Jainam Jagani
Intern ID: 17696

TASK-2

1. Prime Number

Description of Problem: This problem checks whether a given number is prime, i.e., divisible only by 1 and itself.

Approach:

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

Key Challenges:

• Avoiding unnecessary checks • Optimizing time complexity for large numbers

Solutions:

• Loop from 2 up to n // 2 (avoiding full range) • Skip even numbers after checking for 2

Logic Used:

• Take input from the user • If number is less than or equal to 1, it's not prime • Check if divisible by 2, then skip even checks • Loop through odd numbers only up to n // 2

```
# Function to check for prime
def is_prime(n):
  if n <= 1:
    return False
  if n == 2:
    return True
  if n % 2 == 0:
    return False
  for i in range(3, (n // 2) + 1, 2):
    if n % i == 0:
      return False
  return True

# Input
num = int(input("Enter a number: "))

# Output
print("Prime Number:", is_prime(num))
```

What I Learned Through This Problem:

• Efficient prime-checking logic without using math library • Optimization using odd number iteration

Edge Cases Handled:

• 0 and 1 are not prime • Works for even and negative numbers

Output Example:

Enter a number: 13 Prime Number: True

---

2. Sum of Digits

Description of Problem: Find the sum of all individual digits in a given number.

Approach:

- Convert number to string
- Loop through characters and convert each back to integer
- Sum them up

Logic Used:

```
n = int(input("Enter a number: "))
digit_sum = sum(int(digit) for digit in str(n))
print("Sum of digits:", digit_sum)
```

What I Learned Through This Problem:

- String manipulation
- Summing individual digits

Edge Cases Handled:

- Negative numbers can be converted using `abs(n)` if needed

Output Example:

```
Enter a number: 1234
Sum of digits: 10
```

Jainam Jagani
Intern ID: 17696

3. LCM and GCD

Description of Problem: This problem involves calculating the Least Common Multiple (LCM) and Greatest Common Divisor (GCD) without using Python's math module.

Approach:

• Understand the mathematical relation between GCD and LCM • Implement Euclidean algorithm manually

Key Challenges:

• Avoiding division by zero • Handling negative inputs

Solutions:

• Implement while loop for Euclidean GCD • Use the formula: LCM = abs(a*b) // GCD

Logic Used:

• Accept two integers as input • Implement Euclidean algorithm for GCD • Calculate LCM using GCD result

```
# GCD using Euclidean Algorithm
def compute_gcd(a, b):
  while b != 0:
    a, b = b, a % b
  return abs(a)

# LCM using GCD
def compute_lcm(a, b):
  gcd = compute_gcd(a, b)
  return abs(a * b) // gcd if gcd != 0 else 0

# Input
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))

# Output
gcd = compute_gcd(a, b)
lcm = compute_lcm(a, b)
print("GCD:", gcd)
print("LCM:", lcm)
```

What I Learned Through This Problem:

• Euclidean GCD logic without math functions • Integer arithmetic for LCM

Edge Cases Handled:

• Zero or negative input handled

Output Example:

Enter first number: 12 Enter second number: 18 GCD: 6 LCM: 36

---

4. List Reversal

Description of Problem: Reverse a list of integers without using built-in functions.

Approach:

- Use a two-pointer technique to swap elements

Logic Used:

```
lst = [int(x) for x in input("Enter list elements separated by spaces:
").split()]
start, end = 0, len(lst) - 1
while start < end:
    lst[start], lst[end] = lst[end], lst[start]
    start += 1
    end -= 1
print("Reversed List:", lst)
```

What I Learned Through This Problem:

- Index manipulation
- Swapping values using a loop

Edge Cases Handled:

- Empty or single-element list

Output Example:

```
Enter list elements separated by spaces: 1 2 3 4
Reversed List: [4, 3, 2, 1]
```

---

5. Sort a List

Description of Problem: Sort a list of integers in ascending order without using built-in sort functions.

Approach:

- Implement bubble sort algorithm

Logic Used:

```
lst = [int(x) for x in input("Enter list elements separated by spaces:
").split()]
n = len(lst)
for i in range(n):
    for j in range(0, n - i - 1):
        if lst[j] > lst[j + 1]:
            lst[j], lst[j + 1] = lst[j + 1], lst[j]
print("Sorted List:", lst)
```

What I Learned Through This Problem:

- Sorting logic and nested loops

Edge Cases Handled:

- List with duplicates

Output Example:

```
Enter list elements separated by spaces: 5 3 1 4
Sorted List: [1, 3, 4, 5]
```

---

6. Remove Duplicates

Description of Problem: Remove duplicate values from a list.

Approach:

- Use a set to track seen elements
- Build a new list with unique values

Logic Used:

```
lst = [int(x) for x in input("Enter list elements separated by spaces:
").split()]
unique_list = []
seen = set()
for num in lst:
    if num not in seen:
        unique_list.append(num)
        seen.add(num)
print("List with duplicates removed:", unique_list)
```

What I Learned Through This Problem:

- Set operations
- Maintaining insertion order while removing duplicates

Edge Cases Handled:

- All duplicates or all unique elements

Output Example:

```
Enter list elements separated by spaces: 1 2 2 3 4 4
List with duplicates removed: [1, 2, 3, 4]
```

---

7. String Length

Description of Problem: Find the length of a string without using `len()`.

Approach:

- Use a loop to count characters one by one

Logic Used:

```python
text = input("Enter a string: ")
count = 0
for char in text:
    count += 1
print("Length of the string:", count)
```

What I Learned Through This Problem:

- Character iteration in strings
- Manual counting logic

Edge Cases Handled:

- Empty strings

Output Example:

```
Enter a string: hello
Length of the string: 5
```

---

8. Count Vowels and Consonants

Description of Problem: Count the number of vowels and consonants in a string.

Approach:

- Define a set of vowels
- Check each character using `.isalpha()`

Logic Used:

```
text = input("Enter a string: ")
vowels = set('aeiouAEIOU')
vowel_count = consonant_count = 0
for char in text:
    if char.isalpha():
        if char in vowels:
            vowel_count += 1
        else:
            consonant_count += 1
print("Vowels:", vowel_count)
print("Consonants:", consonant_count)
```

What I Learned Through This Problem:

- Set usage
- Alphabet character filtering

Edge Cases Handled:

- Strings with numbers and symbols

Output Example:

```
Enter a string: Hello World
Vowels: 3
Consonants: 7
```

---

9. Maze Generator and Solver

Description of Problem: Generate and solve a random maze using DFS algorithm and represent it in terminal using text.

Approach:

- Represent the maze as a 2D list of walls (1s) and paths (0s)
- Use recursive backtracking (DFS) to generate a maze
- Use DFS to solve from top-left to bottom-right

Logic Used:

```
import random
import sys
sys.setrecursionlimit(10000)
```

```
WIDTH = 21
HEIGHT = 21
WALL, PATH, VISITED = 1, 0, 2
DIRS = [(-2,0),(2,0),(0,-2),(0,2)]
maze = [[WALL]*WIDTH for _ in range(HEIGHT)]

def print_maze(m):
    for row in m:
        print(''.join(['' if cell == WALL else '.' if cell == PATH else '*'
for cell in row]))

def is_valid(x, y):
    return 0 < x < HEIGHT-1 and 0 < y < WIDTH-1

def generate_maze(x, y):
    maze[x][y] = PATH
    random.shuffle(DIRS)
    for dx, dy in DIRS:
        nx, ny = x + dx, y + dy
        if is_valid(nx, ny) and maze[nx][ny] == WALL:
            maze[x + dx//2][y + dy//2] = PATH
            generate_maze(nx, ny)

def solve_maze(x, y, end_x, end_y):
    if not is_valid(x, y) or maze[x][y] != PATH:
        return False
    if (x, y) == (end_x, end_y):
        maze[x][y] = VISITED
        return True
    maze[x][y] = VISITED
    for dx, dy in [(-1,0),(1,0),(0,-1),(0,1)]:
        if solve_maze(x+dx, y+dy, end_x, end_y):
            return True
    maze[x][y] = PATH
    return False

start_x, start_y = 1, 1
end_x, end_y = HEIGHT - 2, WIDTH - 2
generate_maze(start_x, start_y)
print("Generated Maze:")
print_maze(maze)
print("\nSolving Maze...")
if solve_maze(start_x, start_y, end_x, end_y):
    print("\nSolved Maze:")
    print_maze(maze)
else:
    print("No solution found.")
```

What I Learned Through This Problem:

- Graph traversal (DFS)
- Recursive thinking and backtracking
- Maze representation and visualization using text

Edge Cases Handled:

- Path always generated to be solvable

Output :

- Maze printed using ▉ for walls, . for paths, and * for solution path