Assignment On

‘ADVANCED DATA STRUCTURES AND ALGORITHMS’

(Assignment-2)

Submitted by: Suganth C J
Enrollment no: 2503B09901 (MCA).

Submitted to: Dr. Rahul Mishra.

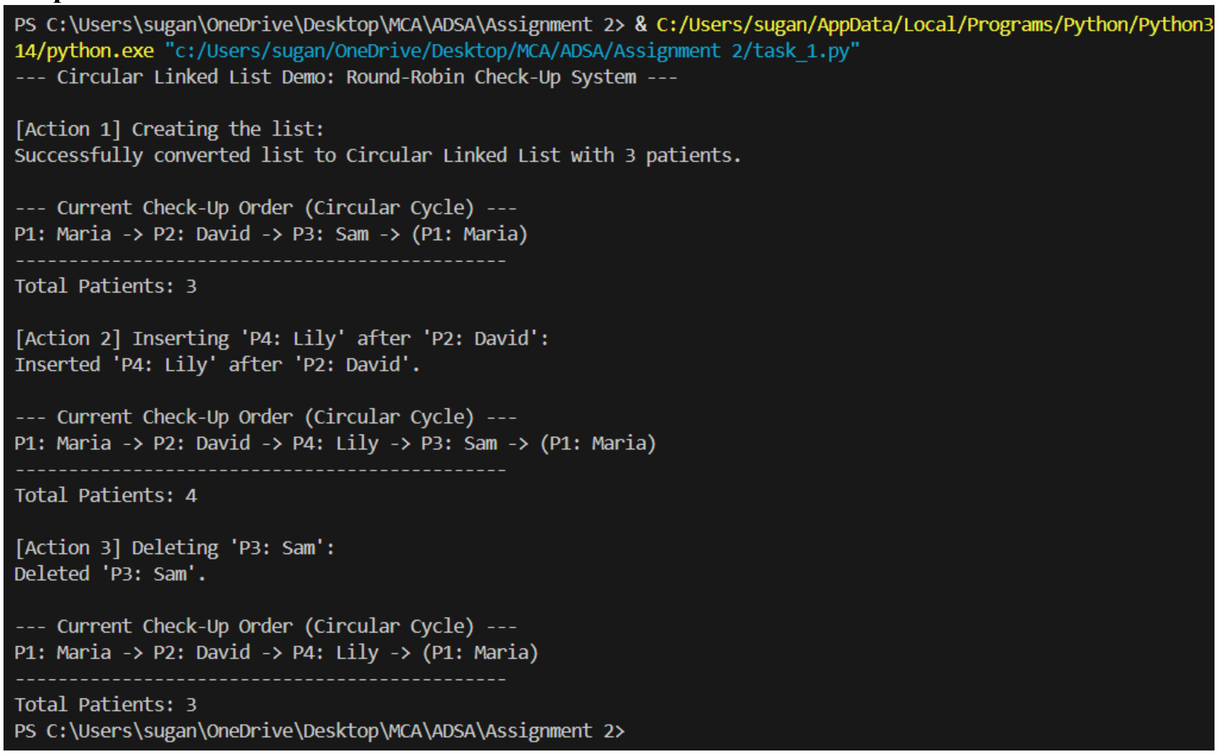
Question: Convert The Patient List to A Circular Linked List for a Round-Robin Check-Up System. Implement Insertion and Deletion.

Code:

```
import sys

# Setting a higher recursion limit for safety, although not
strictly needed for this iterative code.
# sys.setrecursionlimit(2000)

class Node:
    """A single node in the circular linked list."""
    def __init__(self, data):
        self.data = data
        self.next = None

class CircularLinkedList:
    """
    Manages the circular list of patients for a round-robin
check-up system.
    The 'head' points to the starting patient of the check-up
cycle.
    """
    def __init__(self):
        self.head = None

    def is_empty(self):
        """Checks if the list is empty."""
        return self.head is None

    def append(self, data):
        """Adds a new node (patient) to the end of the circular
list."""
        new_node = Node(data)
        if self.is_empty():
            self.head = new_node
            new_node.next = self.head  # Points to itself
            return

        # Find the last node (the one whose 'next' is 'self.head')
        last = self.head
        while last.next is not self.head:
            last = last.next

        last.next = new_node
        new_node.next = self.head # New node points back to
the head

    def create_from_list(self, patient_list):
        """Converts a standard Python list of patients into the
circular linked list."""
        if not patient_list:
            print("Input list is empty. List not created.")
            return

        # Clear existing list if any
        self.head = None

        for patient in patient_list:
            self.append(patient)
        print(f"Successfully converted list to Circular Linked
List with {len(patient_list)} patients.")

    def display(self):
        """
        Displays all patients in the circular list, highlighting the
cycle.
        """
        if self.is_empty():
            print("The patient list is empty.")
            return

        patients = []
        current = self.head

        # Traverse the list exactly once
        while True:
            patients.append(current.data)
            current = current.next
            if current is self.head:
                break

        # Format the output to show the circular nature
        print("\n--- Current Check-Up Order (Circular Cycle)
---")
        print(" -> ".join(patients) + f" -> ({self.head.data})")
        print("--------------------------------------------")
        print(f"Total Patients: {len(patients)}")

    def insert_after(self, target_data, new_data):
        """
        Inserts a new patient immediately after a specific
existing patient.
        Returns True if successful, False otherwise.
        """
        if self.is_empty():
            print("Cannot insert: The list is empty.")
            return False

        current = self.head
        # Traverse until we find the target or loop back to the
head
        while True:
            if current.data == target_data:
                new_node = Node(new_data)
                new_node.next = current.next
                current.next = new_node
                print(f"Inserted '{new_data}' after
'{target_data}'.")
                return True

            current = current.next
            if current is self.head:
                break # End of one full cycle

        print(f"Insertion failed: Target patient '{target_data}'
not found.")
        return False

    def delete(self, data_to_delete):
        """
        Deletes the first occurrence of a patient with the given
data.
        Handles deletion of the head node as well as the only
node.
        """
        if self.is_empty():
            print("Deletion failed: The list is empty.")
            return False

        current = self.head
        prev = None

        # Traverse to find the node and its predecessor
        while True:
            if current.data == data_to_delete:
```

```
            break # Found the node

          prev = current
          current = current.next

          if current is self.head:
              print(f"Deletion failed: Patient '{data_to_delete}'
not found in the list.")
             return False # Completed full loop without finding
it

      # Case 1: Only one node in the list
      if current is self.head and current.next is self.head:
          self.head = None
            print(f"Deleted '{data_to_delete}'. The list is now
empty.")
          return True

      # Case 2: Deleting the head node (not the only node)
      if current is self.head:
          # Find the last node (the one pointing to head)
          last = self.head
          while last.next is not self.head:
              last = last.next

            self.head = current.next # Move head to the next
patient
          last.next = self.head    # Last node now points to the
new head
         print(f"Deleted '{data_to_delete}' (Head). New head
is '{self.head.data}'.")
          return True

      # Case 3: Deleting a node in the middle or end
      # 'prev' is guaranteed to exist here
      prev.next = current.next
      print(f"Deleted '{data_to_delete}'.")
      return True

# --- Simple Example and Demonstration ---
if __name__ == '__main__':
    print("--- Circular Linked List Demo: Round-Robin
Check-Up System ---")

  cll = CircularLinkedList()
  initial_patients = ["P1: Maria", "P2: David", "P3: Sam"]

  # 1. Create the list from a standard patient list
  print("\n[Action 1] Creating the list:")
  cll.create_from_list(initial_patients)
  cll.display()

  # 2. Insert a new patient
  print("\n[Action 2] Inserting 'P4: Lily' after 'P2: David':")
  cll.insert_after("P2: David", "P4: Lily")
  cll.display()

  # 3. Delete a patient
  print("\n[Action 3] Deleting 'P3: Sam':")
  cll.delete("P3: Sam")
  cll.display()
```

Output:



```
PS C:\Users\sugan\OneDrive\Desktop\MCA\ADSA\Assignment 2> & C:/Users/sugan/AppData/Local/Programs/Python/Python3
14/python.exe "c:/Users/sugan/OneDrive/Desktop/MCA/ADSA/Assignment 2/task_1.py"
--- Circular Linked List Demo: Round-Robin Check-Up System ---

[Action 1] Creating the list:
Successfully converted list to Circular Linked List with 3 patients.

--- Current Check-Up Order (Circular Cycle) ---
P1: Maria -> P2: David -> P3: Sam -> (P1: Maria)
-----------------------------------------------
Total Patients: 3

[Action 2] Inserting 'P4: Lily' after 'P2: David':
Inserted 'P4: Lily' after 'P2: David'.

--- Current Check-Up Order (Circular Cycle) ---
P1: Maria -> P2: David -> P4: Lily -> P3: Sam -> (P1: Maria)
-----------------------------------------------
Total Patients: 4

[Action 3] Deleting 'P3: Sam':
Deleted 'P3: Sam'.

--- Current Check-Up Order (Circular Cycle) ---
P1: Maria -> P2: David -> P4: Lily -> (P1: Maria)
-----------------------------------------------
Total Patients: 3
PS C:\Users\sugan\OneDrive\Desktop\MCA\ADSA\Assignment 2>
```