Main Flow Internship

TASK-2

Ankith R

1. Prime Number:

```python
# Prime Number
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
n = int(input("Enter a number: "))
print(is_prime(n))
```

```
Enter a number: 7
True

=== Code Execution Successful ===
```

2. Sum of Digits:

```python
# Sum of Digits
def sum_of_digits(n):
    return sum(int(digit) for digit in str(abs(n)))
n = int(input("Enter a number: "))
print(sum_of_digits(n))
```

```
Enter a number: 789
24

=== Code Execution Successful ===
```

3. LCM and GCD:

```python
#  LCM and GCD
import math
def lcm_gcd(a, b):
    gcd = math.gcd(a, b)
    lcm = abs(a * b) // gcd
    return lcm, gcd
a, b = map(int, input("Enter two numbers: ").split())
lcm, gcd = lcm_gcd(a, b)
print("LCM:", lcm)
print("GCD:", gcd)
```

```
Enter two numbers: 18 27
LCM: 54
GCD: 9

=== Code Execution Successful ===
```

4. List Reversal:

```python
#  List Reversal
def reverse_list(lst):
    left, right = 0, len(lst) - 1
    while left < right:
        lst[left], lst[right] = lst[right], lst[left]
        left += 1
        right -= 1
    return lst
lst = list(map(int, input("Enter numbers separated by space: ").split()))
print("Reversed List:", reverse_list(lst))
```

```
Enter numbers separated by space: 2 3 4 5 6 7
Reversed List: [7, 6, 5, 4, 3, 2]

=== Code Execution Successful ===
```

5. Sort a List:

```python
#  Sort a List
def bubble_sort(lst):
    n = len(lst)
    for i in range(n):
        for j in range(n - i - 1):
            if lst[j] > lst[j + 1]:
                lst[j], lst[j + 1] = lst[j + 1], lst[j]
    return lst
lst = list(map(int, input("Enter numbers separated by space: ").split()))
print("Sorted List:", bubble_sort(lst))
```

```
Enter numbers separated by space: 7 3 5 9 2 4 8
Sorted List: [2, 3, 4, 5, 7, 8, 9]

=== Code Execution Successful ===
```

6. Remove Duplicates:

```python
#  Remove Duplicates
def remove_duplicates(lst):
    unique_list = []
    for num in lst:
        if num not in unique_list:
            unique_list.append(num)
    return unique_list
lst = list(map(int, input("Enter numbers separated by space: ").split()))
```

```
Enter numbers separated by space: 5 6 7 5 3 2 7 8 2
List without duplicates: [5, 6, 7, 3, 2, 8]

=== Code Execution Successful ===
```

7. String Length:

```python
#  String Length
def string_length(s):
    count = 0
    for _ in s:
        count += 1
    return count
s = input("Enter a string: ")
print("Length of the string:", string_length(s))
```

```
Enter a string: 123456789
Length of the string: 9

=== Code Execution Successful ===
```

8. Count Vowels and Consonants:

```python
#  Count Vowels and Consonants
def count_vowels_consonants(s):
    vowels = set("aeiouAEIOU")
    v_count = c_count = 0

    for char in s:
        if char.isalpha():
            if char in vowels:
                v_count += 1
            else:
                c_count += 1

    return v_count, c_count
s = input("Enter a string: ")
vowels, consonants = count_vowels_consonants(s)
print("Vowels:", vowels)
print("Consonants:", consonants)
```

```
Enter a string: Main Flow Services and Technologies
Vowels: 12
Consonants: 19

=== Code Execution Successful ===
```

9. Maze Generator and Solver:

```python
# MAZE
import random
from collections import deque
def create_maze(rows=11, cols=11):
    """Generates a random maze using DFS."""
    maze = [['#'] * cols for _ in range(rows)]
    def dfs(r, c):
        maze[r][c] = ' '
        directions = [(-2, 0), (2, 0), (0, -2), (0, 2)]
        random.shuffle(directions)
        for dr, dc in directions:
            nr, nc = r + dr, c + dc
            if 1 <= nr < rows - 1 and 1 <= nc < cols - 1 and maze[nr][nc] == '#':
                maze[r + dr // 2][c + dc // 2] = ' '
                dfs(nr, nc)
    dfs(1, 1)
    maze[1][1] = 'S'
    maze[rows - 2][cols - 2] = 'E'
    return maze
def solve_maze(maze):
    """Finds the shortest path using BFS."""
    rows, cols = len(maze), len(maze[0])
    start, end = (1, 1), (rows - 2, cols - 2)
    queue = deque([(start, [start])])
    visited = set([start])
    while queue:
        (r, c), path = queue.popleft()
        if (r, c) == end:
            return path
        for dr, dc in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            nr, nc = r + dr, c + dc
            if 0 <= nr < rows and 0 <= nc < cols and maze[nr][nc] in (' ', 'E') and (nr, nc
                ) not in visited:
                queue.append(((nr, nc), path + [(nr, nc)]))
                visited.add((nr, nc))
    return None
def display_maze(maze, path=None):
    """Prints the maze, highlighting the solution path."""
    for r in range(len(maze)):
        for c in range(len(maze[0])):
            if path and (r, c) in path and maze[r][c] not in ('S', 'E'):
                print('.', end='')  # Solution path
            else:
                print(maze[r][c], end='')
        print()
maze = create_maze()
solution = solve_maze(maze)

print("\nGenerated Maze:\n")
display_maze(maze)

if solution:
    print("\nSolved Maze:\n")
    display_maze(maze, solution)
else:
    print("\nNo solution found.")

```

```
Generated Maze:

###########
#S  #     #
### # ### #
# # #   # #
# # ### # #
# #   # # #
# ### # ###
#   # #   #
# # # ### #
# #       E#
###########

Solved Maze:

###########
#S..#     #
###.# ### #
# #.#   # #
# #.### # #
# #...# # #
# ###.# ###
#   #.#   #
# # #.### #
# #  ....E#
###########

=== Code Execution Successful ===
```

```
Generated Maze:

###########
#S  #     #
### # ### #
# # #   # #
# # ### # #
# #   # # #
# ### # ###
#   # #   #
# # # ### #
# #       E#
###########

Solved Maze:

###########
#S..#     #
###.# ### #
# #.#   # #
# #.### # #
# #...# # #
# ###.# ###
#   #.#   #
# # #.### #
# #  ....E#
###########

=== Code Execution Successful ===
```