1. LinkedList Creation and Traversal

```
class Node:
    data  // Data to be stored in the node
    next  // Reference to the next node in the list

    constructor(data):
        this.data = data
        this.next = null

class LinkedList:
    head  // Reference to the head of the list

    constructor():
        this.head = null

    # Insert a new node at the beginning of the linked list
    method insertAtBeginning(data):
        newNode = new Node(data)
        newNode.next = this.head
        this.head = newNode

    # Traverse and print the linked list
    method printList():
        currentNode = this.head
        while currentNode is not null:
            print(currentNode.data)
            currentNode = currentNode.next
```

2. LinkedList Add at ending

```
class Node:
   data  // Data to be stored in the node
   next  // Reference to the next node in the list

   constructor(data):
      this.data = data
      this.next = null

class LinkedList:
   head  // Reference to the head of the list

   constructor():
      this.head = null

   # Insert a new node at the end of the linked list
   method insertAtEnd(data):
      newNode = new Node(data)
      if this.head is null:
         this.head = newNode
      else:
         currentNode = this.head
         while currentNode.next is not null:
            currentNode = currentNode.next
         currentNode.next = newNode

   # Traverse and print the linked list
   method printList():
      currentNode = this.head
      while currentNode is not null:
         print(currentNode.data)
         currentNode = currentNode.next
```

3. LinkedList Add at any random position

```
class Node:
   data  // Data to be stored in the node
   next  // Reference to the next node in the list

   constructor(data):
      this.data = data
      this.next = null

class LinkedList:
   head  // Reference to the head of the list

   constructor():
      this.head = null

   # Insert a new node after a given node
   method insertAfterNode(existingNodeData, newData):
      newNode = new Node(newData)

      # Find the node with existingNodeData
      currentNode = this.head
      while currentNode is not null and currentNode.data != existingNodeData:
         currentNode = currentNode.next

      # If the existing node was found, insert newNode after it
      if currentNode is not null:
         newNode.next = currentNode.next
         currentNode.next = newNode

   # Traverse and print the linked list
   method printList():
      currentNode = this.head
      while currentNode is not null:
         print(currentNode.data)
         currentNode = currentNode.next
```

4. Delete a node from a Singly Linked List

```
class Node:
    data  // Data to be stored in the node
    next  // Reference to the next node in the list

    constructor(data):
        this.data = data
        this.next = null

class LinkedList:
    head  // Reference to the head of the list

    constructor():
        this.head = null

    # Delete a node with a given data value
    method deleteNode(dataToDelete):
        if this.head is null:
            return  // List is empty, nothing to delete

        # If the head node contains the data to delete, update the head
        if this.head.data == dataToDelete:
            this.head = this.head.next
            return

        # Find the node before the one to delete
        currentNode = this.head
        while currentNode.next is not null and currentNode.next.data != dataToDelete:
            currentNode = currentNode.next

        # If the node to delete was found, remove it from the list
        if currentNode.next is not null:
            currentNode.next = currentNode.next.next

    # Traverse and print the linked list
    method printList():
        currentNode = this.head
        while currentNode is not null:
            print(currentNode.data)
            currentNode = currentNode.next
```

5. Update a node in a Singly Linked List

```
class Node:
   data  // Data to be stored in the node
   next  // Reference to the next node in the list

   constructor(data):
      this.data = data
      this.next = null

class LinkedList:
   head  // Reference to the head of the list

   constructor():
      this.head = null

   # Update a node with a new data value
   method updateNode(oldData, newData):
      currentNode = this.head

      # Traverse the list to find the node with the old data
      while currentNode is not null and currentNode.data != oldData:
         currentNode = currentNode.next

      # If the node with old data was found, update its data
      if currentNode is not null:
         currentNode.data = newData

   # Traverse and print the linked list
   method printList():
      currentNode = this.head
      while currentNode is not null:
         print(currentNode.data)
         currentNode = currentNode.next
```