class Node:

    def \_\_init\_\_(self, data):

        self.data = data

        self.next = None

class CircularLinkedList:

    def \_\_init\_\_(self):

        self.tail = None

    # This function insert a node after tail (start)

    # If there is node at start then it becomes the 2nd node

    # Time Complexity is O(1)

    def insert\_at\_beginning(self,data):

        new\_node = Node(data)

        # Empty List

        if (self.tail == None):

            self.tail = new\_node

            new\_node.next = new\_node

            return

        # Single node and  Multi Node

        new\_node.next = self.tail.next

        self.tail.next = new\_node

    # Time Complexity is O(1)

    def insert\_at\_end\_tail(self, data):

        new\_node = Node(data)

        # case 1: Empty List

        if (self.tail == None):

            self.tail = new\_node

            new\_node.next = new\_node

            return

        # case 2: single node and Multi Node

        new\_node.next = self.tail.next

        self.tail.next = new\_node

        self.tail = new\_node

    def delete\_at\_beginning(self):

        if (self.tail == None):

            print("List is Empty")

            return

        # case 1: if single node list will empty

        if (self.tail.next == self.tail):

            self.tail = None

            return

        # case 2: two or More Nodes

        self.tail.next = self.tail.next.next

    # Node before tail becomes te new tail

    # since we don't have address of node before tail node

    # we have to traverse to each there

    def delete\_at\_end\_tail(self):

        # Empty List

        if(self.tail == None):

            return

        # case 2: single node

        if self.tail.next == self.tail:

            self.tail = None

            return

        # case 3: two or more nodes

        # find the reference to the previous to the tail node

        new\_tail = self.tail.next

        while (new\_tail.next != self.tail):

            new\_tail = new\_tail.next

        new\_tail.next =self.tail.next

        self.tail = new\_tail

    def print\_all\_nodes(self):

        # case 1: List is empty

        if (self.tail == None):

            print("List is Empty")

            return

        current\_node = self.tail.next

        while (True):

            print(f"{current\_node.data} --> ", end=" ")

            if (current\_node == self.tail):

                break

            current\_node = current\_node.next

        print(f"Tail ({current\_node.data}) point to first node data ({self.tail.next.data})")

    def search\_key(self, key):

        # case 1: List is empty

        if (self.tail == None):

            print("List is Empty")

            return

        current\_node = self.tail.next

        while (True):

            if (current\_node.data == key):

                print(f"{key} is found in list ✅")

                return

            if current\_node == self.tail:

                break

            current\_node= current\_node.next

        print(f"{key} is not present in list ❌")

def Circular\_Linked\_List\_tests(clist: CircularLinkedList):

    # List is emptyand trying to delete a node

    clist.delete\_at\_beginning()

    clist.delete\_at\_end\_tail()

    clist.print\_all\_nodes()

    clist.search\_key(40)

    # perform insert and delete single node

    clist.insert\_at\_beginning(40)

    clist.delete\_at\_beginning()

    clist.print\_all\_nodes()

    clist.search\_key(40)

    # perform insert at tail and delete at tail

    clist.insert\_at\_end\_tail(50)

    clist.delete\_at\_end\_tail()

    # create list with multiple node

    clist.insert\_at\_beginning(10)

    clist.print\_all\_nodes()

    clist.search\_key(40)

    clist.insert\_at\_beginning(20)

    clist.insert\_at\_beginning(30)

    clist. insert\_at\_end\_tail(40)

    clist.print\_all\_nodes()

    clist.search\_key(40)

    clist.search\_key(100)

if \_\_name\_\_ == "\_\_main\_\_":

    clist = CircularLinkedList()

    Circular\_Linked\_List\_tests(clist)