# Programming Tasks (9 to 16) & Maze Generator/Solver

## 9. Prime Number

Objective: Determine if a number is prime.

Code:

```
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5)+1):
        if n % i == 0:
            return False
    return True
```

## 10. Sum of Digits

Objective: Find the sum of the digits in a number.

Code:

```
def sum_of_digits(n):
    return sum(int(d) for d in str(abs(n)))
```

## 11. LCM and GCD

Objective: Calculate the LCM and GCD of two integers.

Code:

```
import math
def gcd_lcm(a, b):
    gcd = math.gcd(a, b)
    lcm = abs(a * b) // gcd
    return lcm, gcd
```

## 12. List Reversal

Objective: Reverse a list without built-in functions.

Code:

```
def reverse_list(lst):
    n = len(lst)
    for i in range(n // 2):
        lst[i], lst[n - i - 1] = lst[n - i - 1], lst[i]
    return lst
```

## 13. Sort a List

Objective: Sort a list in ascending order (Bubble Sort).

Code:

```
def bubble_sort(lst):
    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]
    return lst
```

# Programming Tasks (9 to 16) & Maze Generator/Solver

## 14. Remove Duplicates

```
Objective: Remove duplicates from a list.
Code:
def remove_duplicates(lst):
    unique = []
    for item in lst:
        if item not in unique:
            unique.append(item)
    return unique
```

## 15. String Length

```
Objective: Find string length without len().
Code:
def string_length(s):
    count = 0
    for _ in s:
        count += 1
    return count
```

## 16. Count Vowels and Consonants

```
Objective: Count vowels and consonants in a string.
Code:
def count_vowels_consonants(s):
    vowels = set('aeiouAEIOU')
    v_count = c_count = 0
    for ch in s:
        if ch.isalpha():
            if ch in vowels:
                v_count += 1
            else:
                c_count += 1
    return v_count, c_count
```

## 17. Maze Generator and Solver

```
Objective: Build and solve a maze using DFS.

Maze Generation:
def generate_maze(rows, cols):
    ...  # DFS-based backtracking maze generation

Maze Solving:
def solve_maze(maze, start, end):
    ...  # DFS or BFS based pathfinding

Print Maze:
def print_maze(maze, path=[]):
    ...  # Use '#' for walls and '.' for path
```

# Programming Tasks (9 to 16) & Maze Generator/Solver

Restrictions:
- No external libraries
- Use plain text visualization
- Must submit within 7 days

Learning Outcome:
- Understand recursive algorithms and graph traversal.