# TASK 1:

def heuristic(node, goal):

    x1, y1 = node

    x2, y2 = goal

    return abs(x1 - x2) + abs(y1 - y2)

def get\_neighbors(node, maze):

    x, y = node

    neighbors = [

         (x+1, y), (x-1, y), (x, y+1), (x, y-1)

    ]

    valid\_neighbors = [

        (nx, ny) for nx, ny in neighbors

        if 0 <= nx < len(maze) and 0 <= ny < len(maze[0]) and maze[nx][ny] != ' '

    ]

    return valid\_neighbors

def hill\_climbing(maze, start, goal):

    current\_node = start

    path = [current\_node]

    while current\_node != goal:

        neighbors = get\_neighbors(current\_node, maze)

        if not neighbors:

            break

        neighbor\_costs = {neighbor: heuristic(neighbor, goal) for neighbor in neighbors}

        best\_neighbor = min(neighbor\_costs, key=neighbor\_costs.get)

        if heuristic(best\_neighbor, goal) >= heuristic(current\_node, goal):

            # If no better neighbor is found, break the loop

            break

        current\_node = best\_neighbor

        path.append(current\_node)

    return path

maze = [

    [' ', ' ', 'W', ' ', 'X', 'Y'],

    ['R', 'S', 'T', 'U', ' ', 'V'],

    ['M', 'N', ' ', 'O', 'P', 'Q'],

    ['H', 'I', 'J', ' ', 'K', 'L'],

    ['F', ' ', 'G', ' ', ' ', ' '],

    ['A', ' ', 'B', 'C', 'D', 'E']

]

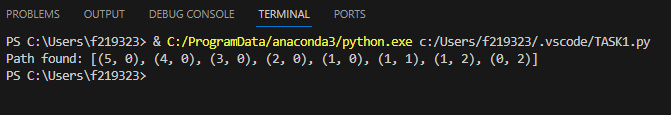
start\_position = (5, 0)

goal\_position = (0, 5)

hill\_climbing\_path = hill\_climbing(maze, start\_position, goal\_position)

print("Path found:", hill\_climbing\_path)

# OUTPUT:



# TASK 2:

import random

def objective\_function(x):

    return x\*\*2 - 4\*x + 4

def hill\_climbing(initial\_position, step\_size, max\_iterations):

    current\_position = initial\_position

    for \_ in range(max\_iterations):

        current\_value = objective\_function(current\_position)

        # Check neighboring points

        next\_positions = [current\_position - step\_size, current\_position, current\_position + step\_size]

        next\_values = [objective\_function(pos) for pos in next\_positions]

        # Move to the position with a higher (for maximum) or lower (for minimum) value

        if current\_value == max(next\_values):

            current\_position = next\_positions[next\_values.index(max(next\_values))]

        elif current\_value == min(next\_values):

            current\_position = next\_positions[next\_values.index(min(next\_values))]

        else:

            # If current position is not the maximum or minimum, break the loop

            break

    return current\_position, objective\_function(current\_position)

# Initialize random starting point within the range [-10, 10]

initial\_point = random.uniform(-10, 10)

# Set step size and maximum iterations

step\_size = 0.1

max\_iterations = 100

# For Max

max\_position, max\_value = hill\_climbing(initial\_point, step\_size, max\_iterations)

print(f"Maximum value found at position {max\_position}, with value {max\_value}")

# For Min

min\_position, min\_value = hill\_climbing(initial\_point, -step\_size, max\_iterations)

print(f"Minimum value found at position {min\_position}, with value {min\_value}")

# OUTPUT:

