string.
- Alternatively, loop through each character and prepend it to a new string.

```
text = input("Enter a string: ")
print("Reversed string:", text[::-1])
```

What I Learned Through This Problem:

- String slicing
- Understanding of string indices in Python

Edge Cases Handled:

- Empty strings return empty strings
- Works with spaces and special characters

Output Example:

```
Enter a string: hello
Reversed string: olleh
```

---

6. Palindrome Check

Description of Problem: A palindrome is a string that reads the same forwards and backwards, e.g., “madam” or “racecar”.

Approach:

- Understand the core requirement of the problem
- Break down the steps logically
- Write and test simple working code

Key Challenges:

- Handling invalid input or unexpected behavior
- Ensuring accuracy of logic with edge cases

Solutions:

- Use input validation (if needed)
- Test with multiple edge cases to confirm output

Logic Used:

- Reverse the string using slicing.
- Compare the original string to the reversed string.
- Return `True` if they match.

```
text = input("Enter a string: ")
print("Palindrome:", text == text[::-1])
```

What I Learned Through This Problem:

- String comparison
- Palindromic logic and reversal

Edge Cases Handled:

- Case-sensitivity affects the result (e.g., “Madam” ≠ “madam”)
- Spaces and punctuation affect validity

Output Example:

```
Enter a string: madam
Palindrome: True
```

---

7. Leap Year Check

Description of Problem: A leap year has 366 days. A year is a leap year if:

- It is divisible by 4 and not by 100, OR
- It is divisible by 400

Approach:

- Understand the core requirement of the problem
- Break down the steps logically
- Write and test simple working code

Key Challenges:

- Handling invalid input or unexpected behavior
- Ensuring accuracy of logic with edge cases

Solutions:

- Use input validation (if needed)
- Test with multiple edge cases to confirm output

Logic Used:

- Use modulo operator and logical conditions to check leap year rule.

```python
year = int(input("Enter a year: "))
is_leap = (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)
print("Leap Year:", is_leap)
```

What I Learned Through This Problem:

- Logical operators (and, or)
- Real-world rules encoded in code

Edge Cases Handled:

- Works for any 4-digit or more year
- Handles century years (e.g., 1900 is not, but 2000 is a leap year)

Output Example:

```
Enter a year: 2024
Leap Year: True
```

---

8. Armstrong Number

Description of Problem: An Armstrong number is an n-digit number equal to the sum of its digits each raised to the power n. Example: $153 = 1^3 + 5^3 + 3^3 = 153$.

Approach:

- Understand the core requirement of the problem
- Break down the steps logically
- Write and test simple working code

Key Challenges:

- Handling invalid input or unexpected behavior
- Ensuring accuracy of logic with edge cases

Solutions:

- Use input validation (if needed)
- Test with multiple edge cases to confirm output

Logic Used:

- Convert the number to a string to get digit count.
- Use a loop or list comprehension to calculate the powered sum of digits.
- Compare the sum with the original number.

```python
num = int(input("Enter a number: "))
digits = str(num)
power = len(digits)
total = sum(int(d)**power for d in digits)
print("Armstrong Number:", total == num)
```

What I Learned Through This Problem:

- Working with digits of a number

- Power operations and list comprehensions

Edge Cases Handled:

- Works with single-digit numbers
- Invalid for negative numbers (not Armstrongs)

Output Example:

```
Enter a number: 153
Armstrong Number: True
```

---

9. Custom Encryption-Decryption System

Description of Problem: This problem involves creating a basic encryption and decryption system. The program should transform text into an unreadable format (encryption) and then back to original (decryption) using a key.

Approach:

- Understand the core requirement of the problem
- Break down the steps logically
- Write and test simple working code

Key Challenges:

- Handling invalid input or unexpected behavior
- Ensuring accuracy of logic with edge cases

Solutions:

- Use input validation (if needed)
- Test with multiple edge cases to confirm output

Logic Used:

- For encryption:
    - Shift alphanumeric characters using their Unicode values.
    - Reverse the final string for added security.
- For decryption:
    - Reverse the encrypted string.
    - Apply the inverse of the character shift.

```
def encrypt(msg, key):
    encrypted = ""
    for char in msg:
        encrypted += chr((ord(char) + key) % 256) if char.isalnum() else char
```

```
    return encrypted[::-1]

def decrypt(msg, key):
    reversed_msg = msg[::-1]
    decrypted = ""
    for char in reversed_msg:
        decrypted += chr((ord(char) - key) % 256) if char.isalnum() else char
    return decrypted
```

What I Learned Through This Problem:

- Algorithm design for text transformation
- Unicode manipulation using `ord()` and `chr()`
- Reversible logic with string reversal and encryption keys

Edge Cases Handled:

- Non-alphanumeric characters are preserved
- Works with spaces, punctuation, and numbers

Output Example:

```
Enter message: Hello123
Key: 3
Encrypted: 654oorKho
Decrypted: Hello123
```