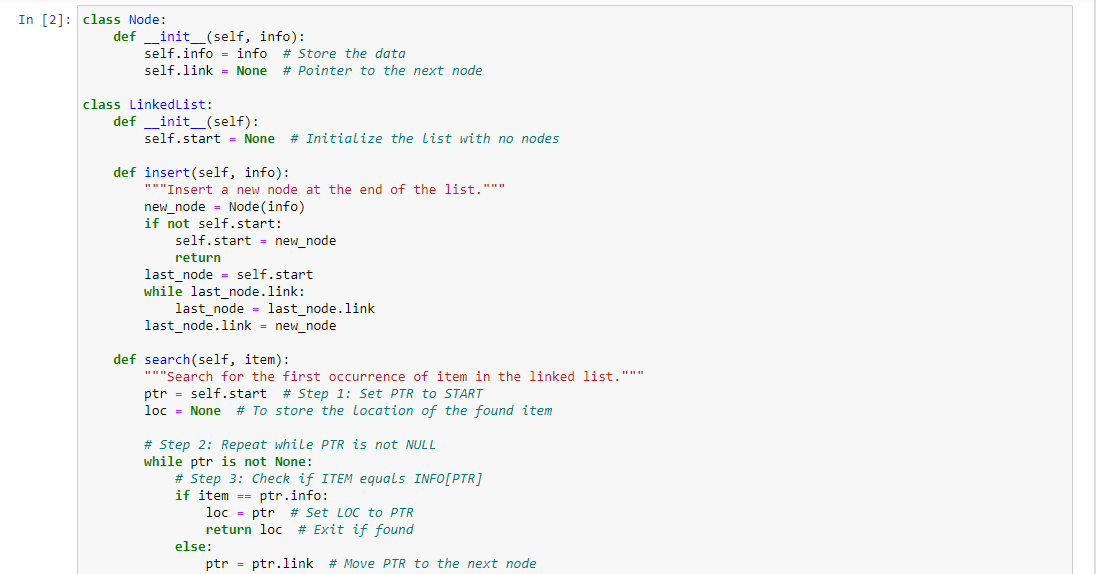
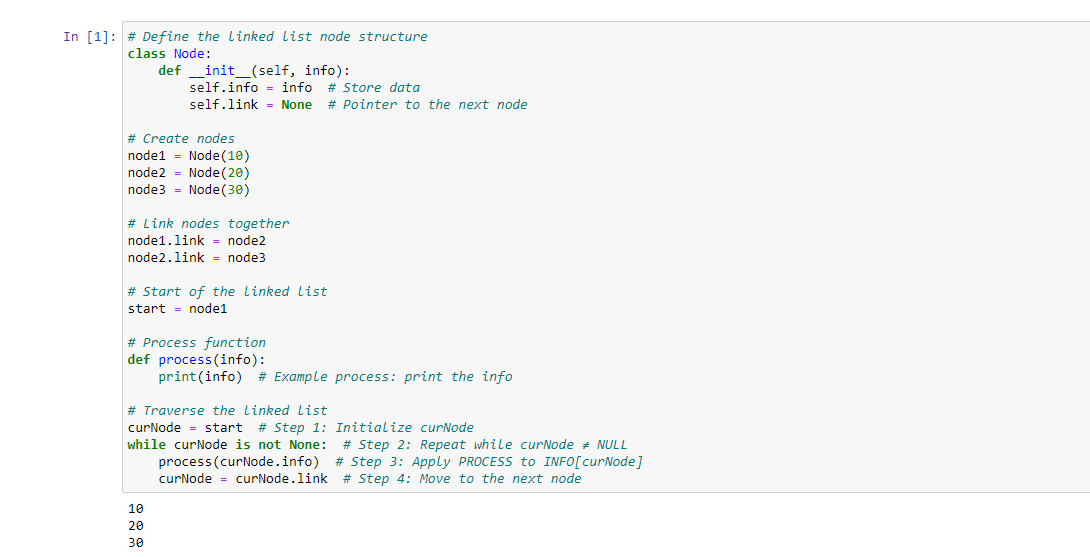
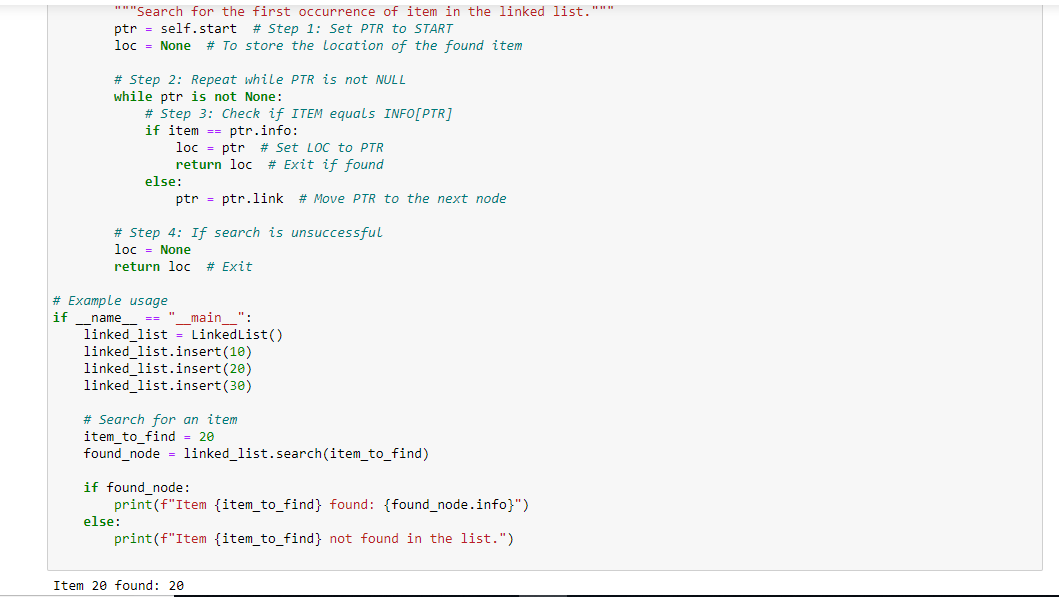
**Exercise**

1. Implement the above algorithms. Use proper indentation and comments in your code.

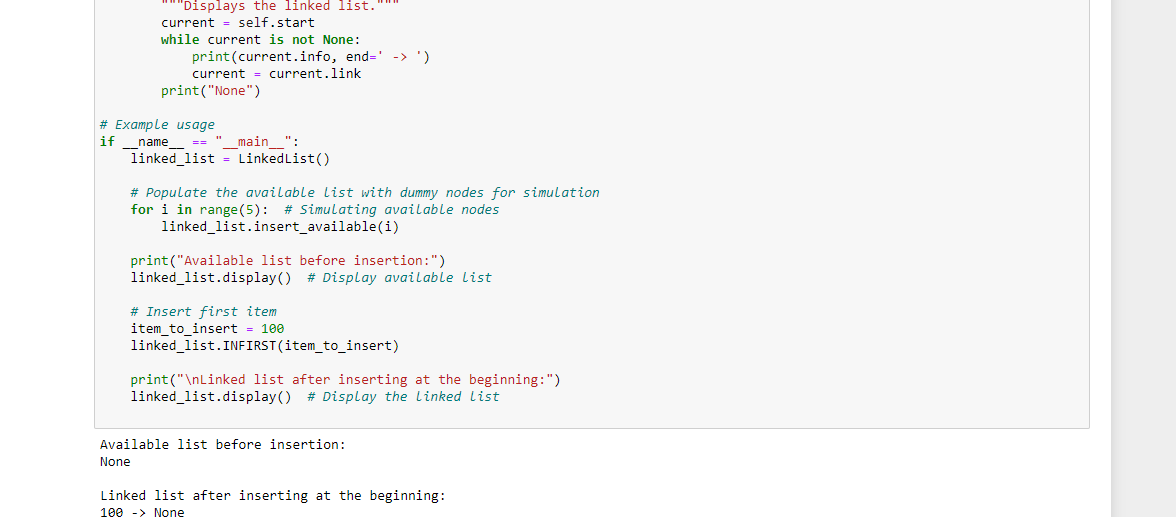
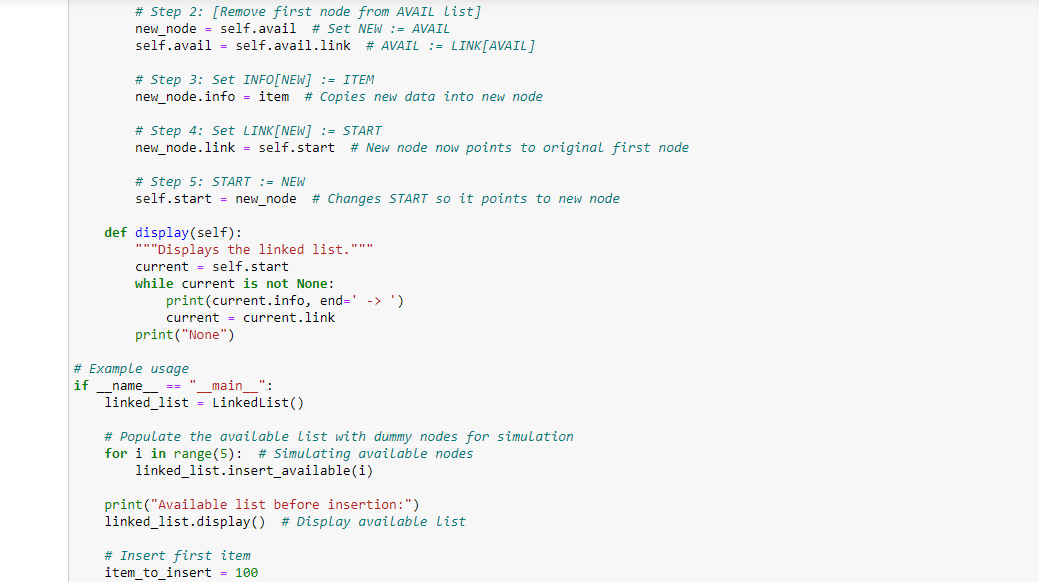
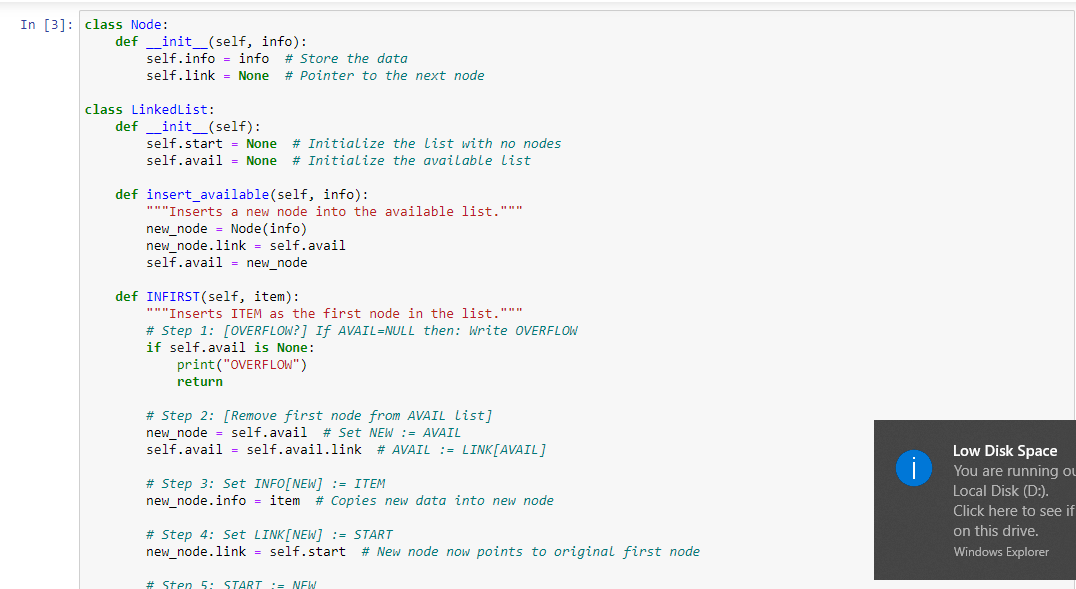
**Traversing a Linked List**

1. 

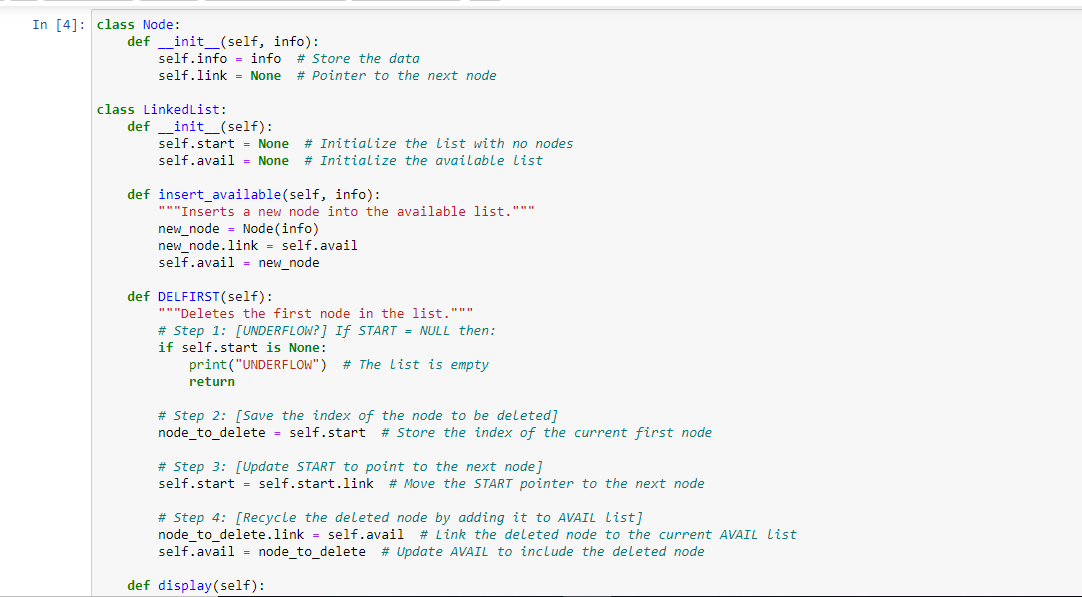
  
**Searching an element as in an Unsorted linked list**

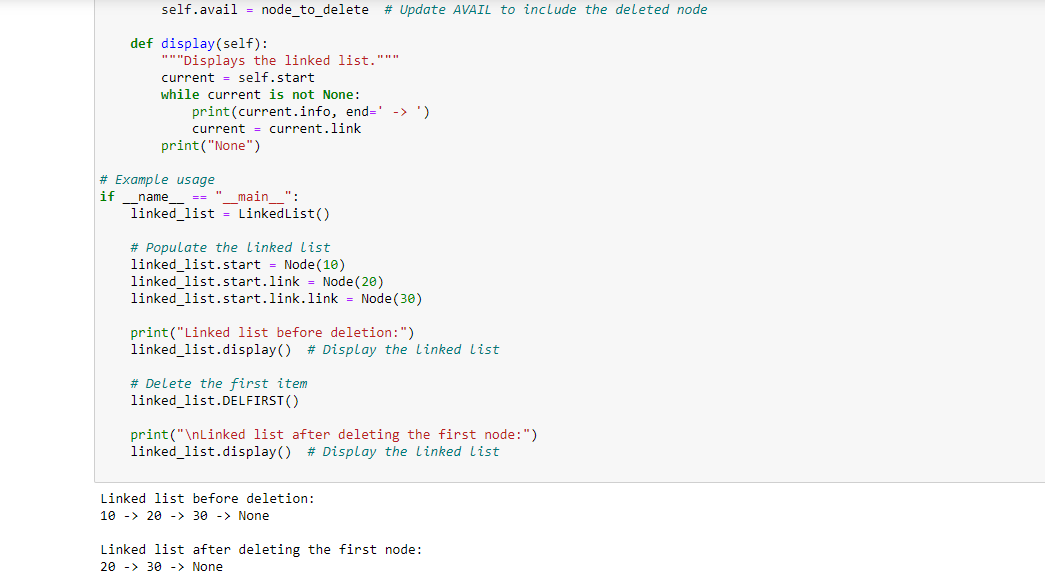


**Inserting an element as first index node**

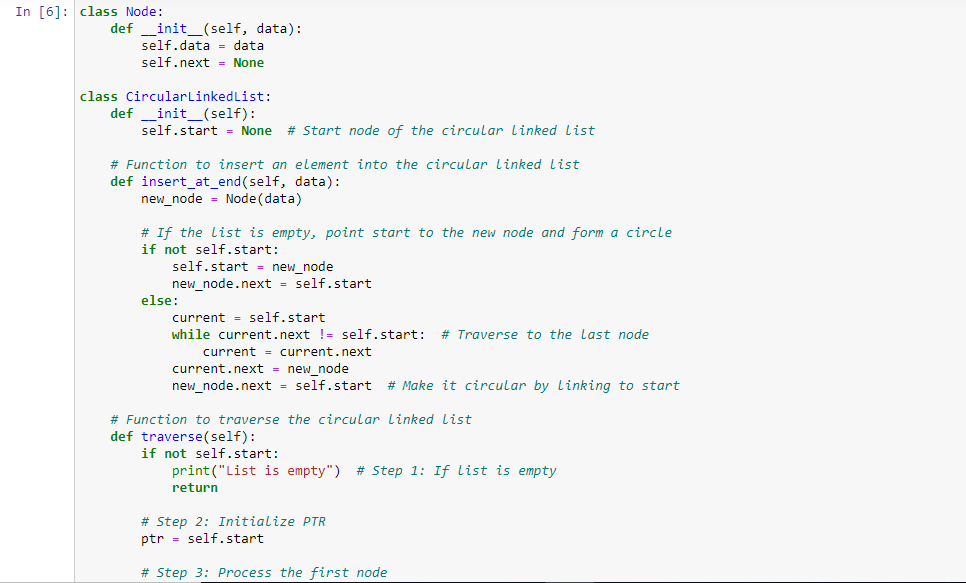


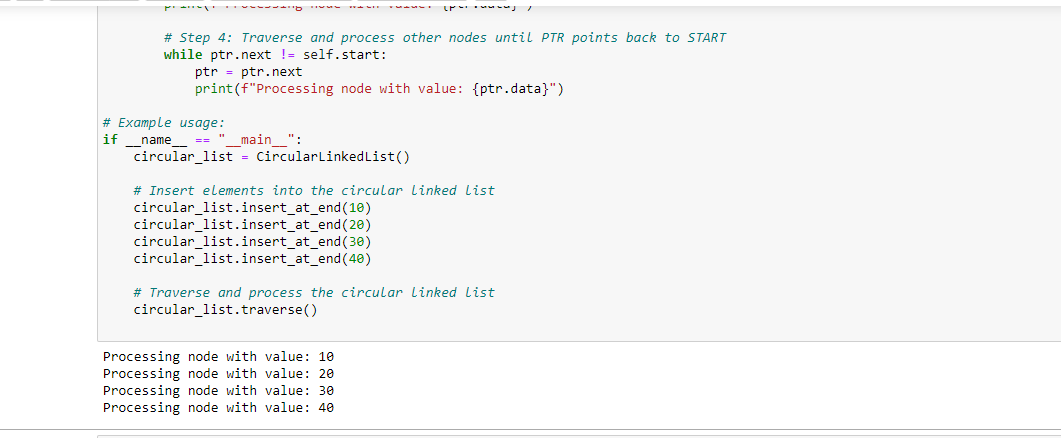
**Deleting a node from the linked list**



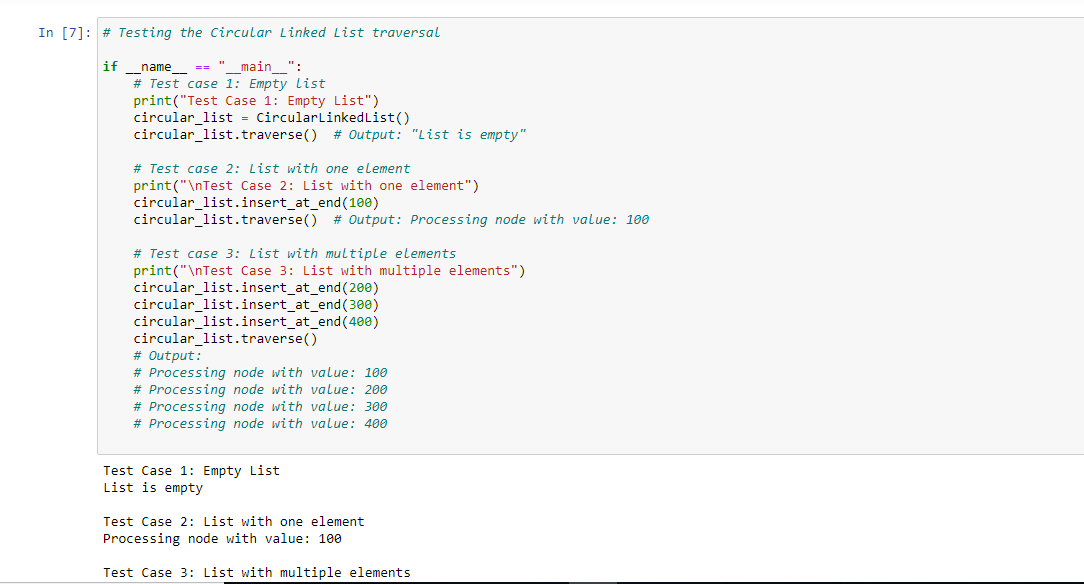


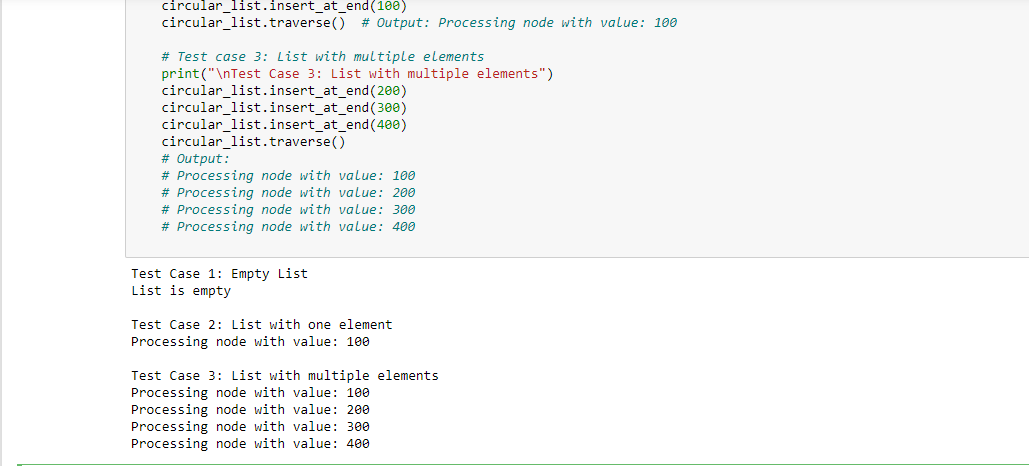
**Traversing a Circular Linked List**





1. Test your programs with different values and show the output.





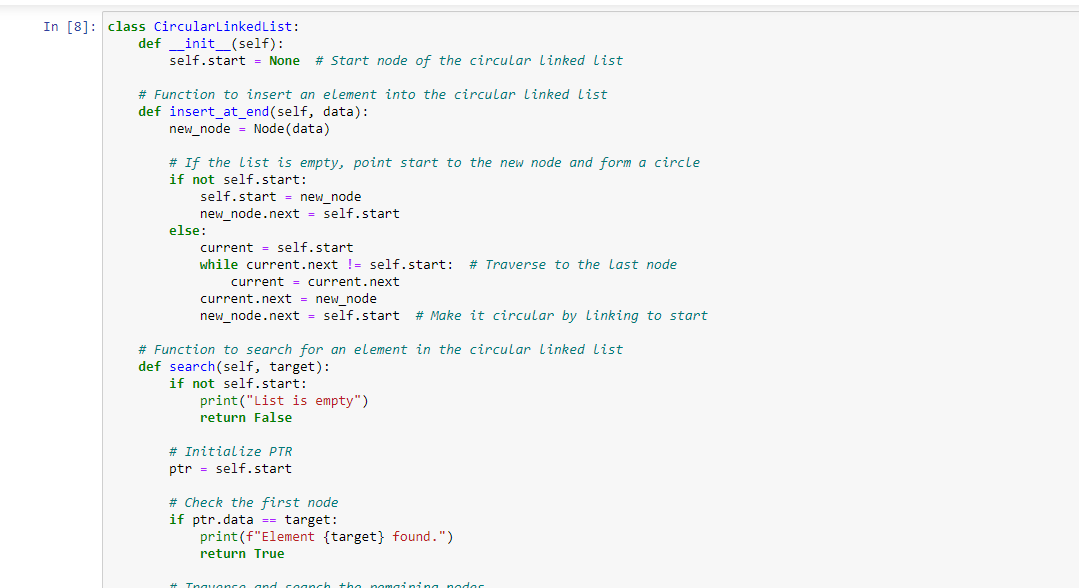
1. Modify the given implementation for traversing the list so that it can be used for searching an element in linked list.

**4. Modify the Given Implementation for Searching an Element in the Circular Linked List:**

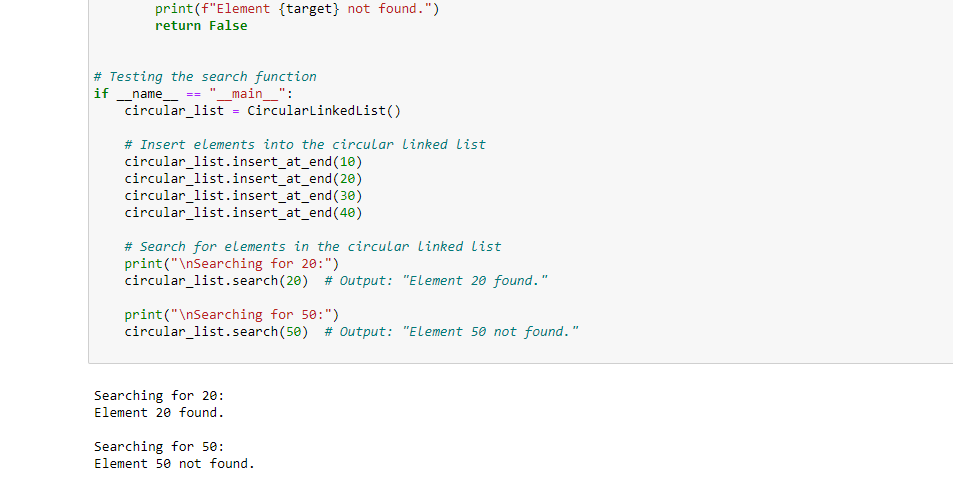
To search for an element in the circular linked list, we can modify the traverse function to stop and return the result when the target element is found.

**Modified Algorithm for Searching:**

1. **Start**: If the list is empty, print "List is empty" and exit.
2. **Initialize Pointer**: Set PTR to START.
3. **Search First Node**: Check if the current node contains the target value.
4. **Traverse the List**:
   * If the target element is not found, move the pointer to the next node and repeat the check.
   * If the pointer reaches back to START, the element is not in the list.
5. **Exit**: Print whether the target element was found or not.







1. Devise an algorithm for inserting an element at the end of linked list and implement it.

**4. Algorithm for Inserting an Element at the End of a Linked List:**

**Algorithm**:

1. Create a new node with the given data.
2. If the linked list is empty, set the head to point to the new node.
3. If the linked list is not empty:
   * Traverse the list until you reach the last node.
   * Set the next of the last node to point to the new node.
4. The new node's next pointer should be None (as it's the last element).

**Python Implementation**:

python

Copy code

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class LinkedList:

def \_\_init\_\_(self):

self.head = None

# Insert an element at the end of the list

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

new\_node = Node(data)

# If the list is empty, set the head to the new node

if not self.head:

self.head = new\_node

return

# Otherwise, traverse to the end of the list

current = self.head

while current.next:

current = current.next

# Link the last node to the new node

current.next = new\_node

# Method to print the list (for testing purposes)

def print\_list(self):

current = self.head

while current:

print(current.data, end=" -> ")

current = current.next

print("None")

# Example usage

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

linked\_list = LinkedList()

linked\_list.insert\_at\_end(10)

linked\_list.insert\_at\_end(20)

linked\_list.insert\_at\_end(30)

linked\_list.print\_list() # Output: 10 -> 20 -> 30 -> None

1. Devise an algorithm for deleting an element at the middle of linked list and implement it.

**Algorithm for Deleting an Element in the Middle of a Linked List:**

**Algorithm**:

1. If the linked list is empty, return (nothing to delete).
2. If the node to be deleted is the first node, update the head to the next node.
3. Otherwise:
   * Traverse the list to find the node just before the target node (let's call it the previous node).
   * Update the previous node's next pointer to point to the node after the target node.
4. Free the memory of the target node (in languages like Python, this is handled by garbage collection).

**Python Implementation**:

python

Copy code

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class LinkedList:

def \_\_init\_\_(self):

self.head = None

# Insert an element at the end of the list (for testing)

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

new\_node = Node(data)

if not self.head:

self.head = new\_node

return

current = self.head

while current.next:

current = current.next

current.next = new\_node

# Method to delete a node from the middle of the list

def delete\_middle(self, target):

if not self.head:

return # List is empty

# If the target is the head node

if self.head.data == target:

self.head = self.head.next # Move head to the next node

return

# Traverse to find the target node and its previous node

current = self.head

previous = None

while current and current.data != target:

previous = current

current = current.next

# If the target node was found, unlink it

if current:

previous.next = current.next

else:

print(f"{target} not found in the list.")

# Method to print the list (for testing purposes)

def print\_list(self):

current = self.head

while current:

print(current.data, end=" -> ")

current = current.next

print("None")

# Example usage

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

linked\_list = LinkedList()

# Inserting elements

linked\_list.insert\_at\_end(10)

linked\_list.insert\_at\_end(20)

linked\_list.insert\_at\_end(30)

linked\_list.insert\_at\_end(40)

linked\_list.print\_list() # Output: 10 -> 20 -> 30 -> 40 -> None

# Deleting an element from the middle

linked\_list.delete\_middle(30)

linked\_list.print\_list() # Output: 10 -> 20 -> 40 -> None

1. **Case Stude: Managing Clinic Appointments with Linked Lists**

**Background:**

Dr. Ali runs a small clinic where he manages patient appointments using a linked list data structure. Each appointment is represented by a node in the linked list, which includes the patient's name and appointment time. The list is maintained in chronological order based on appointment time.

**Tasks:**

1. **Traversing the Appointment List:**
   * Describe how you would traverse a linked list to display all the appointments for the day. What steps would you take to ensure that each appointment is printed in the correct order?
2. **Searching for a Specific Appointment:**
   * Explain how you would search for a specific patient's appointment in the linked list. How would you determine if the patient has an appointment, and what actions would you take if the patient is not found in the list?
3. **Inserting a New Appointment:**
   * Outline the process for inserting a new appointment into the linked list. Assume that the list is already sorted by appointment time. How would you ensure that the new appointment is placed in the correct position to maintain the chronological order?

**Scenario Example:**

Consider the following appointments in a linked list:

* 09:00 AM: John Doe
* 10:00 AM: Jane Roe
* 11:00 AM: Sam Smith

**Questions:**

1. **Traversing the List:** How would you traverse the linked list to display the schedule for the day?
2. **Searching for an Appointment:** If Amna calls to inquire about her appointment time, how would you search the list to find her appointment?
3. **Inserting a New Appointment:** A new patient, Aqsa, books an appointment for 09:30 AM. How would you insert this new appointment into the linked list while maintaining the chronological order?

**Instructions:**

* Provide detailed explanations for each task.
* Assume the linked list is implemented as a sequence of nodes where each node contains a patient's name, appointment time, and a pointer to the next node.

### Case Study: Managing Clinic Appointments with Linked Lists

#### 1. Traversing the Appointment List:

To traverse the linked list and display all the appointments for the day, follow these steps:

1. \*\*Start at the Head\*\*: Set a pointer (e.g., `current`) to the head of the linked list, which represents the first appointment of the day.

2. \*\*Print Each Appointment\*\*: Move through the list using the pointer, printing each patient's name and appointment time as you go.

3. \*\*Move to the Next Node\*\*: Update the pointer to point to the next node (the next appointment) in the list.

4. \*\*Repeat Until End of List\*\*: Continue this process until you reach the end of the list (when the pointer is `None`).

Since the linked list is already sorted chronologically by appointment time, simply traversing from the head to the end will ensure that the appointments are printed in order.

\*\*Example\*\*:

If the appointments in the list are:

- 09:00 AM: John Doe

- 10:00 AM: Jane Roe

- 11:00 AM: Sam Smith

The traversal would display them in this order.

#### 2. Searching for a Specific Appointment:

To search for a specific patient's appointment in the linked list, follow these steps:

1. \*\*Start at the Head\*\*: Set a pointer (e.g., `current`) to the head of the linked list.

2. \*\*Compare Patient Names\*\*: For each node, compare the patient's name stored in the node with the name of the patient you are searching for (e.g., "Amna").

3. \*\*Move to the Next Node\*\*: If the current node does not contain the name you are looking for, move the pointer to the next node.

4. \*\*Check for End of List\*\*: If you reach the end of the list (`None`) without finding the patient, print a message indicating that the patient does not have an appointment.

5. \*\*Appointment Found\*\*: If the patient's name matches a node, print the appointment time and return.

If Amna calls and asks for her appointment time, this method will search the list node by node until either Amna’s appointment is found or it’s determined that she is not in the list.

#### 3. Inserting a New Appointment:

To insert a new appointment while maintaining chronological order, follow these steps:

1. \*\*Create a New Node\*\*: Create a new node that stores the patient's name and appointment time (e.g., Aqsa, 09:30 AM).

2. \*\*Find the Correct Position\*\*:

- If the list is empty, set the new node as the head of the list.

- If the new appointment time is earlier than the current head, insert the new node at the head of the list.

- Otherwise, traverse the list and find the first appointment whose time is later than the new appointment time.

3. \*\*Insert the Node\*\*:

- Insert the new node before the node with the later appointment time. Update the pointers of the surrounding nodes so that the list remains linked.

4. \*\*Update the List\*\*: Ensure that the new node’s `next` pointer points to the correct subsequent node, and the previous node's `next` pointer is updated to point to the new node.

\*\*Scenario Example\*\*:

The current list is:

- 09:00 AM: John Doe

- 10:00 AM: Jane Roe

- 11:00 AM: Sam Smith

A new patient, Aqsa, books an appointment at 09:30 AM. To insert Aqsa's appointment:

1. Start at the head of the list (John Doe, 09:00 AM).

2. Since Aqsa's appointment time (09:30 AM) is later than John's but earlier than Jane's (10:00 AM), we insert Aqsa between John and Jane.

3. The resulting list will be:

- 09:00 AM: John Doe

- 09:30 AM: Aqsa

- 10:00 AM: Jane Roe

- 11:00 AM: Sam Smith

### Summary:

- \*\*Traversing\*\*: Start at the head and visit each node to print the appointments in the correct order.

- \*\*Searching\*\*: Compare each node's name with the target patient's name and stop when found.

- \*\*Inserting\*\*: Traverse the list to find the correct chronological position and insert the new appointment while maintaining order.

This approach ensures efficient appointment management and easy access to appointments in a clinical setting.







