CENTRAL UNIVERSITY OF HARYANA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF ENGINEERING AND TECHNOLOGY

Competitive Programming
BT CS 731

Submitted by
Rudra Chinhara, 202117
B. Tech. CSE (4th Year)

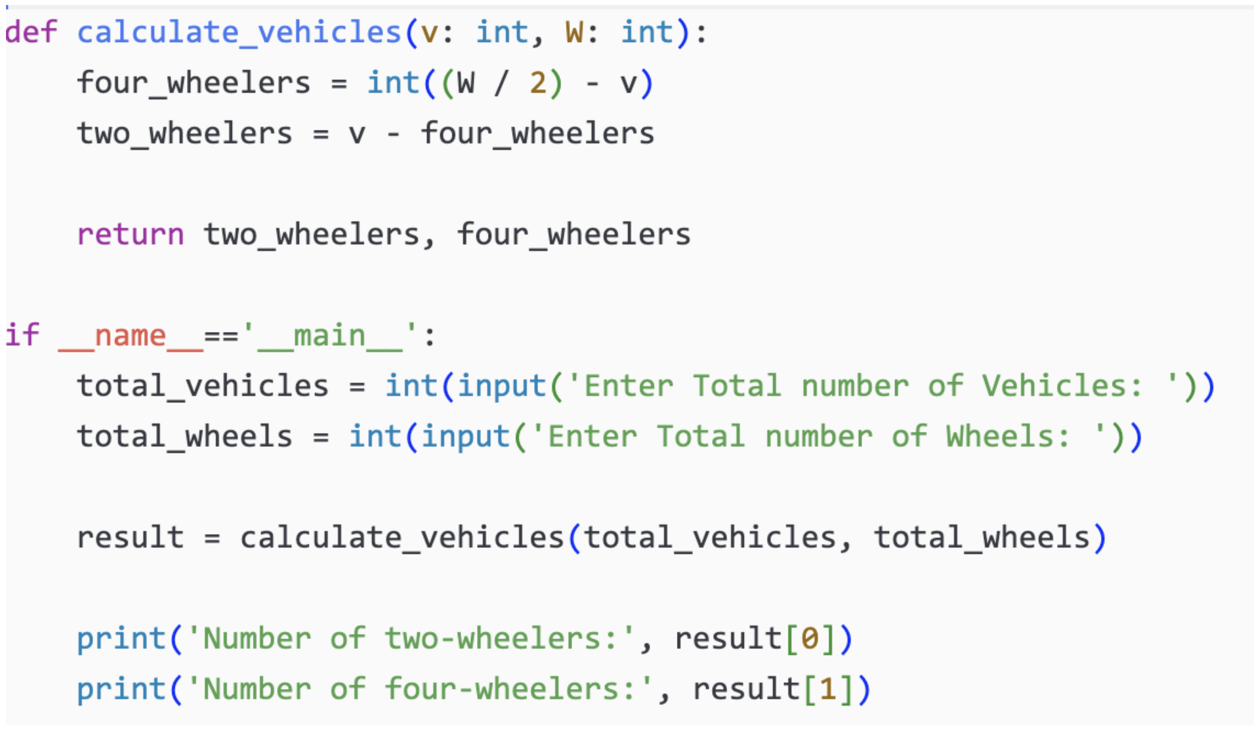
Submitted to
Ms. Sangeeta
Assistant Professor
Department of Computer Science and Engineering
School of Engineering and Technology
Central University of Haryana

Problem Statement 1

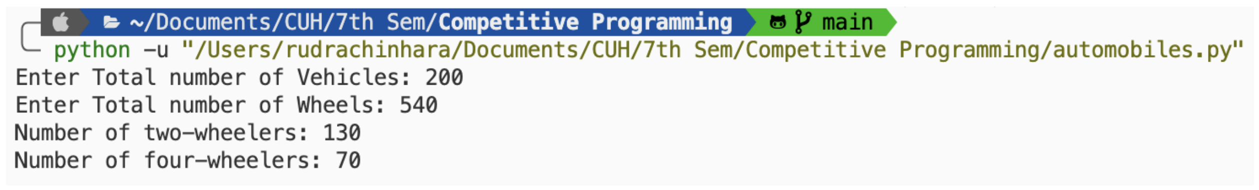
An automobile company manufactures both a two wheeler (TW) and a four wheeler (FW). A company manager wants to make the production of both types of vehicle according to the given data below:

- 1st data, Total number of vehicle (two-wheeler + four-wheeler) =v
- 2nd data, Total number of wheels = W

The task is to find how many two-wheelers as well as four-wheelers need to manufacture as per the given data.



```python
def calculate_vehicles(v: int, W: int):
    four_wheelers = int((W / 2) - v)
    two_wheelers = v - four_wheelers

    return two_wheelers, four_wheelers

if __name__=='__main__':
    total_vehicles = int(input('Enter Total number of Vehicles: '))
    total_wheels = int(input('Enter Total number of Wheels: '))

    result = calculate_vehicles(total_vehicles, total_wheels)

    print('Number of two-wheelers:', result[0])
    print('Number of four-wheelers:', result[1])
```

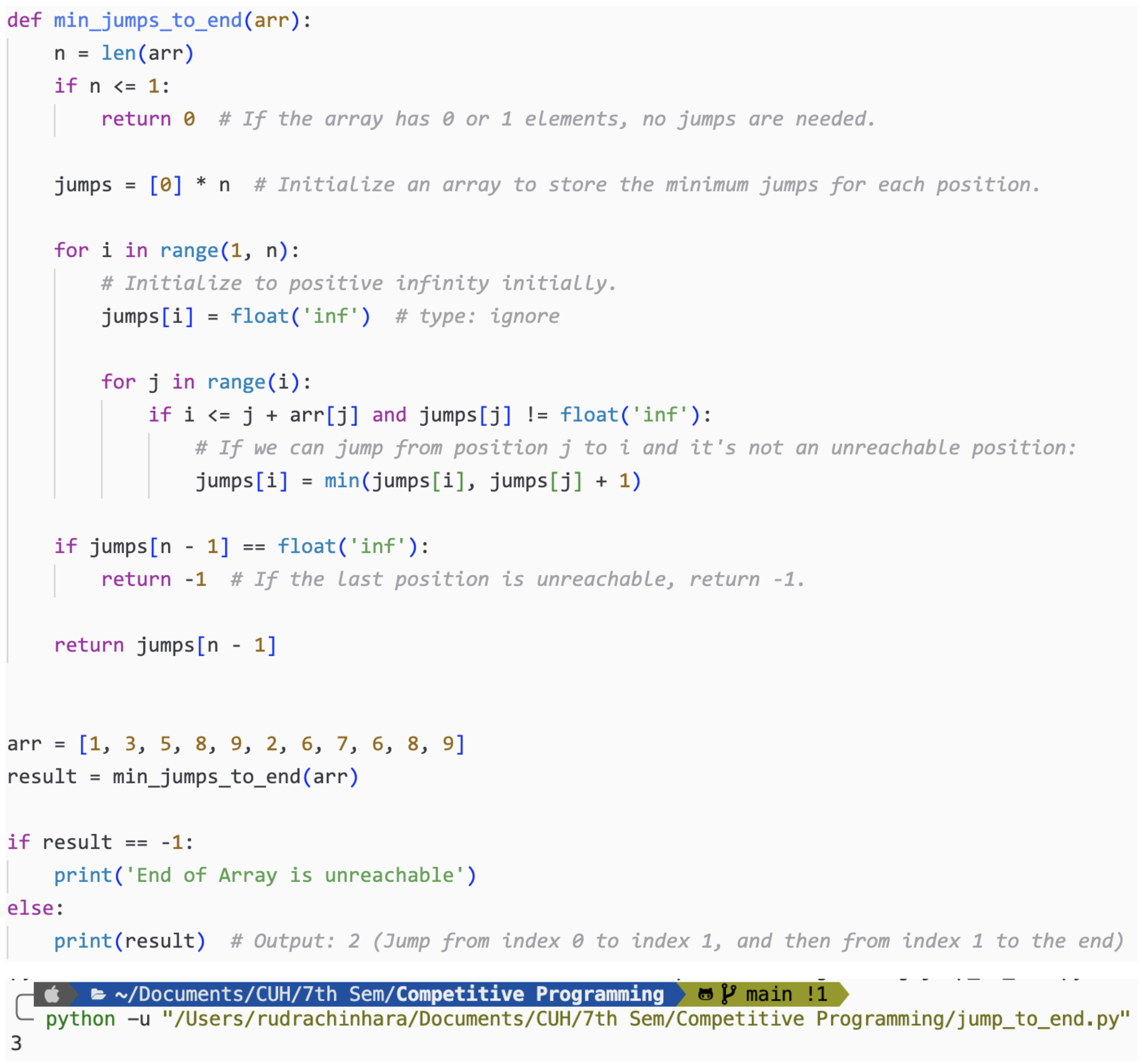


```
  ~/Documents/CUH/7th Sem/Competitive Programming   main
  python -u "/Users/rudrachinhara/Documents/CUH/7th Sem/Competitive Programming/automobiles.py"
Enter Total number of Vehicles: 200
Enter Total number of Wheels: 540
Number of two-wheelers: 130
Number of four-wheelers: 70
```

Problem Statement 2

Given an array of N integers arr [] where each element represents the maximum length of the jump that can be made forward from that element. This means if arr[i] = x, then we can jump any distance y such that y ≤ x. Find the minimum number of jumps to reach the end of the array (starting from the first element). If an element is 0, then you cannot move through that element.

Note: Return -1 if you can't reach the end of the array.



```python
def min_jumps_to_end(arr):
    n = len(arr)
    if n <= 1:
        return 0  # If the array has 0 or 1 elements, no jumps are needed.

    jumps = [0] * n  # Initialize an array to store the minimum jumps for each position.

    for i in range(1, n):
        # Initialize to positive infinity initially.
        jumps[i] = float('inf')  # type: ignore

        for j in range(i):
            if i <= j + arr[j] and jumps[j] != float('inf'):
                # If we can jump from position j to i and it's not an unreachable position:
                jumps[i] = min(jumps[i], jumps[j] + 1)

    if jumps[n - 1] == float('inf'):
        return -1  # If the last position is unreachable, return -1.

    return jumps[n - 1]


arr = [1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9]
result = min_jumps_to_end(arr)

if result == -1:
    print('End of Array is unreachable')
else:
    print(result)  # Output: 2 (Jump from index 0 to index 1, and then from index 1 to the end)
```

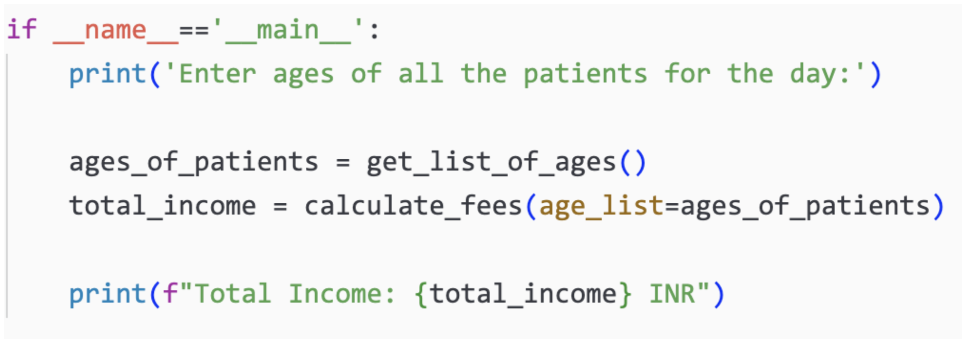
```
~/Documents/CUH/7th Sem/Competitive Programming  main !1
python -u "/Users/rudrachinhara/Documents/CUH/7th Sem/Competitive Programming/jump_to_end.py"
3
```

Problem Statement 3

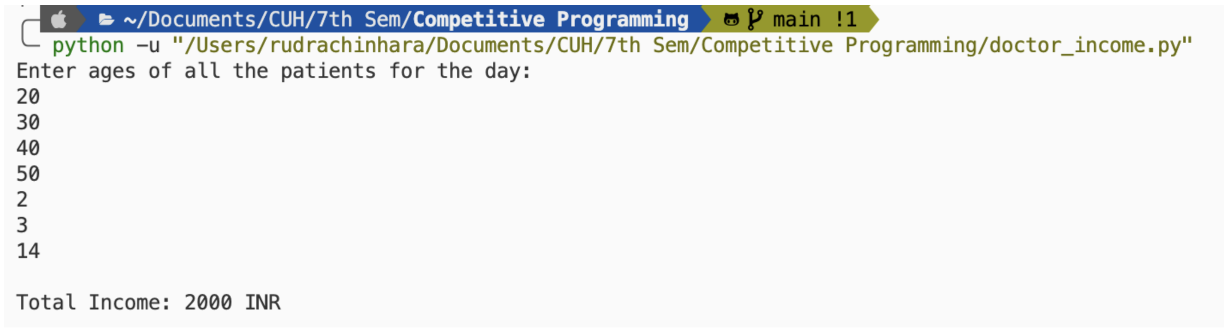
A doctor has a clinic where he serves his patients. The doctor’s consultation fees are different for different groups of patients depending on their age. If the patient’s age is below 17, fees is 200 INR. If the patient’s age is between 17 and 40, the fee is 400 INR. If the patient's age is above 40, the fee is 300 INR. Write a code to calculate earnings in a day for which one array/List of values representing the age of patients visited on that day is passed as input. Note:

- Age should not be zero or less than zero or above 120
- Doctor consults a maximum of 20 patients a day
- Enter age value (press Enter without a value to stop)

```python
def get_list_of_ages():
    age = []
    for i in range(20):
        m = input()

        try:
            if m == "":
                break
            elif int(m) in range(0,120):
                age.append(int(m))
            else:
                print("INVALID INPUT")
                continue
        except ValueError:
            print("INVALID INPUT")
            continue

    return age


def calculate_fees(age_list):
    fees = 0

    for i in age_list:
        if i < 17:
            fees+=200
        elif i < 40:
            fees+=400
        else:
            fees+=300

    return fees
```



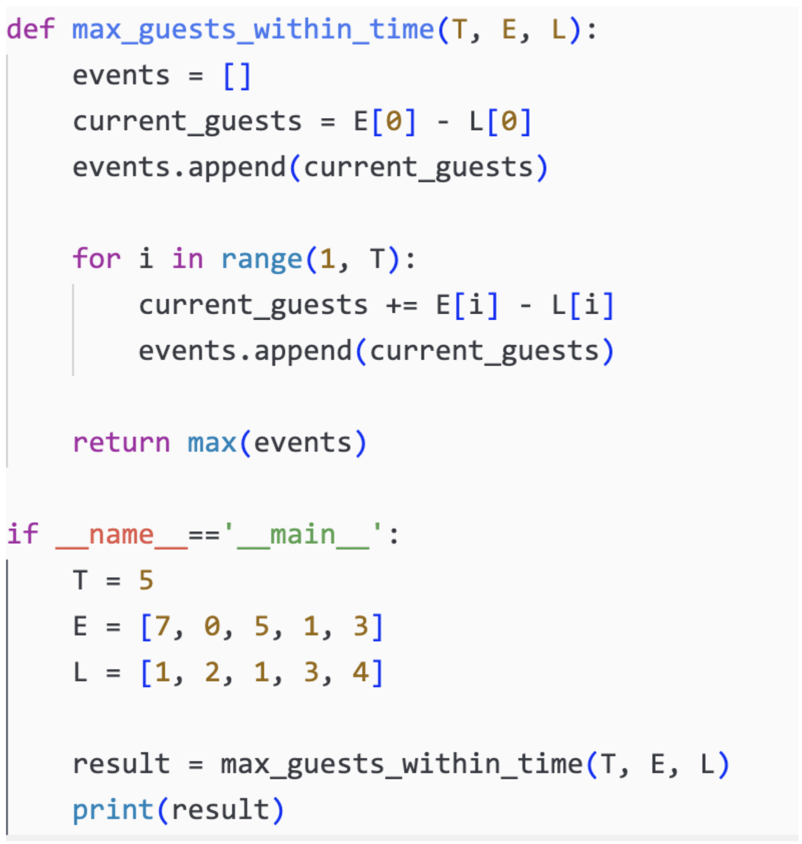
```python
if __name__=='__main__':
    print('Enter ages of all the patients for the day:')

    ages_of_patients = get_list_of_ages()
    total_income = calculate_fees(age_list=ages_of_patients)

    print(f"Total Income: {total_income} INR")
```



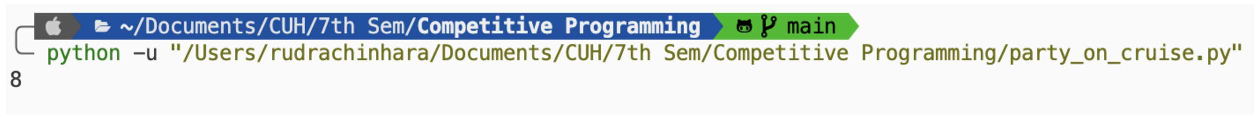
```
 ~/Documents/CUH/7th Sem/Competitive Programming  main !1
python -u "/Users/rudrachinhara/Documents/CUH/7th Sem/Competitive Programming/doctor_income.py"
Enter ages of all the patients for the day:
20
30
40
50
2
3
14

Total Income: 2000 INR
```

<u>Problem Statement 4</u>

A party has been organized on cruise. The party is organized for a limited time(T). The number of guests entering (E[i]) and leaving (L[i]) the party at every hour is represented as elements of the array. The task is to find the maximum number of guests present on the cruise at any given instance within T hours.



```python
def max_guests_within_time(T, E, L):
    events = []
    current_guests = E[0] - L[0]
    events.append(current_guests)

    for i in range(1, T):
        current_guests += E[i] - L[i]
        events.append(current_guests)

    return max(events)

if __name__=='__main__':
    T = 5
    E = [7, 0, 5, 1, 3]
    L = [1, 2, 1, 3, 4]

    result = max_guests_within_time(T, E, L)
    print(result)
```

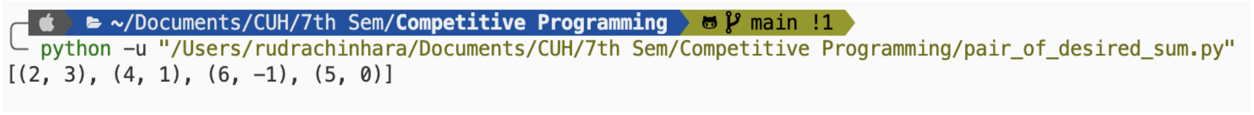


```
 ~/Documents/CUH/7th Sem/Competitive Programming   main
python -u "/Users/rudrachinhara/Documents/CUH/7th Sem/Competitive Programming/party_on_cruise.py"
8
```

Problem Statement 5

Given an array of integers that may contain both positive and negative integers. Write a program to find all the pairs of integers whose sum is equal to the desired sum.

```python
def find_pairs_with_sum(arr, target_sum):
    pair_set = set()  # A set to store seen numbers

    pairs = []  # List to store pairs that sum to the desired value

    for num in arr:
        # Calculate the complement needed for the current number to reach the target sum
        complement = target_sum - num

        # Check if the complement exists in the dictionary
        if complement in pair_set:
            # If it does, add the pair to the result list
            pairs.append((complement, num))

        # Add the current number to the dictionary with a dummy value
        pair_set.add(num)

    return pairs

arr = [2, 4, 3, 1, 5, 6, -1, 0, 9]
target_sum = 5

result = find_pairs_with_sum(arr, target_sum)
print(result)
```



```
~/Documents/CUH/7th Sem/Competitive Programming   main !1
python -u "/Users/rudrachinhara/Documents/CUH/7th Sem/Competitive Programming/pair_of_desired_sum.py"
[(2, 3), (4, 1), (6, -1), (5, 0)]
```

<u>Problem Statement 6</u>

Write a program for PostOrder Traversal without Recursion.

```python
class TreeNode:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None
```

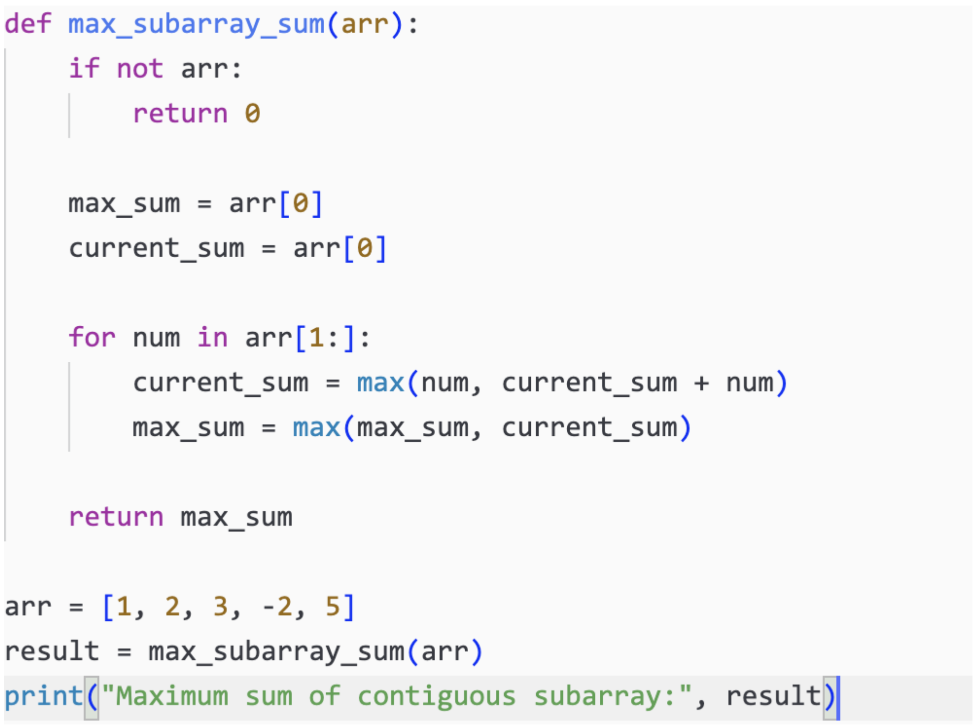
```python
def post_order_traversal(root):
    if root is None:
        return []

    result = []
    stack = []
    current = root
    visited = set()

    while stack or current:
        if current and current not in visited:
            # Traverse left subtree
            stack.append(current)
            current = current.left
        elif stack:
            # Traverse right subtree
            if stack[-1].right and stack[-1].right not in visited:
                current = stack[-1].right
            else:
                visited.add(stack[-1])
                result.append(stack.pop().value)
        else:
            current = None

    return result
```

```python
# Example usage:
# Construct a sample binary tree:
#         1
#        / \
#       2   3
#      / \
#     4   5
root = TreeNode(1)
root.left = TreeNode(2) # type: ignore
root.right = TreeNode(3) # type: ignore
root.left.left = TreeNode(4) # type: ignore
root.left.right = TreeNode(5) # type: ignore

result = post_order_traversal(root)
print(result)
```

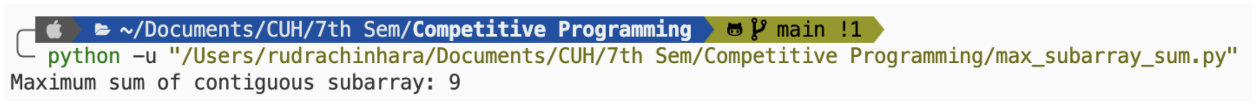
```
~/Documents/CUH/7th Sem/Competitive Programming  main !1
python -u "/Users/rudrachinhara/Documents/CUH/7th Sem/Competitive Programming/post_order_traversal.py"
[4, 5, 2, 3, 1]
```

<u>Problem Statement 7</u>

Given an array Arr[] of N integers. Find the contiguous sub-array (containing at least one number) which has the maximum sum and return its sum.



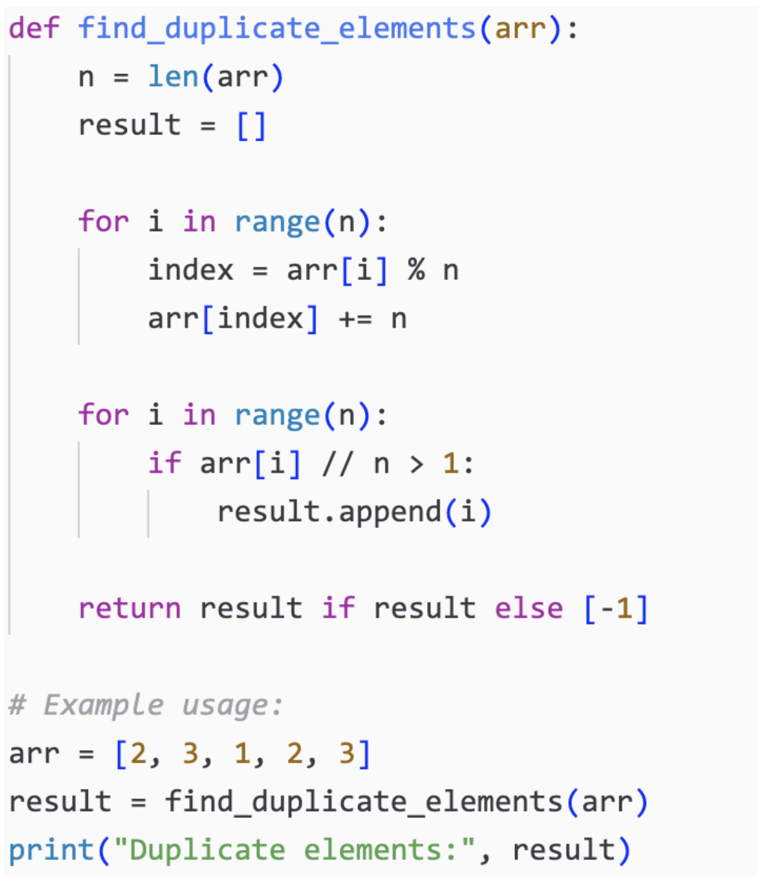
```python
def max_subarray_sum(arr):
    if not arr:
        return 0

    max_sum = arr[0]
    current_sum = arr[0]

    for num in arr[1:]:
        current_sum = max(num, current_sum + num)
        max_sum = max(max_sum, current_sum)

    return max_sum

arr = [1, 2, 3, -2, 5]
result = max_subarray_sum(arr)
print("Maximum sum of contiguous subarray:", result)
```



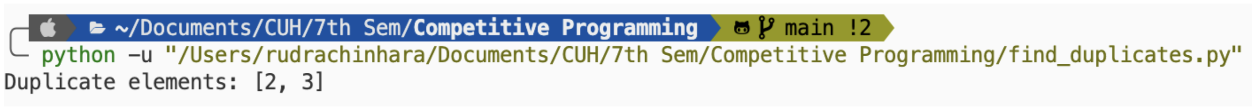
```
~/Documents/CUH/7th Sem/Competitive Programming  main !1
python -u "/Users/rudrachinhara/Documents/CUH/7th Sem/Competitive Programming/max_subarray_sum.py"
Maximum sum of contiguous subarray: 9
```

<u>Problem Statement 8</u>

Given an array A of size N which contains elements from 0 to N-1, you need to find all the elements occurring more than once in the given array. Return the answer in ascending order. If no such element is found, return list containing [-1]. Note: The extra space is only for the array to be returned. Try to perform all operations within the provided array.



```python
def find_duplicate_elements(arr):
    n = len(arr)
    result = []

    for i in range(n):
        index = arr[i] % n
        arr[index] += n

    for i in range(n):
        if arr[i] // n > 1:
            result.append(i)

    return result if result else [-1]

# Example usage:
arr = [2, 3, 1, 2, 3]
result = find_duplicate_elements(arr)
print("Duplicate elements:", result)
```

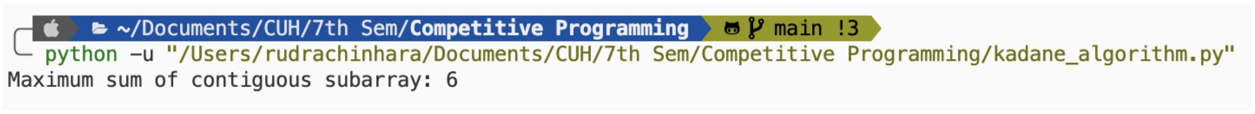


```
╭─  ~/Documents/CUH/7th Sem/Competitive Programming   main !2
╰─ python -u "/Users/rudrachinhara/Documents/CUH/7th Sem/Competitive Programming/find_duplicates.py"
Duplicate elements: [2, 3]
```

Problem Statement 9

Write a program explaining Kadane’s Algorithm.

```python
def kadanes_algorithm(arr):
    if not arr:
        return 0

    max_sum = arr[0]
    current_sum = arr[0]

    for num in arr[1:]:
        current_sum = max(num, current_sum + num)
        max_sum = max(max_sum, current_sum)

    return max_sum

arr = [-2, 1, -3, 4, -1, 2, 1, -5, 4]
result = kadanes_algorithm(arr)
print("Maximum sum of contiguous subarray:", result)
```



```
╭─  ~/Documents/CUH/7th Sem/Competitive Programming   main !3
╰─ python -u "/Users/rudrachinhara/Documents/CUH/7th Sem/Competitive Programming/kadane_algorithm.py"
Maximum sum of contiguous subarray: 6
```

Problem Statement 10

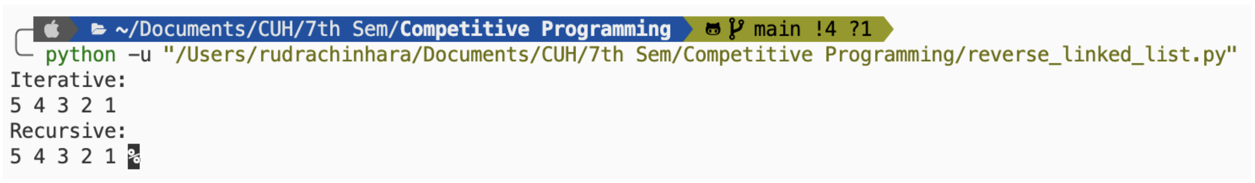
Write a Program to reverse the Linked List. (Both iterative and recursive).

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

def reverse_linked_list_iterative(head):
    prev = None
    current = head

    while current is not None:
        next_node = current.next
        current.next = prev
        prev = current
        current = next_node

    return prev
```

```python
def reverse_linked_list_recursive(head):
    if head is None or head.next is None:
        return head

    # Recursively reverse the rest of the list
    rest_reversed = reverse_linked_list_recursive(head.next)

    # Change the next of the current node
    head.next.next = head
    head.next = None

    # The new head is the last node of the reversed list
    return rest_reversed
```

```python
# Example Usage:
# Assume you have a linked list with nodes: 1 -> 2 -> 3 -> 4 -> 5
# Construct the linked list
head = Node(1)
head.next = Node(2)
head.next.next = Node(3)
head.next.next.next = Node(4)
head.next.next.next.next = Node(5)

# Reverse the linked list(Iterative)
print("Iterative: ")
new_head = reverse_linked_list_iterative(head)

# Print the reversed linked list
while new_head is not None:
    print(new_head.data, end=" ")
    new_head = new_head.next
```

```python
# Reverse the linked list(Recursive)
print("Recursive: ")
new_head = reverse_linked_list_recursive(head)

# Print the reversed linked list
while new_head is not None:
    print(new_head.data, end=" ")
    new_head = new_head.next
```

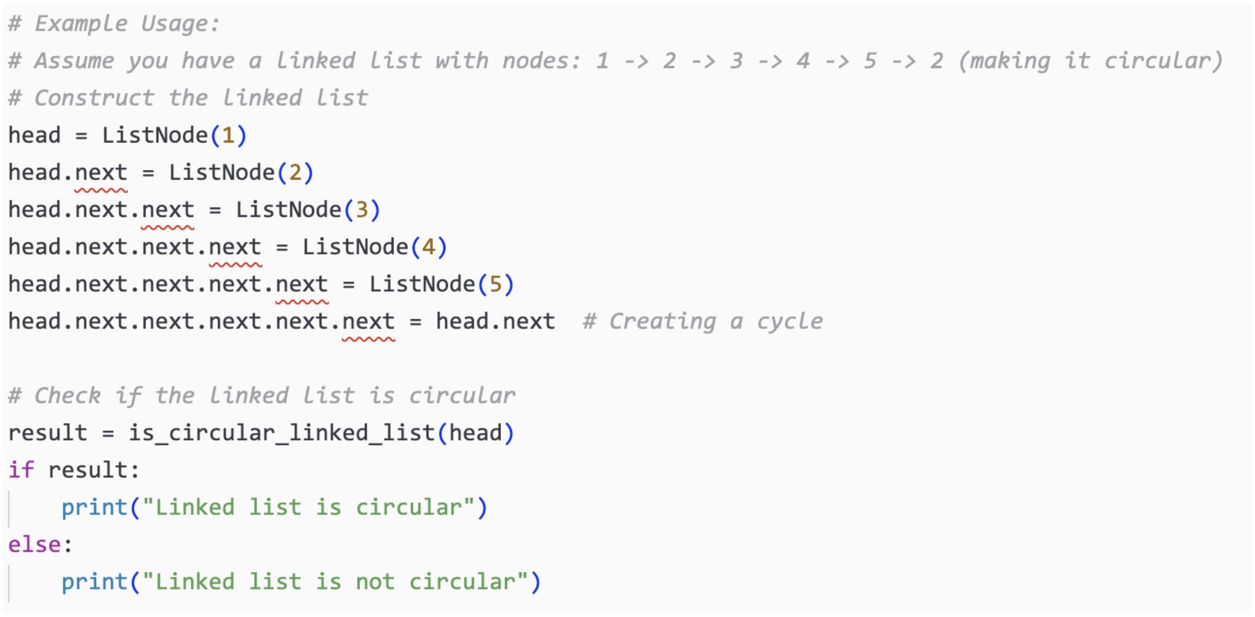


```
 ~/Documents/CUH/7th Sem/Competitive Programming  main !4 ?1
python -u "/Users/rudrachinhara/Documents/CUH/7th Sem/Competitive Programming/reverse_linked_list.py"
Iterative:
5 4 3 2 1
Recursive:
5 4 3 2 1 %
```

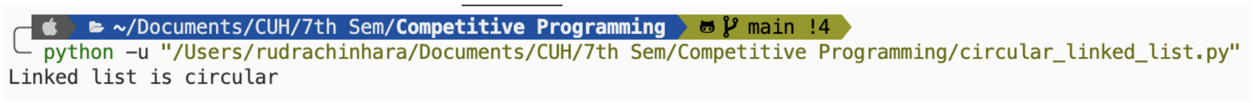
Problem Statement 11

Given head, the head of a singly linked list, find if the linked list is circular or not. A linked list is called circular if it is not NULL terminated and all nodes are connected in the form of a cycle. An empty linked list is considered circular.

```python
class ListNode:
    def __init__(self, value):
        self.value = value
        self.next = None

def is_circular_linked_list(head):
    if not head:
        return True  # An empty linked list is considered circular

    slow_pointer = head
    fast_pointer = head

    while fast_pointer and fast_pointer.next:
        slow_pointer = slow_pointer.next
        fast_pointer = fast_pointer.next.next

        if slow_pointer == fast_pointer:
            return True  # There is a cycle in the linked list

    return False  # No cycle found
```



```python
# Example Usage:
# Assume you have a linked list with nodes: 1 -> 2 -> 3 -> 4 -> 5 -> 2 (making it circular)
# Construct the linked list
head = ListNode(1)
head.next = ListNode(2)
head.next.next = ListNode(3)
head.next.next.next = ListNode(4)
head.next.next.next.next = ListNode(5)
head.next.next.next.next.next = head.next  # Creating a cycle

# Check if the linked list is circular
result = is_circular_linked_list(head)
if result:
    print("Linked list is circular")
else:
    print("Linked list is not circular")
```

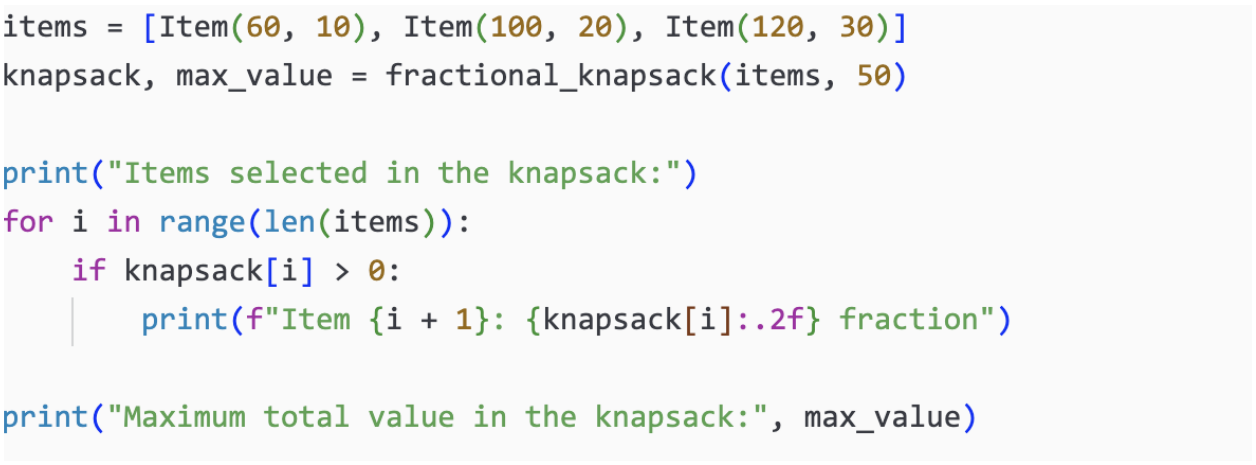


```
 ~/Documents/CUH/7th Sem/Competitive Programming   main !4
python -u "/Users/rudrachinhara/Documents/CUH/7th Sem/Competitive Programming/circular_linked_list.py"
Linked list is circular
```

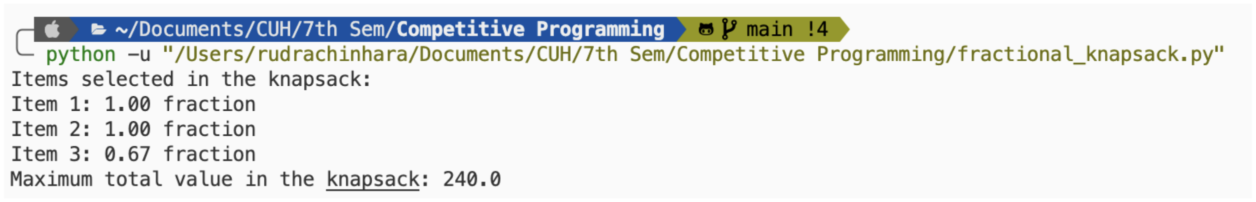
Problem Statement 12

Given weights and values of N items, we need to put these items in a knapsack of capacity W to get the maximum total value in the knapsack.
Note: Unlike 0/1 knapsack, you are allowed to break the item.

```python
class Item:
    def __init__(self, value, weight):
        self.value = value
        self.weight = weight
        self.value_per_weight = value / weight

def fractional_knapsack(items, capacity):
    # Sort items based on value-to-weight ratio in descending order
    items.sort(key=lambda x: x.value_per_weight, reverse=True)

    total_value = 0
    knapsack = [0] * len(items)

    for i in range(len(items)):
        if capacity == 0:
            break

        if items[i].weight <= capacity:
            knapsack[i] = 1  # Take the whole item
            total_value += items[i].value
            capacity -= items[i].weight
        else:
            fraction = capacity / items[i].weight
            knapsack[i] = fraction  # Take a fraction of the item
            total_value += items[i].value * fraction
            capacity = 0

    return knapsack, total_value
```



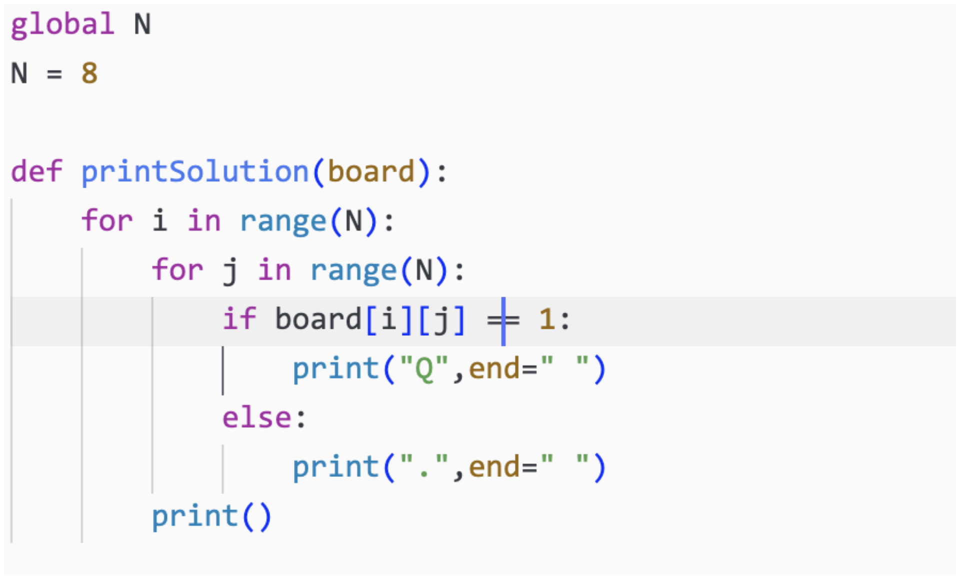
```python
items = [Item(60, 10), Item(100, 20), Item(120, 30)]
knapsack, max_value = fractional_knapsack(items, 50)

print("Items selected in the knapsack:")
for i in range(len(items)):
    if knapsack[i] > 0:
        print(f"Item {i + 1}: {knapsack[i]:.2f} fraction")

print("Maximum total value in the knapsack:", max_value)
```



```
 ~/Documents/CUH/7th Sem/Competitive Programming   main !4
python -u "/Users/rudrachinhara/Documents/CUH/7th Sem/Competitive Programming/fractional_knapsack.py"
Items selected in the knapsack:
Item 1: 1.00 fraction
Item 2: 1.00 fraction
Item 3: 0.67 fraction
Maximum total value in the knapsack: 240.0
```

Problem Statement 13

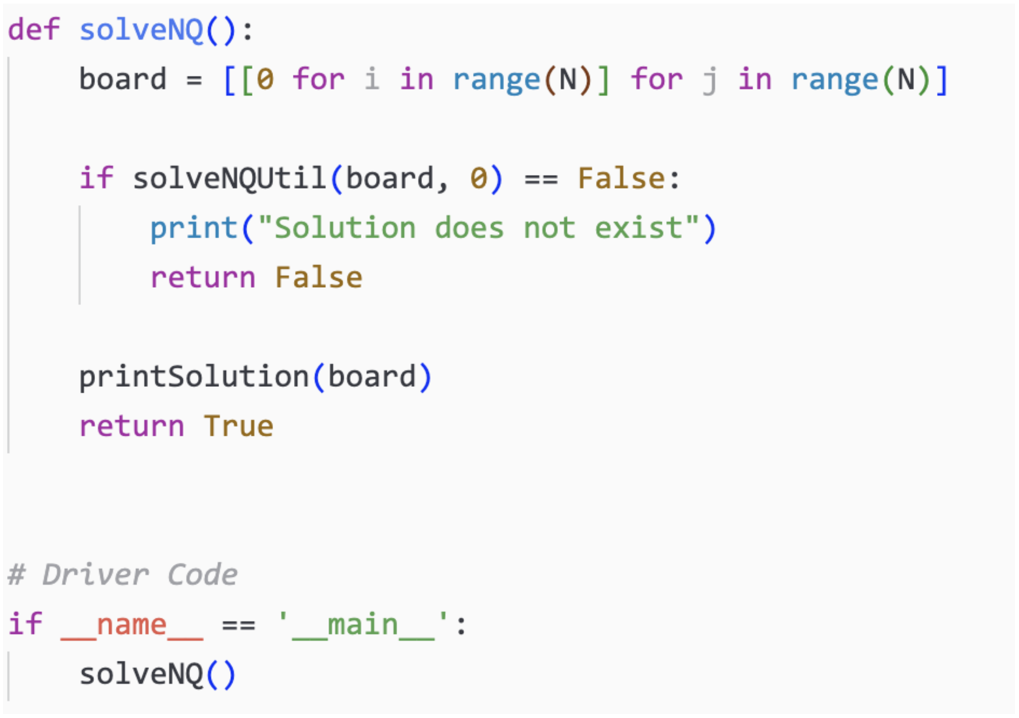
Solve N-Queens Problem. The n-queens puzzle is the problem of placing n queens on an n x n chessboard such that no two queens attack each other.



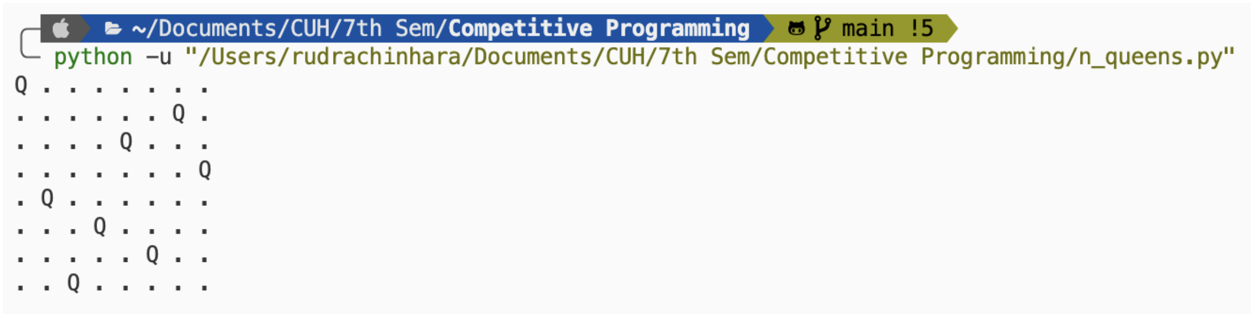
```python
global N
N = 8

def printSolution(board):
    for i in range(N):
        for j in range(N):
            if board[i][j] == 1:
                print("Q",end=" ")
            else:
                print(".",end=" ")
        print()
```

```python
def isSafe(board, row, col):

    # Check this row on left side
    for i in range(col):
        if board[row][i] == 1:
            return False

    # Check upper diagonal on left side
    for i, j in zip(range(row, -1, -1),
                    range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    # Check lower diagonal on left side
    for i, j in zip(range(row, N, 1),
                    range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    return True
```

```python
def solveNQUtil(board, col):

    # Base case: If all queens are placed
    # then return true
    if col >= N:
        return True

    # Consider this column and try placing
    # this queen in all rows one by one
    for i in range(N):

        if isSafe(board, i, col):

            # Place this queen in board[i][col]
            board[i][col] = 1

            # Recur to place rest of the queens
            if solveNQUtil(board, col + 1) == True:
                return True

            # If placing queen in board[i][col
            # doesn't lead to a solution, then
            # queen from board[i][col]
            board[i][col] = 0

    # If the queen can not be placed in any row in
    # this column col then return false
    return False
```



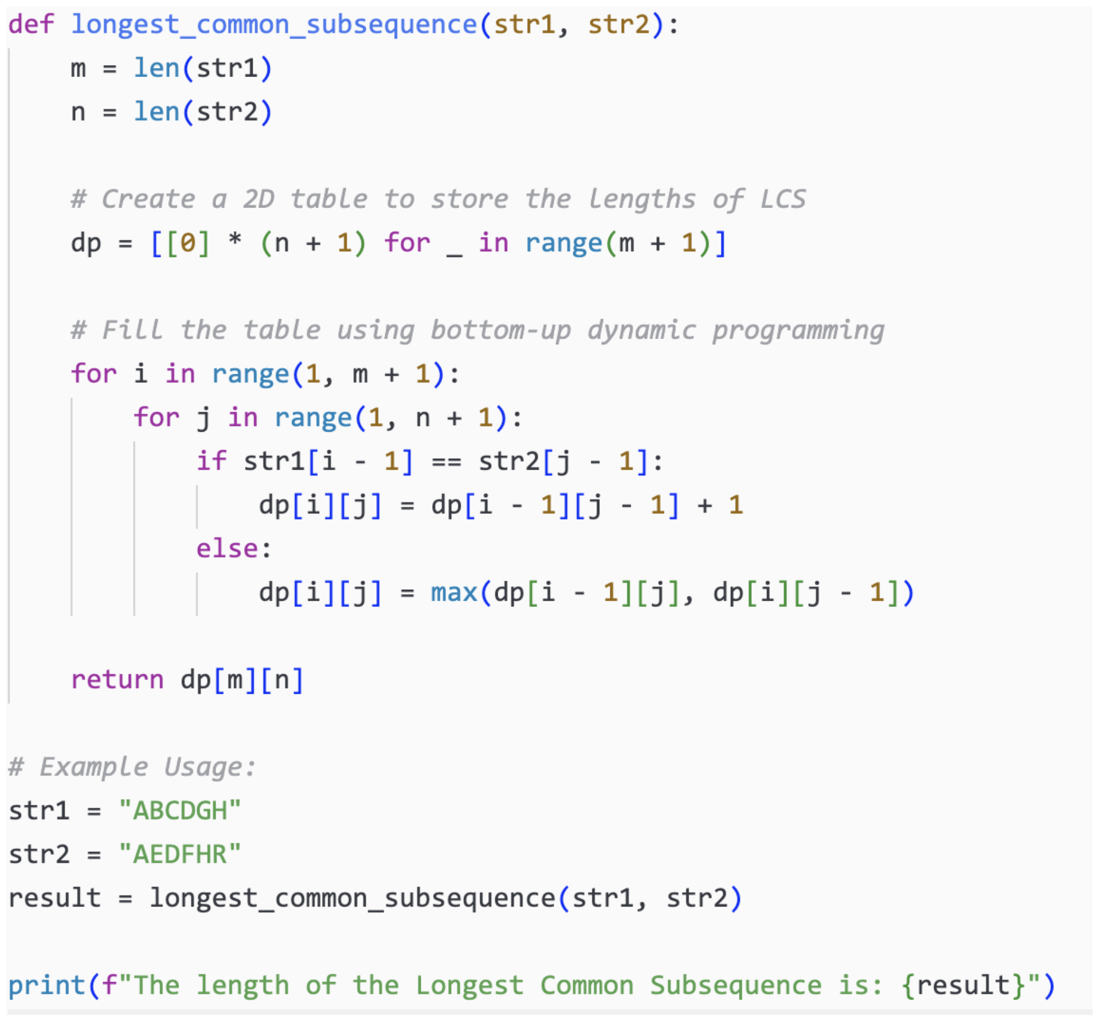
```python
def solveNQ():
    board = [[0 for i in range(N)] for j in range(N)]

    if solveNQUtil(board, 0) == False:
        print("Solution does not exist")
        return False

    printSolution(board)
    return True


# Driver Code
if __name__ == '__main__':
    solveNQ()
```



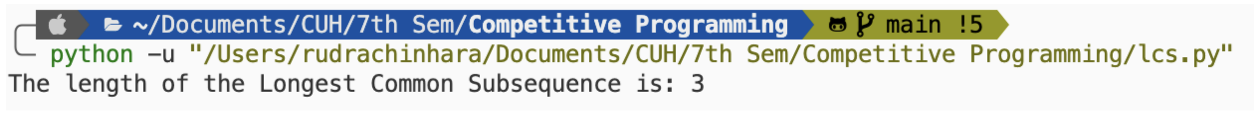
```
~/Documents/CUH/7th Sem/Competitive Programming   main !5
python -u "/Users/rudrachinhara/Documents/CUH/7th Sem/Competitive Programming/n_queens.py"
Q . . . . . . .
. . . . . . Q .
. . . . Q . . .
. . . . . . . Q
. Q . . . . . .
. . . Q . . . .
. . . . . Q . .
. . Q . . . . .
```

Problem Statement 14

Given two strings, find the length of the longest subsequence present in both of them. Both the strings are in uppercase Latin alphabets.



```python
def longest_common_subsequence(str1, str2):
    m = len(str1)
    n = len(str2)

    # Create a 2D table to store the lengths of LCS
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    # Fill the table using bottom-up dynamic programming
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if str1[i - 1] == str2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1] + 1
            else:
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

    return dp[m][n]

# Example Usage:
str1 = "ABCDGH"
str2 = "AEDFHR"
result = longest_common_subsequence(str1, str2)

print(f"The length of the Longest Common Subsequence is: {result}")
```



```
  ~/Documents/CUH/7th Sem/Competitive Programming   main !5
└ python -u "/Users/rudrachinhara/Documents/CUH/7th Sem/Competitive Programming/lcs.py"
The length of the Longest Common Subsequence is: 3
```

Problem Statement 15

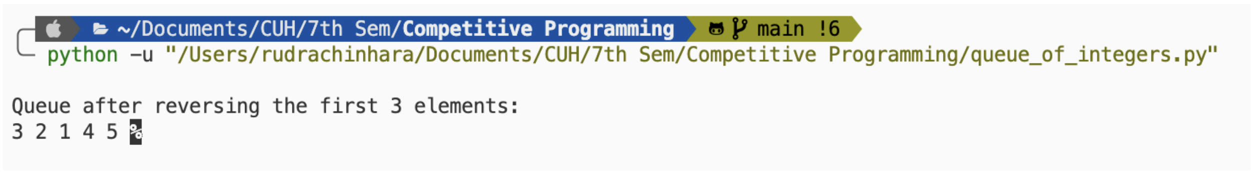
Given an integer K and a queue of integers, we need to reverse the order of the first K elements of the queue, leaving the other elements in the same relative order. Only following standard operations are allowed on queue.

- enqueue(x) : Add an item x to rear of queue
- dequeue() : Remove an item from front of queue
- size() : Returns the number of elements in the queue.
- front() : Finds front item.

Note: The above operations represent the general processing. In-built functions of the respective languages can be used to solve the problem.

```python
from queue import Queue

def reverse_k_elements(queue, k):
    if k < 0 or k > queue.qsize():
        print("Invalid value of K")
        return

    stack = []

    # Dequeue the first K elements and push them onto the stack
    for _ in range(k):
        stack.append(queue.get())

    # Enqueue the elements from the stack back to the queue
    while stack:
        queue.put(stack.pop())

    # Enqueue the remaining elements (after K) back to the queue
    for _ in range(queue.qsize() - k):
        queue.put(queue.get())

# Example Usage:
k = 3
my_queue = Queue()

# Enqueue elements to the queue
for i in range(1, 6):
    my_queue.put(i)

print("Original Queue:")
while not my_queue.empty():
    print(my_queue.get(), end=" ")

# Reverse the order of the first K elements
reverse_k_elements(my_queue, k)
```

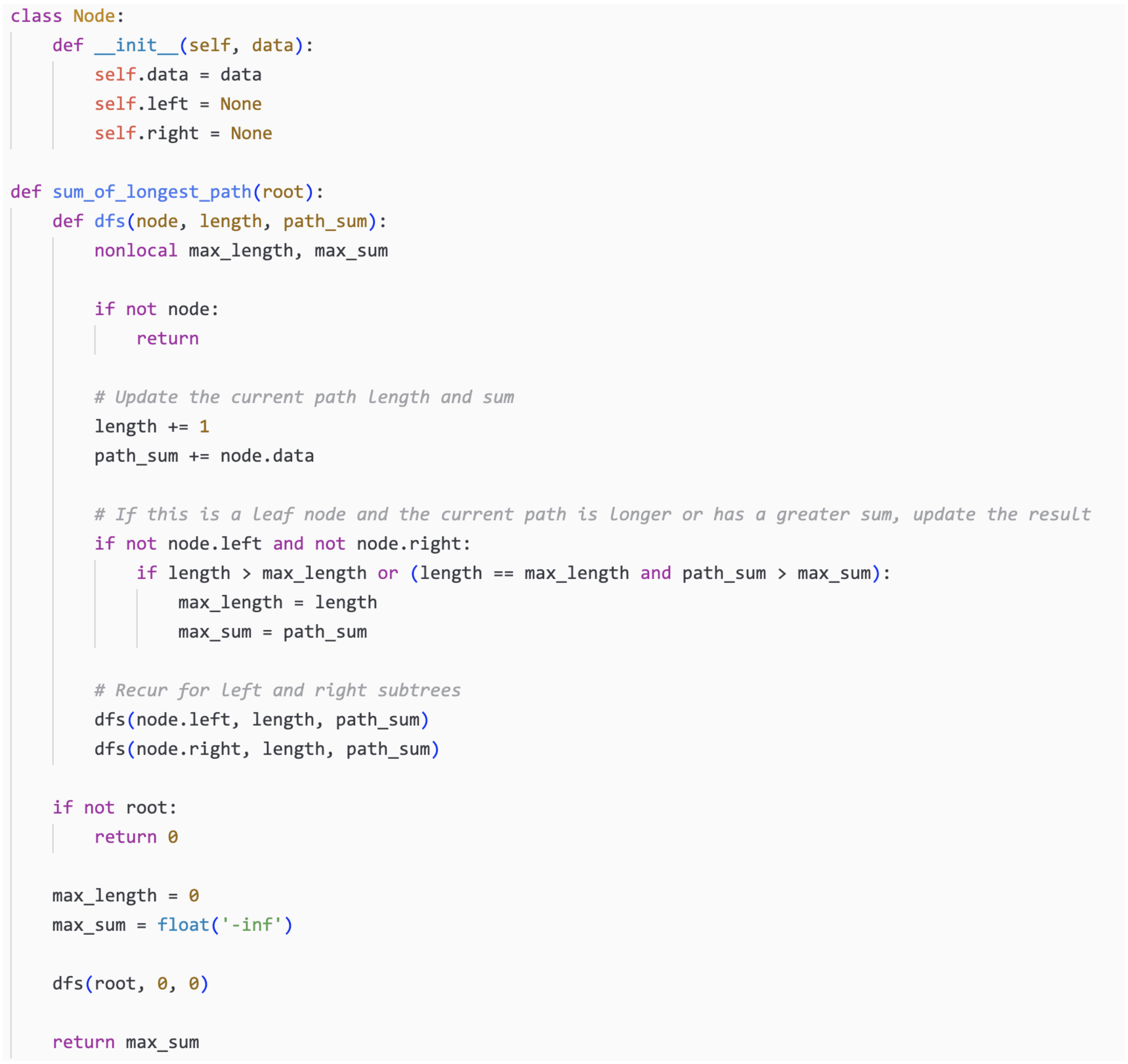
```python
print("\nQueue after reversing the first", k, "elements:")
while not my_queue.empty():
    print(my_queue.get(), end=" ")
```



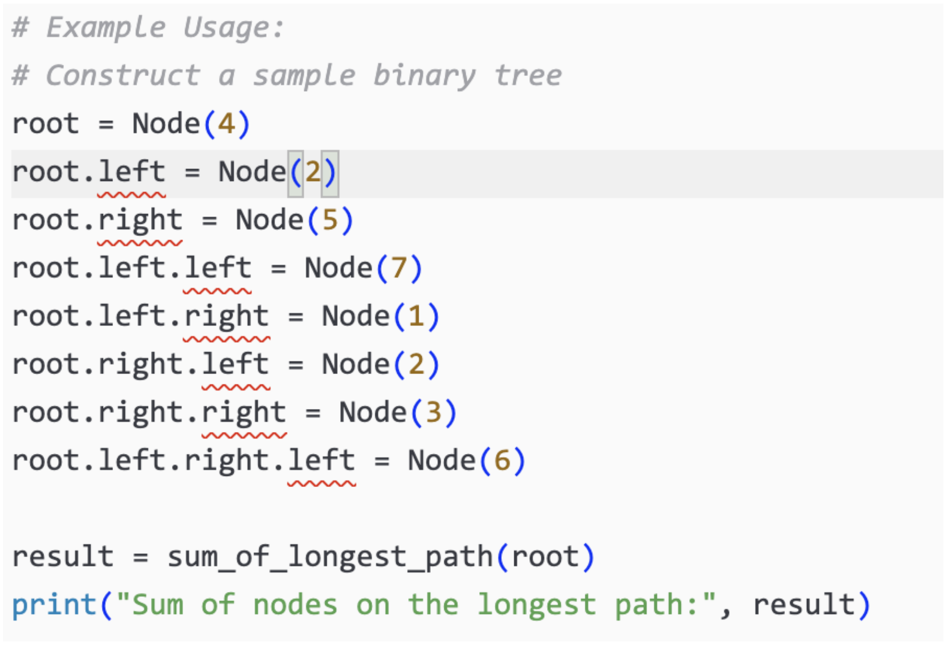
```
 ~/Documents/CUH/7th Sem/Competitive Programming  main !6
python -u "/Users/rudrachinhara/Documents/CUH/7th Sem/Competitive Programming/queue_of_integers.py"

Queue after reversing the first 3 elements:
3 2 1 4 5 %
```

Problem Statement 16

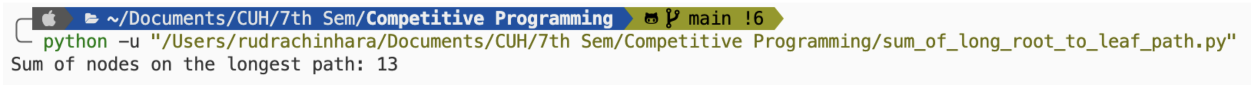
Given a binary tree of size N. Your task is to complete the function sumOfLongRootToLeafPath(), that finds the sum of all nodes on the longest path from root to leaf node.
If two or more paths compete for the longest path, then the path having the maximum sum of nodes is being considered.



```python
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

def sum_of_longest_path(root):
    def dfs(node, length, path_sum):
        nonlocal max_length, max_sum

        if not node:
            return

        # Update the current path length and sum
        length += 1
        path_sum += node.data

        # If this is a leaf node and the current path is longer or has a greater sum, update the result
        if not node.left and not node.right:
            if length > max_length or (length == max_length and path_sum > max_sum):
                max_length = length
                max_sum = path_sum

        # Recur for left and right subtrees
        dfs(node.left, length, path_sum)
        dfs(node.right, length, path_sum)

    if not root:
        return 0

    max_length = 0
    max_sum = float('-inf')

    dfs(root, 0, 0)

    return max_sum
```



```python
# Example Usage:
# Construct a sample binary tree
root = Node(4)
root.left = Node(2)
root.right = Node(5)
root.left.left = Node(7)
root.left.right = Node(1)
root.right.left = Node(2)
root.right.right = Node(3)
root.left.right.left = Node(6)

result = sum_of_longest_path(root)
print("Sum of nodes on the longest path:", result)
```

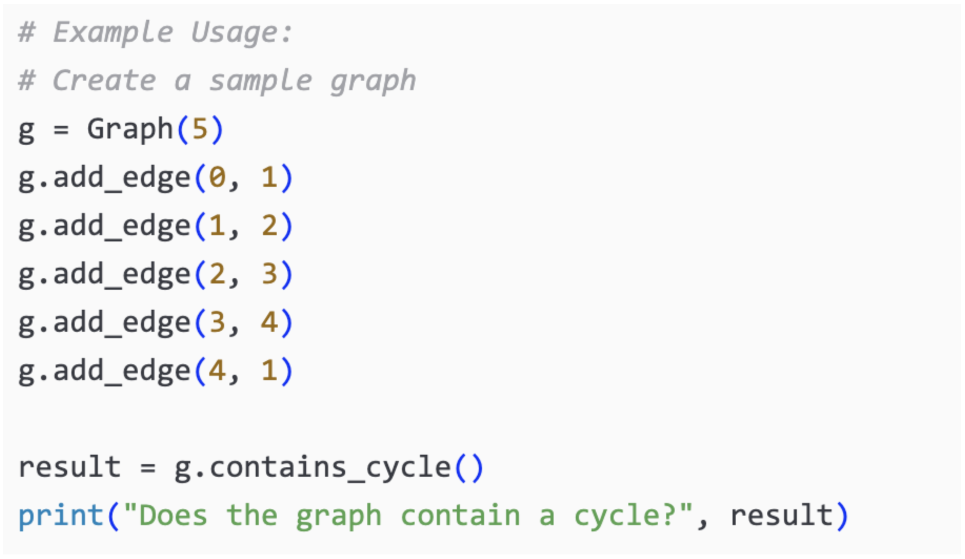


```
~/Documents/CUH/7th Sem/Competitive Programming  main !6
python -u "/Users/rudrachinhara/Documents/CUH/7th Sem/Competitive Programming/sum_of_long_root_to_leaf_path.py"
Sum of nodes on the longest path: 13
```

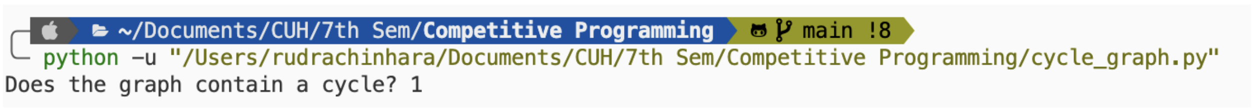
Problem Statement 17

Given an undirected graph with V vertices and E edges, check whether it contains any cycle or not. Graph is in the form of adjacency list where adj[i] contains all the nodes ith node is having edges with.
Return 1 for yes (containing cycle) otherwise 0.

```python
from collections import defaultdict

class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.graph = defaultdict(list)

    def add_edge(self, u, v):
        self.graph[u].append(v)
        self.graph[v].append(u)

    def is_cyclic_util(self, v, visited, parent):
        visited[v] = True

        for neighbor in self.graph[v]:
            if not visited[neighbor]:
                if self.is_cyclic_util(neighbor, visited, v):
                    return True
            elif parent != neighbor:
                return True

        return False

    def contains_cycle(self):
        visited = [False] * self.vertices

        for v in range(self.vertices):
            if not visited[v]:
                if self.is_cyclic_util(v, visited, -1):
                    return 1  # The graph contains a cycle

        return 0  # No cycle found
```



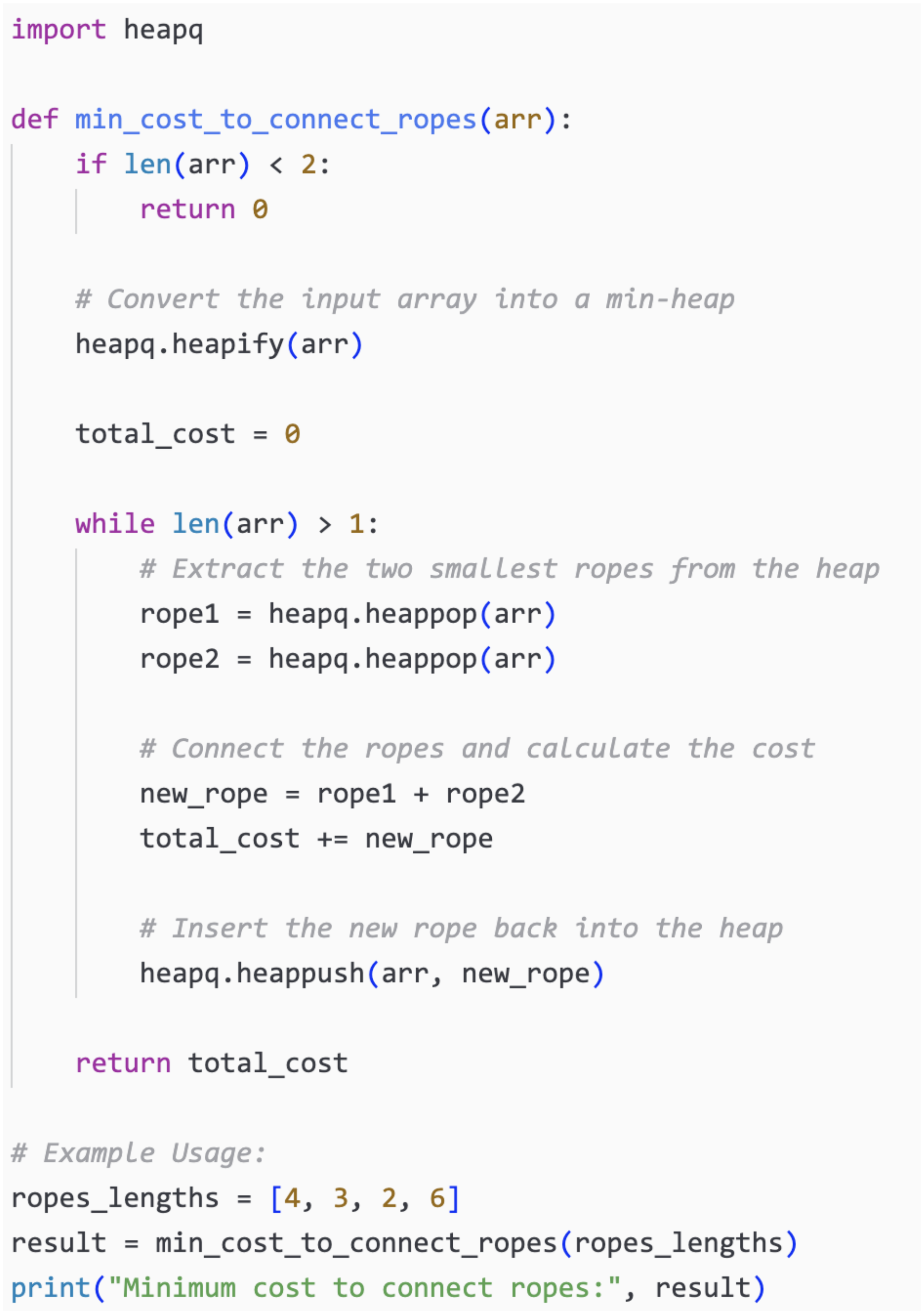
```python
# Example Usage:
# Create a sample graph
g = Graph(5)
g.add_edge(0, 1)
g.add_edge(1, 2)
g.add_edge(2, 3)
g.add_edge(3, 4)
g.add_edge(4, 1)

result = g.contains_cycle()
print("Does the graph contain a cycle?", result)
```



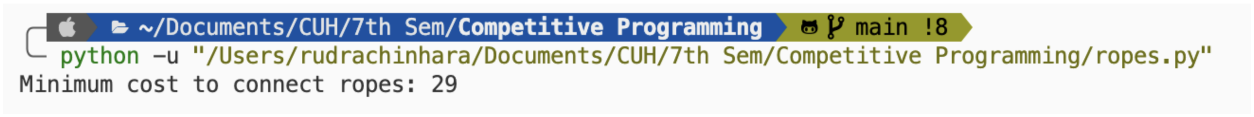
```
 ~/Documents/CUH/7th Sem/Competitive Programming  main !8
python -u "/Users/rudrachinhara/Documents/CUH/7th Sem/Competitive Programming/cycle_graph.py"
Does the graph contain a cycle? 1
```

Problem Statement 18

There are given N ropes of different lengths, we need to connect these ropes into one rope. The cost to connect two ropes is equal to the sum of their lengths. The task is to connect the ropes with minimum cost. Given an N size array arr[] contains the lengths of the ropes.



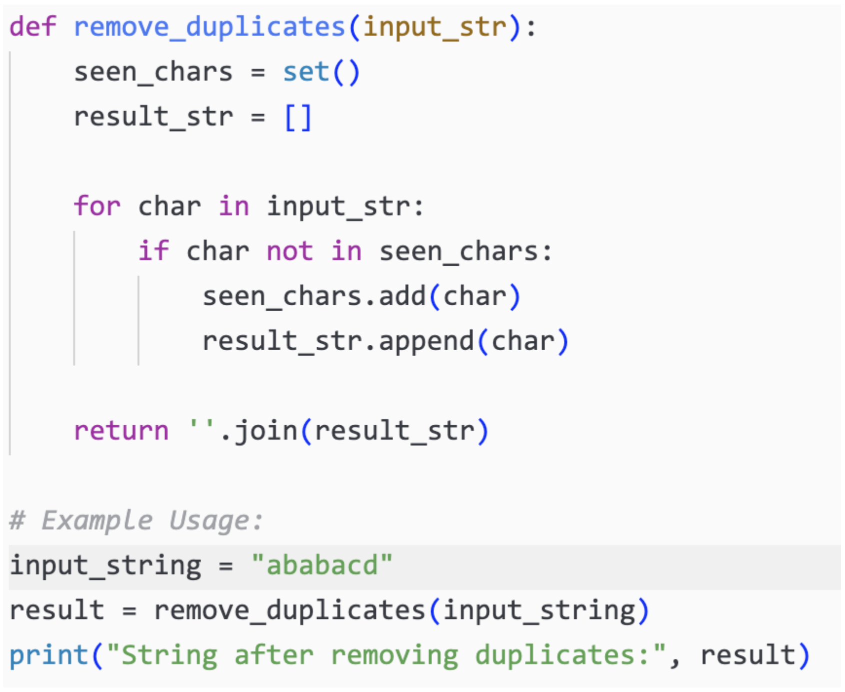
```python
import heapq

def min_cost_to_connect_ropes(arr):
    if len(arr) < 2:
        return 0

    # Convert the input array into a min-heap
    heapq.heapify(arr)

    total_cost = 0

    while len(arr) > 1:
        # Extract the two smallest ropes from the heap
        rope1 = heapq.heappop(arr)
        rope2 = heapq.heappop(arr)

        # Connect the ropes and calculate the cost
        new_rope = rope1 + rope2
        total_cost += new_rope

        # Insert the new rope back into the heap
        heapq.heappush(arr, new_rope)

    return total_cost

# Example Usage:
ropes_lengths = [4, 3, 2, 6]
result = min_cost_to_connect_ropes(ropes_lengths)
print("Minimum cost to connect ropes:", result)
```



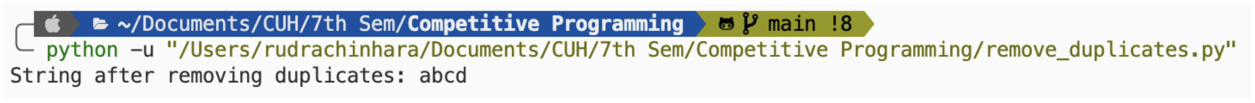
```
 ~/Documents/CUH/7th Sem/Competitive Programming   main !8
 python -u "/Users/rudrachinhara/Documents/CUH/7th Sem/Competitive Programming/ropes.py"
Minimum cost to connect ropes: 29
```

<u>Problem Statement 19</u>

You have given a string. You need to remove all the duplicates from the string. The final output string should contain each character only once. The respective order of the characters inside the string should remain the same. You can only traverse the string at once.



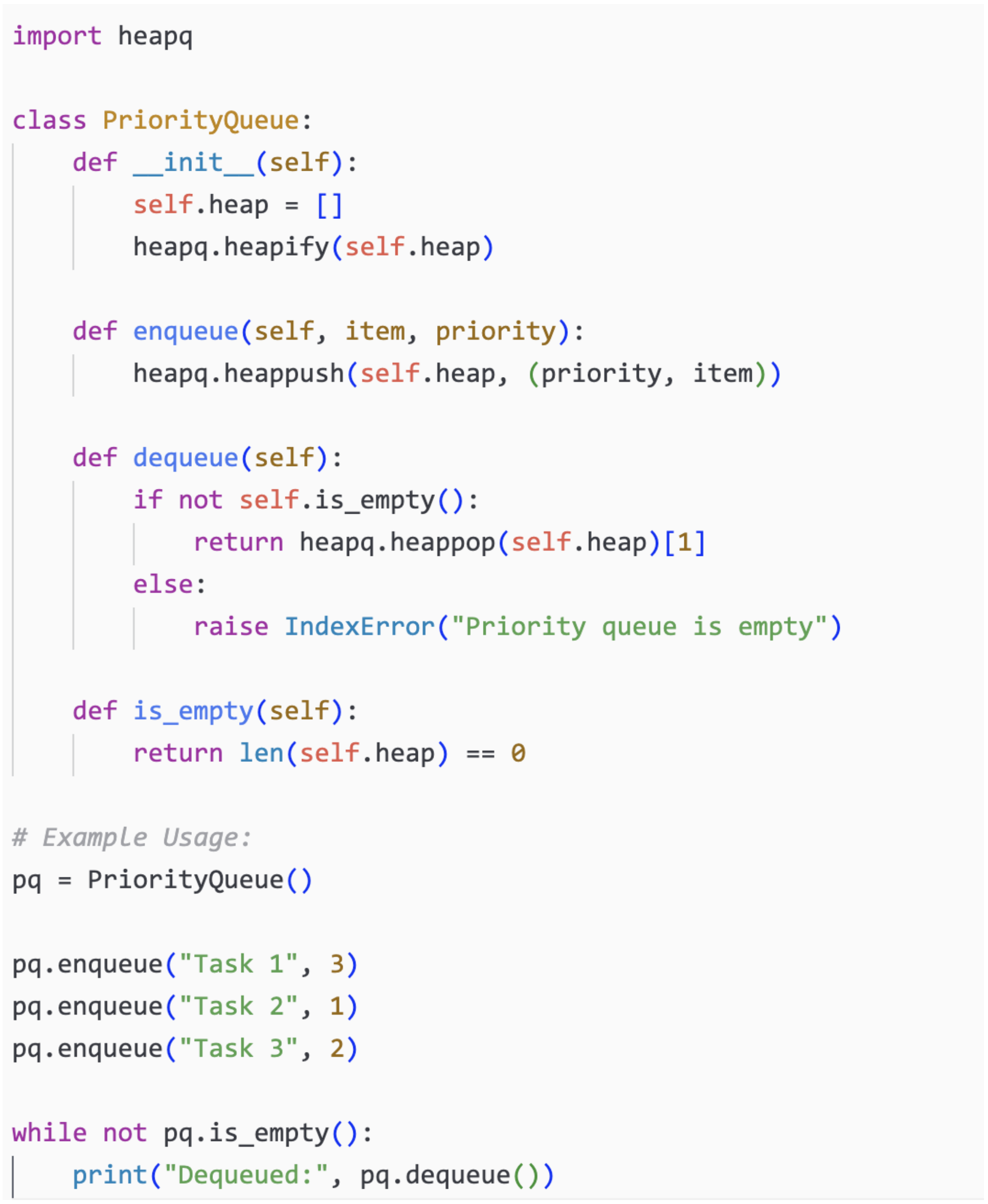
```python
def remove_duplicates(input_str):
    seen_chars = set()
    result_str = []

    for char in input_str:
        if char not in seen_chars:
            seen_chars.add(char)
            result_str.append(char)

    return ''.join(result_str)

# Example Usage:
input_string = "ababacd"
result = remove_duplicates(input_string)
print("String after removing duplicates:", result)
```



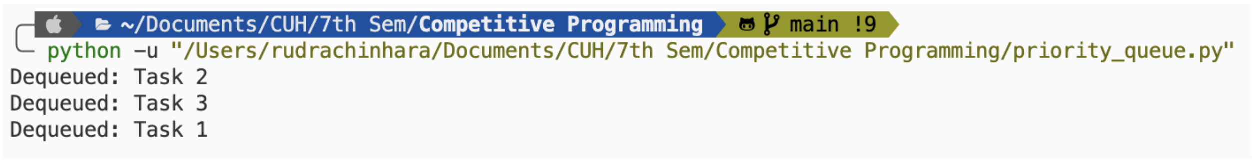
```
╭─  ~/Documents/CUH/7th Sem/Competitive Programming   main !8
╰─ python -u "/Users/rudrachinhara/Documents/CUH/7th Sem/Competitive Programming/remove_duplicates.py"
String after removing duplicates: abcd
```

Problem Statement 20

Implement Priority Queue.



```python
import heapq

class PriorityQueue:
    def __init__(self):
        self.heap = []
        heapq.heapify(self.heap)

    def enqueue(self, item, priority):
        heapq.heappush(self.heap, (priority, item))

    def dequeue(self):
        if not self.is_empty():
            return heapq.heappop(self.heap)[1]
        else:
            raise IndexError("Priority queue is empty")

    def is_empty(self):
        return len(self.heap) == 0

# Example Usage:
pq = PriorityQueue()

pq.enqueue("Task 1", 3)
pq.enqueue("Task 2", 1)
pq.enqueue("Task 3", 2)

while not pq.is_empty():
    print("Dequeued:", pq.dequeue())
```



```
~/Documents/CUH/7th Sem/Competitive Programming   main !9
python -u "/Users/rudrachinhara/Documents/CUH/7th Sem/Competitive Programming/priority_queue.py"
Dequeued: Task 2
Dequeued: Task 3
Dequeued: Task 1
```