 # Prime Number

Objective: Determine if a number is prime.

Input: A single integer nnn.

Program :

```
import math

def is_prime(n):
  if n <= 1:
    return False
  for i in range(2, int(math.sqrt(n)) + 1):
    if n % i == 0:
      return False
  return True

num = int(input("Enter a number :"))
print(is_prime(num))
```

output : Enter a number :2

True

#Sum of Digits

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

Input: An integer nnn

Program :

```
def sum_of_digits(n):

  if n < 0:
    n = abs(n)
  sum_digits = 0
```

```
 while n > 0:
  sum_digits += n % 10
  n //= 10
 return sum_digits

number = 12345
result = sum_of_digits(number)
print(f"The sum of the digits in {number} is: {result}")
```

output :

The sum of the digits in 12345 is: 15

#. LCM and GCD

Objective: Calculate the Least Common Multiple (LCM) and Greatest Common Divisor (GCD) of two integers.

Input: Two integers aaa and bbb

Program :

```
def gcd(a, b):

 while b:
  a, b = b, a % b
 return a

def lcm(a, b):

 return (a * b) // gcd(a, b)

num1 = 24
```

```
num2 = 36

result_gcd = gcd(num1, num2)
result_lcm = lcm(num1, num2)

print(f"GCD of {num1} and {num2} is: {result_gcd}")
print(f"LCM of {num1} and {num2} is: {result_lcm}")
```

Output :

```
GCD of 24 and 36 is: 12
LCM of 24 and 36 is: 72
```

 # List Reversal

Objective: Reverse a given list without using built-in functions.

Input: A list of integers

Program :

```
def reverse_list(lst):

 left = 0
 right = len(lst) - 1

 while left < right:

  lst[left], lst[right] = lst[right], lst[left]
  left += 1
  right -= 1

 return lst

my_list = [1, 2, 3, 4, 5]
```

```
print(f"Original list: {my_list}")
reversed_list = reverse_list(my_list)
print(f"Reversed list: {reversed_list}")
```

Output :

Original list: [1, 2, 3, 4, 5]

Reversed list: [5, 4, 3, 2, 1]

#**Sort a List**

Objective: Sort a list of numbers in ascending order.

Input: A list of integers.

Program :

```
def sort_list(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

# Example usage:
my_list = [5, 2, 8, 1, 9, 4]
print(f"Original list: {my_list}")
sorted_list = sort_list(my_list)
print(f"Sorted list: {sorted_list}")
```

Output :

Original list: [5, 2, 8, 1, 9, 4]

Sorted list: [1, 2, 4, 5, 8, 9]

#Remove Duplicates

Objective: Remove duplicate elements from a list.

Input: A list of integers.

Program :

```
def remove_duplicates(lst):

 seen = []
 result = []

 for item in lst:
  if item not in seen:
   seen.append(item)
   result.append(item)

 return result

my_list = [1, 2, 2, 3, 4, 4, 5]
print(f"Original list: {my_list}")
unique_list = remove_duplicates(my_list)
print(f"List with duplicates removed: {unique_list}")
```

Output :

Original list: [1, 2, 2, 3, 4, 4, 5]

List with duplicates removed: [1, 2, 3, 4, 5]

#String Length

Objective: Find the length of a string without using the len() function.

Input: A string.

Program :

```
def string_length(s):

  count = 0
  for char in s:
    count += 1
  return count

my_string = "Hello, world!"
length = string_length(my_string)
print(f"The length of '{my_string}' is: {length}")
```

Output :

```
The length of 'Hello, world!' is: 13
```

#Count Vowels and Consonants

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

Input: A string.

Program :

```
def count_vowels_and_consonants(s):

  vowels = "aeiouAEIOU"
  vowel_count = 0
  consonant_count = 0
```

```
 for char in s:
   if char.isalpha():
     if char in vowels:
       vowel_count += 1
     else:
       consonant_count += 1

 return vowel_count, consonant_count

my_string = "Hello, world!"
vowel_count, consonant_count = count_vowels_and_consonants(my_string)
print(f"Vowel count: {vowel_count}")
print(f"Consonant count: {consonant_count}")
```

Output :

Vowel count: 3

Consonant count: 7

#Description: Build a program that generates random mazes and solves them using techniques like Depth-First Search (DFS) or Breadth-First Search (BFS).

Program :

```
import random

def generate_maze(rows, cols):


   maze = [[1] * cols for _ in range(rows)]
   start = (1, 1)
   maze[start[0]][start[1]] = 0
   stack = [start]
```

```
    def get_neighbors(r, c):
        neighbors = []
        if r > 1:
            neighbors.append((r - 2, c))
        if r < rows - 2:
            neighbors.append((r + 2, c))
        if c > 1:
            neighbors.append((r, c - 2))
        if c < cols - 2:
            neighbors.append((r, c + 2))
        return neighbors

    while stack:
        r, c = stack[-1]
        neighbors = get_neighbors(r, c)
        unvisited_neighbors = [(nr, nc) for nr, nc in neighbors if maze[nr][nc] == 1]

        if unvisited_neighbors:
            nr, nc = random.choice(unvisited_neighbors)
            maze[nr][nc] = 0
            maze[(r + nr) // 2][(c + nc) // 2] = 0
            stack.append((nr, nc))
        else:
            stack.pop()
    return maze

def solve_maze_dfs(maze, start, end):
    """Solves a maze using Depth-First Search."""
    rows = len(maze)
    cols = len(maze[0])
```

```
    visited = [[False] * cols for _ in range(rows)]
    path = []

    def dfs(r, c):
        if r < 0 or r >= rows or c < 0 or c >= cols or maze[r][c] == 1 or visited[r][c]:
            return False
        visited[r][c] = True
        path.append((r, c))

        if (r, c) == end:
            return True

        if dfs(r + 1, c) or dfs(r - 1, c) or dfs(r, c + 1) or dfs(r, c - 1):
            return True

        path.pop()
        return False

    if dfs(start[0], start[1]):
        return path
    else:
        return None

def print_maze(maze, path=None):
    """Prints the maze to the console, optionally highlighting the path."""
    if path is None:
        path = []
    for r, row in enumerate(maze):
        for c, cell in enumerate(row):
            if (r, c) in path:
                print(".", end=" ")
```

```
            elif cell == 1:
                print("#", end=" ")
            else:
                print(" ", end=" ")
        print()

rows = 15
cols = 25
maze = generate_maze(rows, cols)
start = (1, 1)
end = (rows - 2, cols - 2)

print("Generated Maze:")
print_maze(maze)

solution = solve_maze_dfs(maze, start, end)

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

Output :

Generated Maze:

```
# # # # # # # # # # # # # # # # # # # # # # # # #
#   #       #       #
# ### ##### # ##### ### #
#   #   # #   #   #   # #
### ### # ##### # # # # #
#   # #   # #   # # # # #
```

```
# ### ### # # ### ### # #
# #   #   # # #   #   # #
# # # # ### # ### # ### #
# # # # #   #   #   #   #
# # ### # ##### ##### ###
# #       #   #   #   # #
# ######### # ### # ### #
#           #     #     #
#############################
```

Solved Maze:

```
#############################
#.#         #     .....#
#.### ##### # #####.###.#
#...#   # #   #...#...#.#
###.### # #####.#.# #.#.#
#...# #   # #...#.# #.#.#
#.### ### # #.###.###.#.#
#.#   #   # #.#  .#...#.#
#.# # # ### #.###.#.###.#
#.# # # #   #...#...#...#
#.# ### # #####.#####.###
#.#       #...#...#...# #
#.##########.#.###.#.### #
#..........#.....#.....#
#############################
```