# قسم الانظمة الطبية الذكية

## المرحلة الثانية

# Lecture: ( 4 )

**Subject: Object oriented programming II**
**Class: Second**
**Lecturers: Dr. Dunia H. Hameed , Dr. Maytham N. Meqdad**

# *Object Oriented Programming (II) – Fourth Lecture*

### 3.4.3 Doubly Linked List Operations

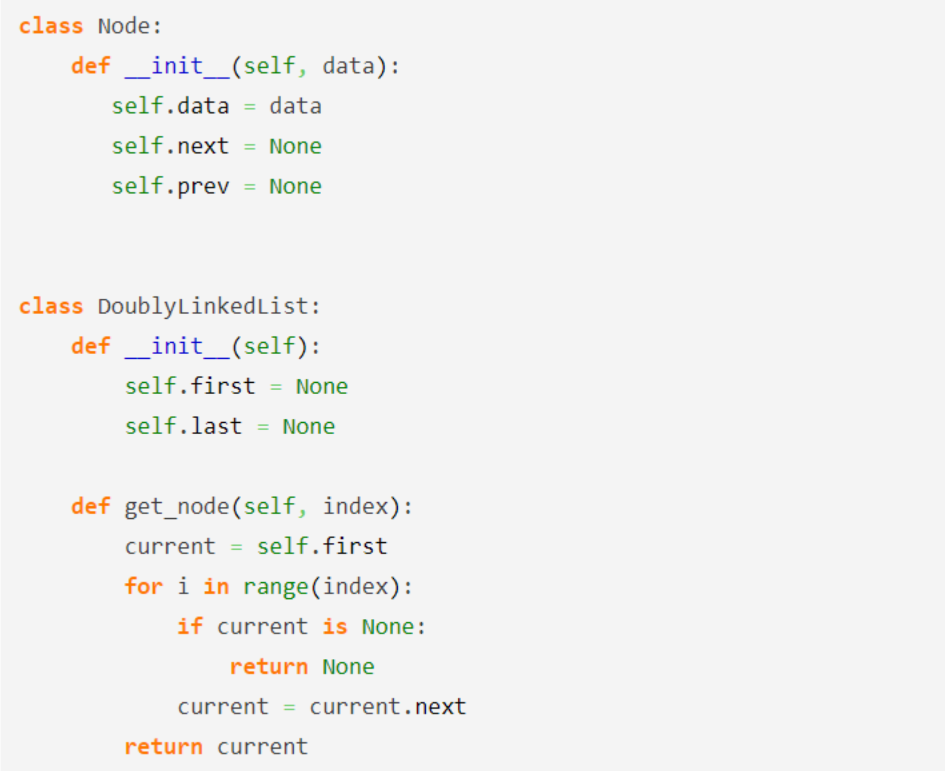
**Problem Description**

The program creates a doubly linked list and presents the user with a menu to perform various operations on the list.

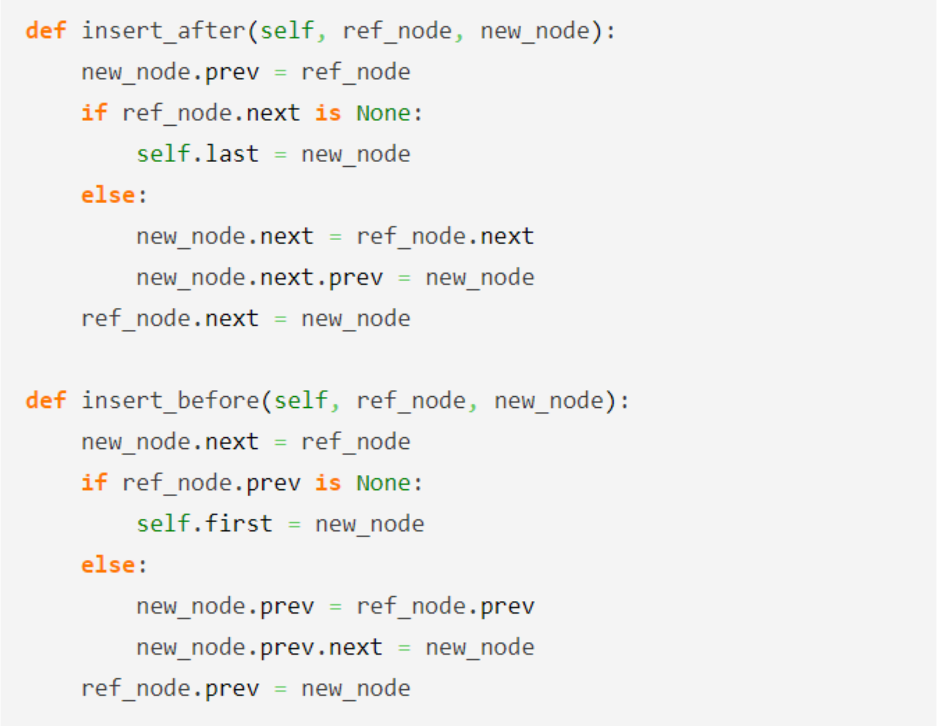
**Problem Solution**

1. Create a class Node with instance variables data and next.
2. Create a class DoublyLinkedList with instance variables first and last.
3. The variable first points to the first element in the doubly linked list while last points to the last element.
4. Define methods get_node, insert_after, insert_before, insert_at_beg, insert_at_end, remove and display.
5. get_node takes an index as argument and traverses the list from the first node that many times to return the node at that index.
6. The methods insert_after and insert_before insert a node after or before some reference node in the list.
7. The methods insert_at_beg and insert_at_end insert a node at the first or last position of the list.
8. The method remove takes a node as argument and removes it from the list.
9. The method display traverses the list from the first node and prints the data of each node.
10. Create an instance of DoublyLinkedList and present the user with a menu to perform operations on the list.

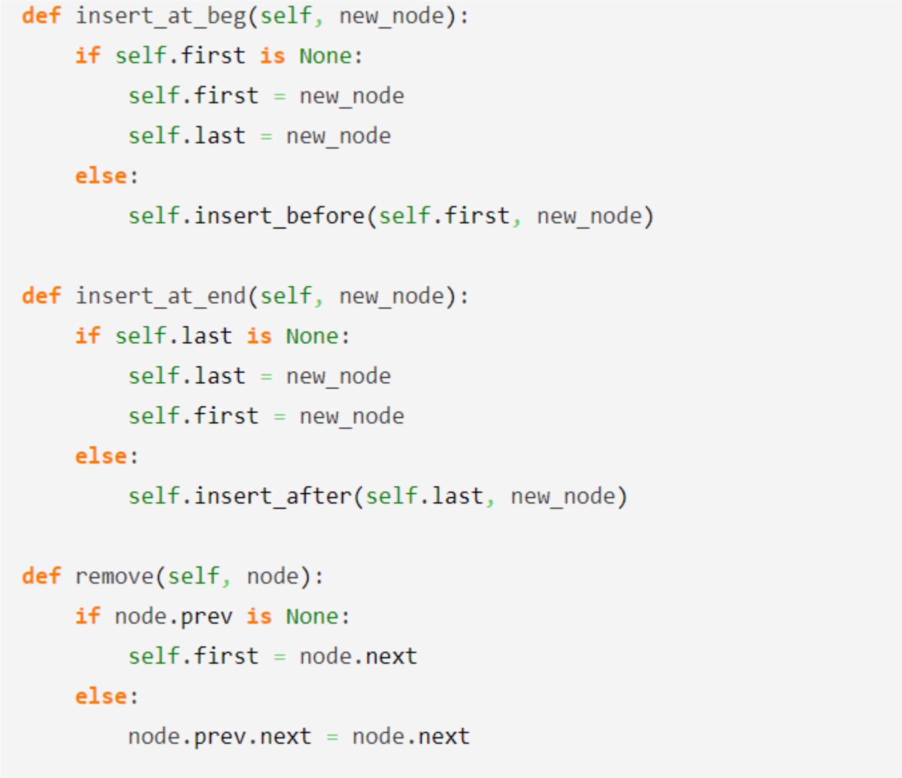
## Program/Source Code



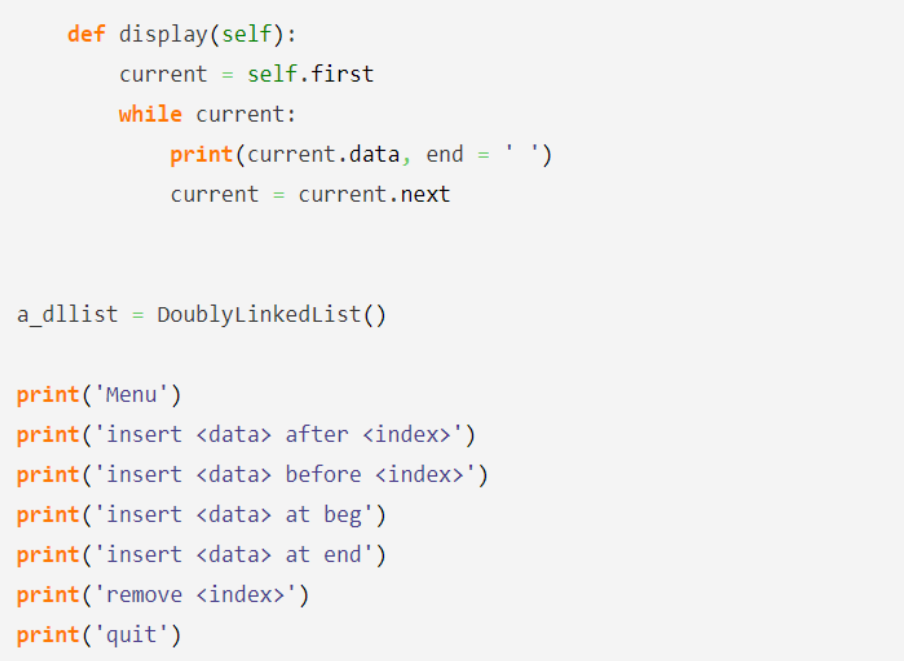
```
class Node:
    def __init__(self, data):
       self.data = data
       self.next = None
       self.prev = None


class DoublyLinkedList:
    def __init__(self):
        self.first = None
        self.last = None

    def get_node(self, index):
        current = self.first
        for i in range(index):
            if current is None:
                return None
            current = current.next
        return current
```



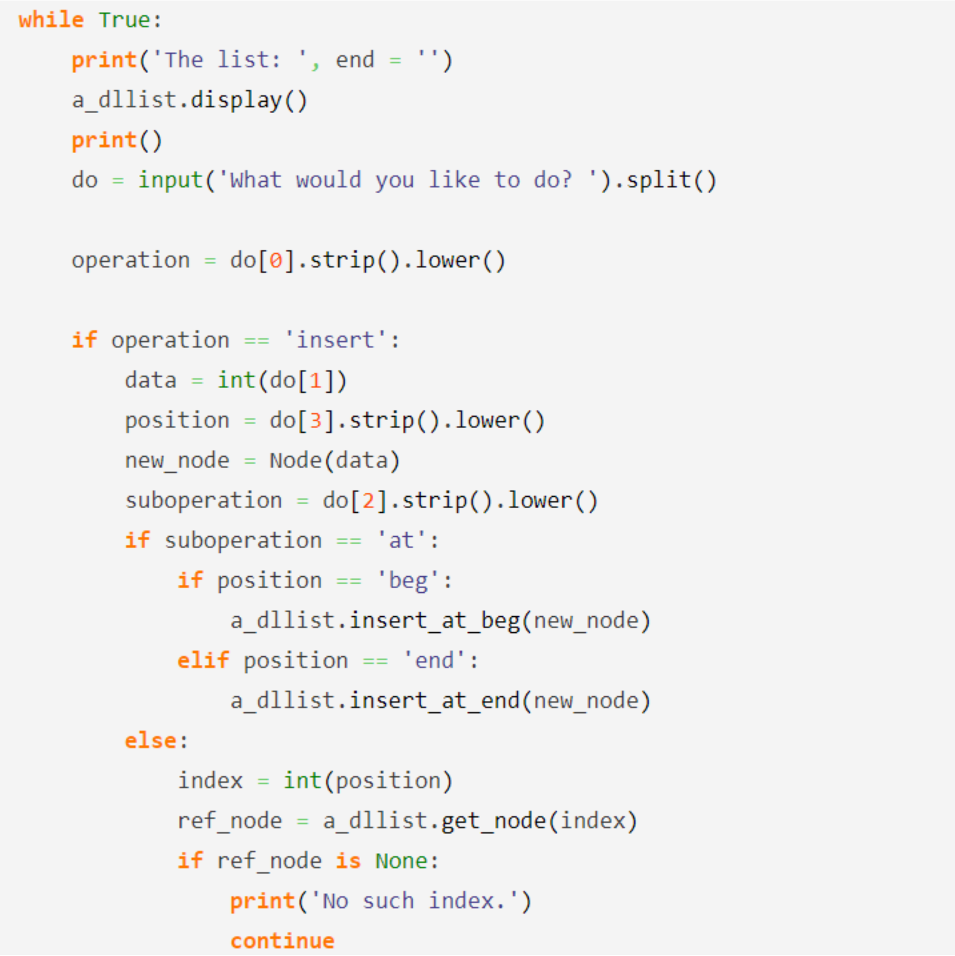
```
def insert_after(self, ref_node, new_node):
    new_node.prev = ref_node
    if ref_node.next is None:
        self.last = new_node
    else:
        new_node.next = ref_node.next
        new_node.next.prev = new_node
    ref_node.next = new_node

def insert_before(self, ref_node, new_node):
    new_node.next = ref_node
    if ref_node.prev is None:
        self.first = new_node
    else:
        new_node.prev = ref_node.prev
        new_node.prev.next = new_node
    ref_node.prev = new_node
```



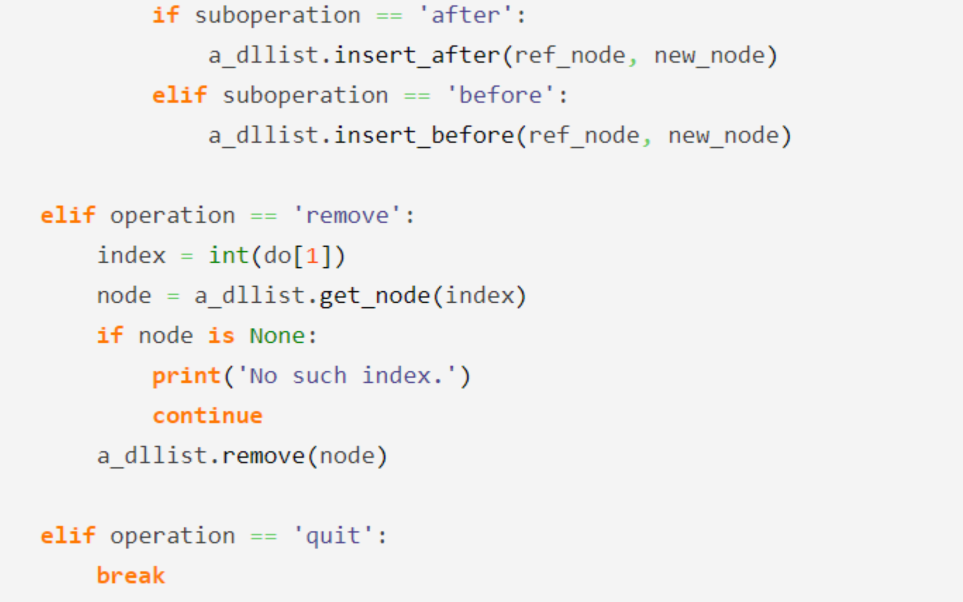
```
def insert_at_beg(self, new_node):
    if self.first is None:
        self.first = new_node
        self.last = new_node
    else:
        self.insert_before(self.first, new_node)

def insert_at_end(self, new_node):
    if self.last is None:
        self.last = new_node
        self.first = new_node
    else:
        self.insert_after(self.last, new_node)

def remove(self, node):
    if node.prev is None:
        self.first = node.next
    else:
        node.prev.next = node.next
```



```
    def display(self):
        current = self.first
        while current:
            print(current.data, end = ' ')
            current = current.next


a_dllist = DoublyLinkedList()

print('Menu')
print('insert <data> after <index>')
print('insert <data> before <index>')
print('insert <data> at beg')
print('insert <data> at end')
print('remove <index>')
print('quit')
```



```
while True:
    print('The list: ', end = '')
    a_dllist.display()
    print()
    do = input('What would you like to do? ').split()

    operation = do[0].strip().lower()

    if operation == 'insert':
        data = int(do[1])
        position = do[3].strip().lower()
        new_node = Node(data)
        suboperation = do[2].strip().lower()
        if suboperation == 'at':
            if position == 'beg':
                a_dllist.insert_at_beg(new_node)
            elif position == 'end':
                a_dllist.insert_at_end(new_node)
        else:
            index = int(position)
            ref_node = a_dllist.get_node(index)
            if ref_node is None:
                print('No such index.')
                continue
```



```
            if suboperation == 'after':
                a_dllist.insert_after(ref_node, new_node)
            elif suboperation == 'before':
                a_dllist.insert_before(ref_node, new_node)

    elif operation == 'remove':
        index = int(do[1])
        node = a_dllist.get_node(index)
        if node is None:
            print('No such index.')
            continue
        a_dllist.remove(node)

    elif operation == 'quit':
        break
```

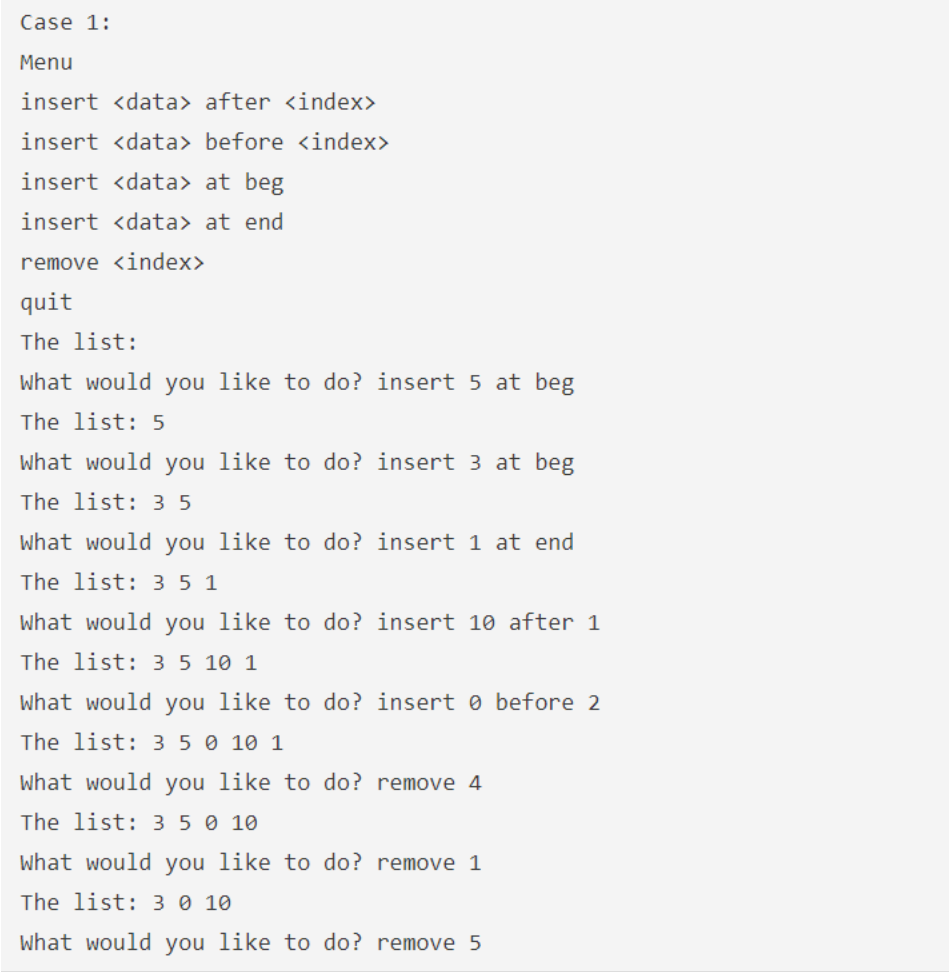
## Program Explanation

1. An instance of DoublyLinkedList is created.

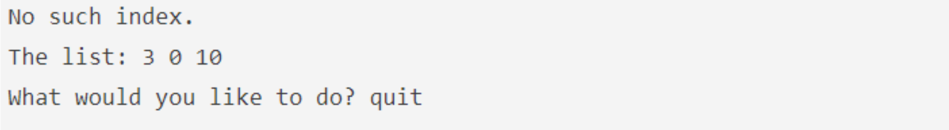
2. The user is presented with a menu to perform various operations on the list.

3. The corresponding methods are called to perform each operation.

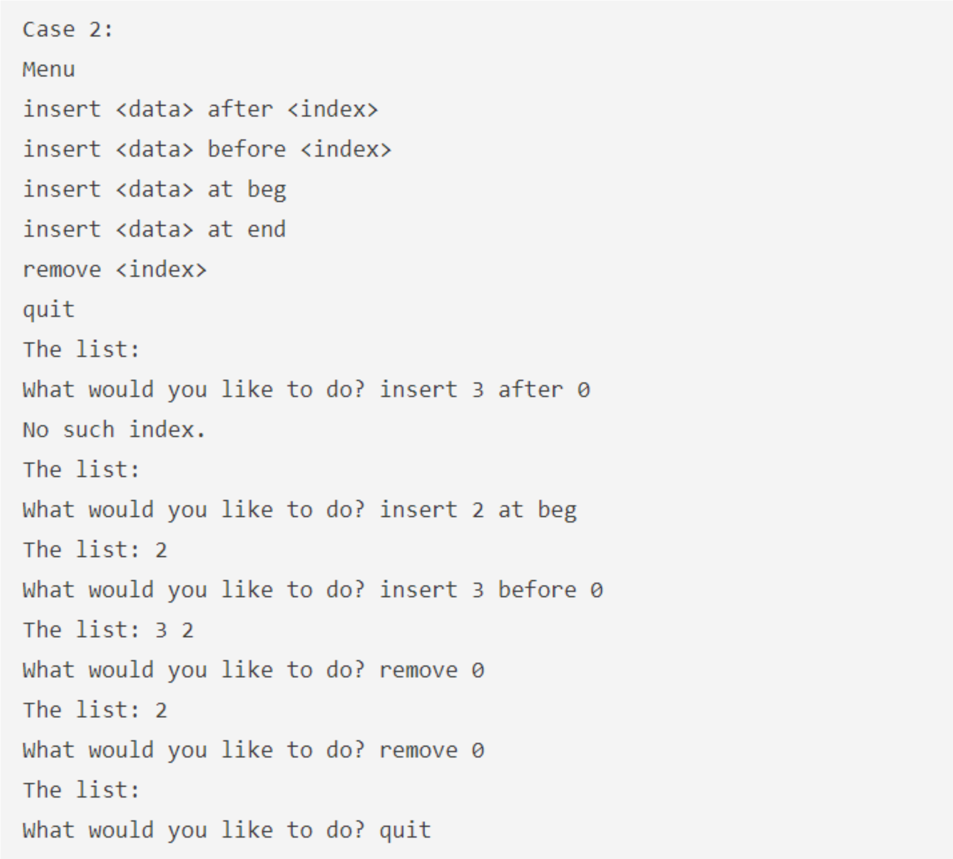
## Runtime Test Cases



```
Case 1:
Menu
insert <data> after <index>
insert <data> before <index>
insert <data> at beg
insert <data> at end
remove <index>
quit
The list:
What would you like to do? insert 5 at beg
The list: 5
What would you like to do? insert 3 at beg
The list: 3 5
What would you like to do? insert 1 at end
The list: 3 5 1
What would you like to do? insert 10 after 1
The list: 3 5 10 1
What would you like to do? insert 0 before 2
The list: 3 5 0 10 1
What would you like to do? remove 4
The list: 3 5 0 10
What would you like to do? remove 1
The list: 3 0 10
What would you like to do? remove 5
```



```
No such index.
The list: 3 0 10
What would you like to do? quit
```



```
Case 2:
Menu
insert <data> after <index>
insert <data> before <index>
insert <data> at beg
insert <data> at end
remove <index>
quit
The list:
What would you like to do? insert 3 after 0
No such index.
The list:
What would you like to do? insert 2 at beg
The list: 2
What would you like to do? insert 3 before 0
The list: 3 2
What would you like to do? remove 0
The list: 2
What would you like to do? remove 0
The list:
What would you like to do? quit
```

### 3.4.4 Circular Doubly Linked List Operations

**Problem Description**

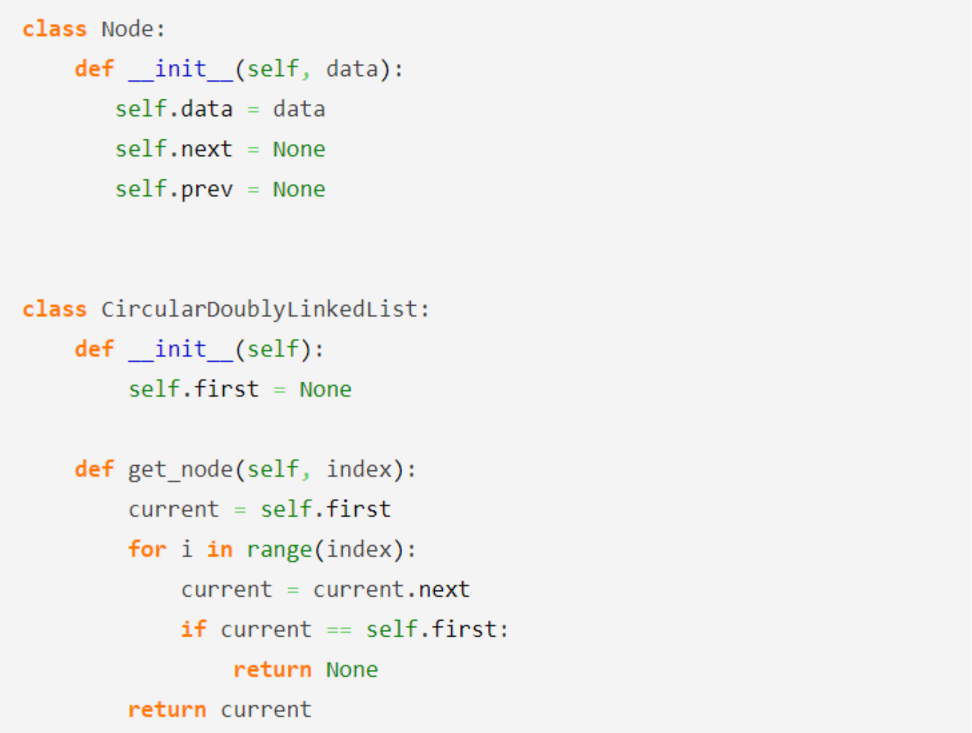
The program creates a circular doubly linked list and presents the user with a menu to perform various operations on the list.

**Problem Solution**

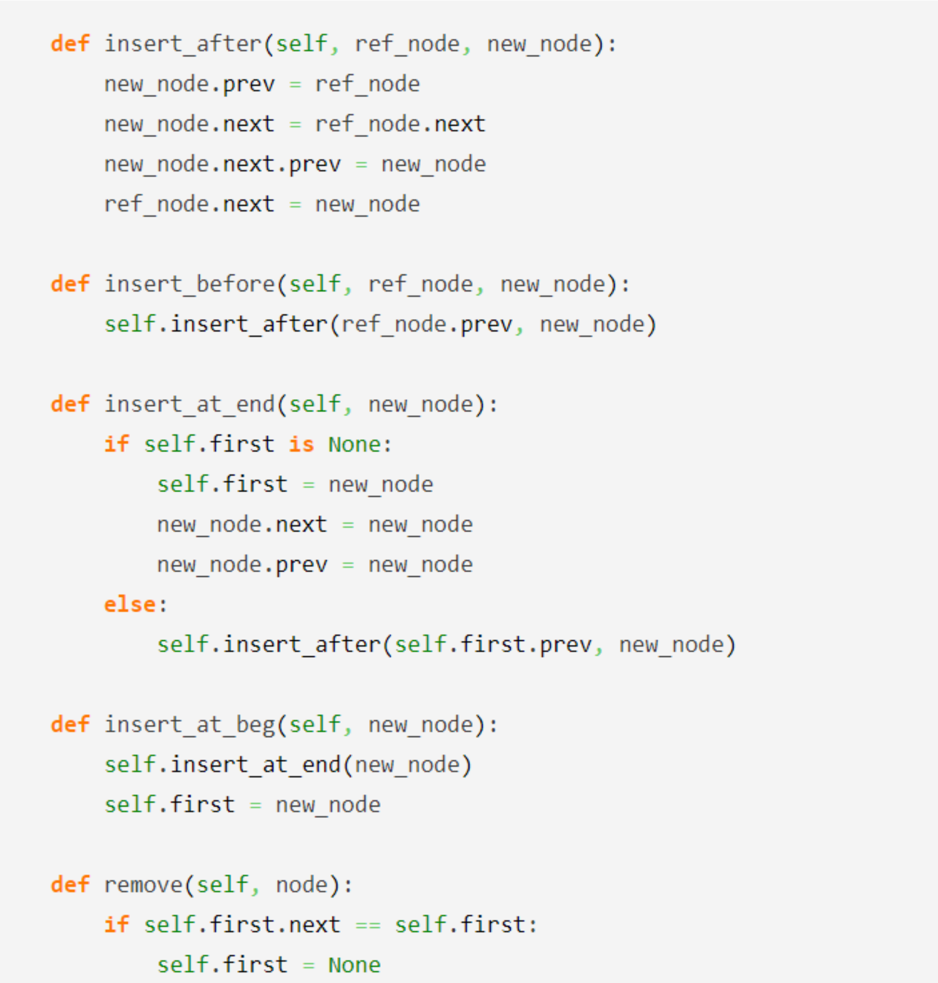
1. Create a class Node with instance variables data and next.
2. Create a class CircularDoublyLinkedList with instance variable first.
3. The variable first points to the first element in the circular doubly linked list.
4. Define methods get_node, insert_after, insert_before, insert_at_beg, insert_at_end, remove and display.

5. get_node takes an index as argument and traverses the list from the first node that many times to return the node at that index. It stops if it reaches the first node again.
6. The methods insert_after and insert_before insert a node after or before some reference node in the list.
7. The methods insert_at_beg and insert_at_end insert a node at the first or last position of the list. insert_at_beg modifies the variable first to point to the new node.
8. The method remove takes a node as argument and removes it from the list.
9. The method display traverses the list from the first node and prints the data of each node until it reaches the first node again.
10. Create an instance of CircularDoublyLinkedList and present the user with a menu to perform operations on the list.

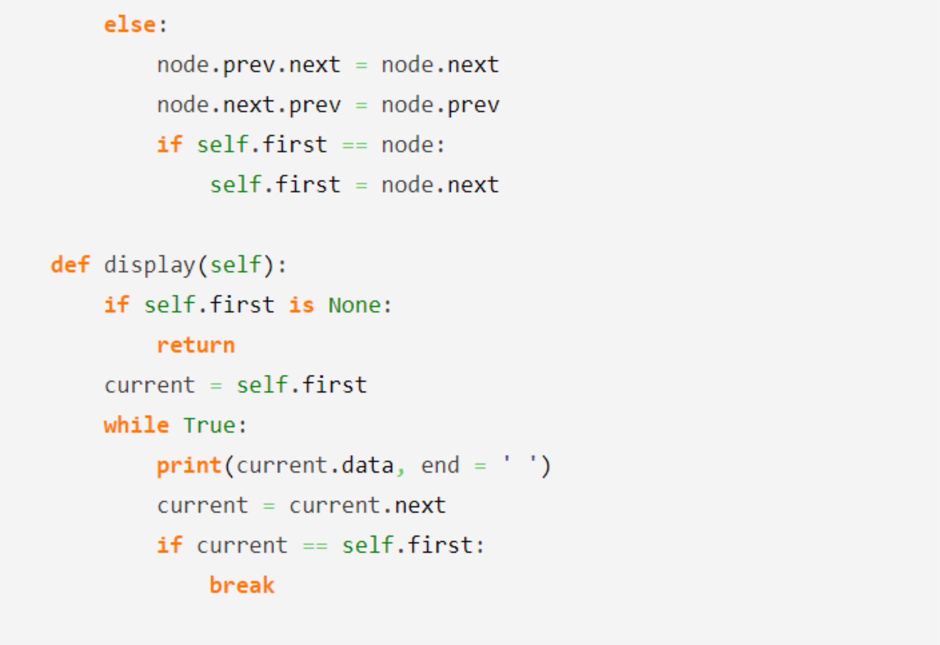
**Program/Source Code**



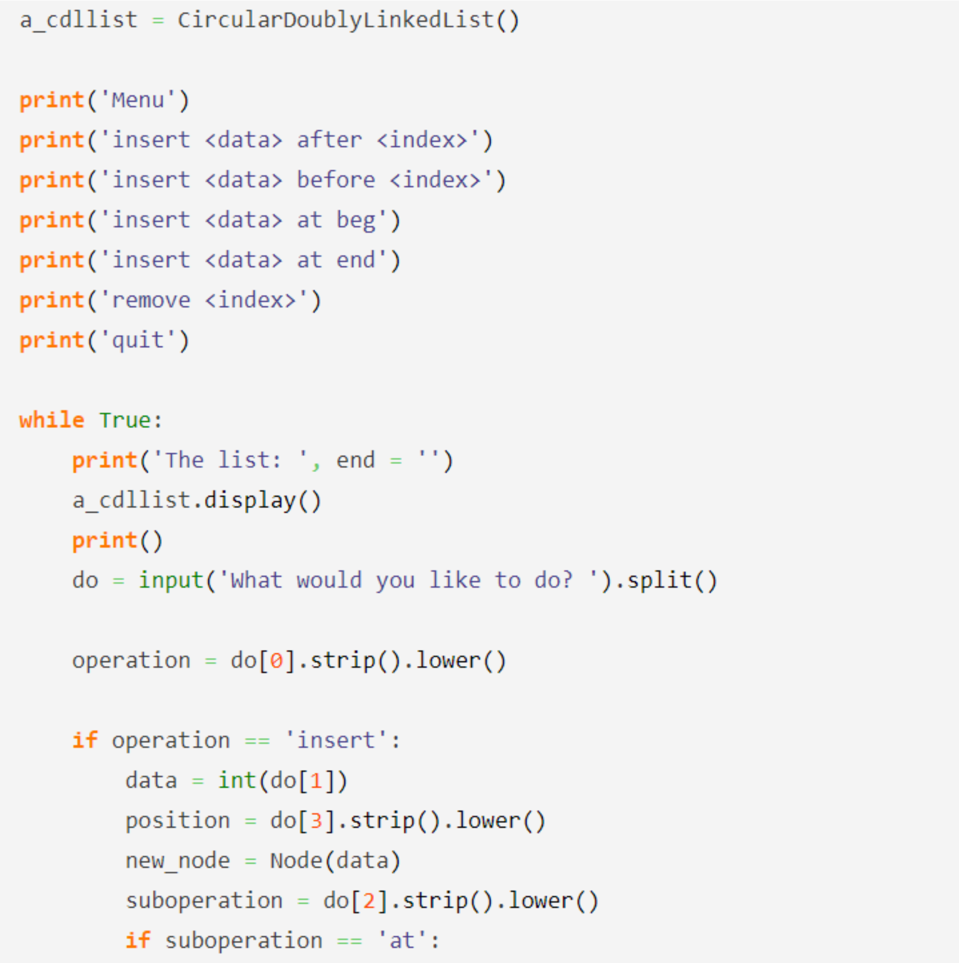
```
class Node:
    def __init__(self, data):
       self.data = data
       self.next = None
       self.prev = None


class CircularDoublyLinkedList:
    def __init__(self):
        self.first = None

    def get_node(self, index):
        current = self.first
        for i in range(index):
            current = current.next
            if current == self.first:
                return None
        return current
```



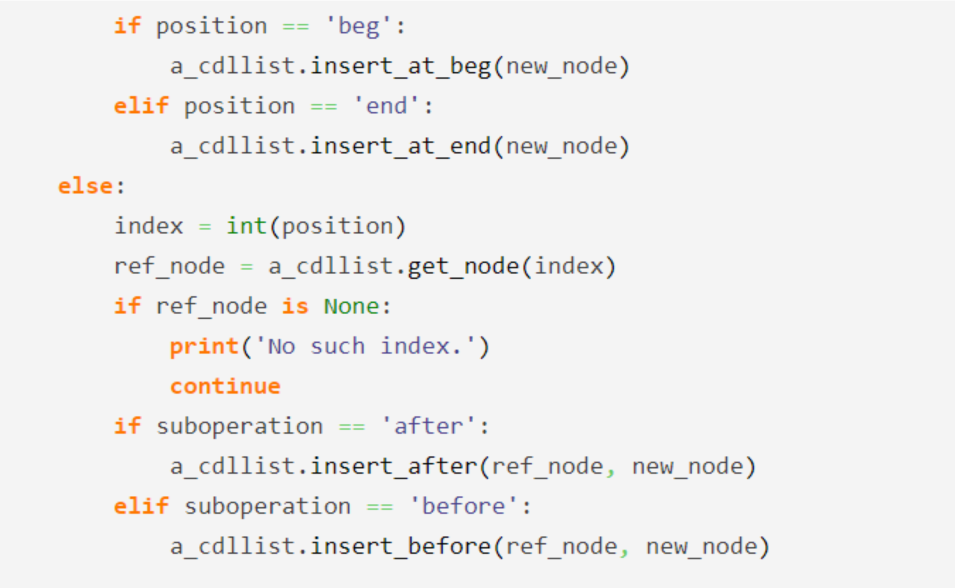
```
    def insert_after(self, ref_node, new_node):
        new_node.prev = ref_node
        new_node.next = ref_node.next
        new_node.next.prev = new_node
        ref_node.next = new_node

    def insert_before(self, ref_node, new_node):
        self.insert_after(ref_node.prev, new_node)

    def insert_at_end(self, new_node):
        if self.first is None:
            self.first = new_node
            new_node.next = new_node
            new_node.prev = new_node
        else:
            self.insert_after(self.first.prev, new_node)

    def insert_at_beg(self, new_node):
        self.insert_at_end(new_node)
        self.first = new_node

    def remove(self, node):
        if self.first.next == self.first:
            self.first = None
```



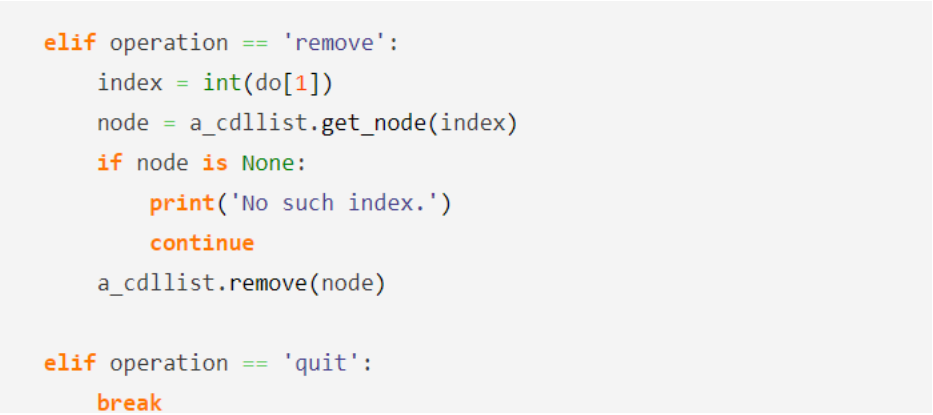
```
        else:
            node.prev.next = node.next
            node.next.prev = node.prev
            if self.first == node:
                self.first = node.next

    def display(self):
        if self.first is None:
            return
        current = self.first
        while True:
            print(current.data, end = ' ')
            current = current.next
            if current == self.first:
                break
```



```
a_cdllist = CircularDoublyLinkedList()

print('Menu')
print('insert <data> after <index>')
print('insert <data> before <index>')
print('insert <data> at beg')
print('insert <data> at end')
print('remove <index>')
print('quit')

while True:
    print('The list: ', end = '')
    a_cdllist.display()
    print()
    do = input('What would you like to do? ').split()

    operation = do[0].strip().lower()

    if operation == 'insert':
        data = int(do[1])
        position = do[3].strip().lower()
        new_node = Node(data)
        suboperation = do[2].strip().lower()
        if suboperation == 'at':
```



```
            if position == 'beg':
                a_cdllist.insert_at_beg(new_node)
            elif position == 'end':
                a_cdllist.insert_at_end(new_node)
        else:
            index = int(position)
            ref_node = a_cdllist.get_node(index)
            if ref_node is None:
                print('No such index.')
                continue
            if suboperation == 'after':
                a_cdllist.insert_after(ref_node, new_node)
            elif suboperation == 'before':
                a_cdllist.insert_before(ref_node, new_node)
```



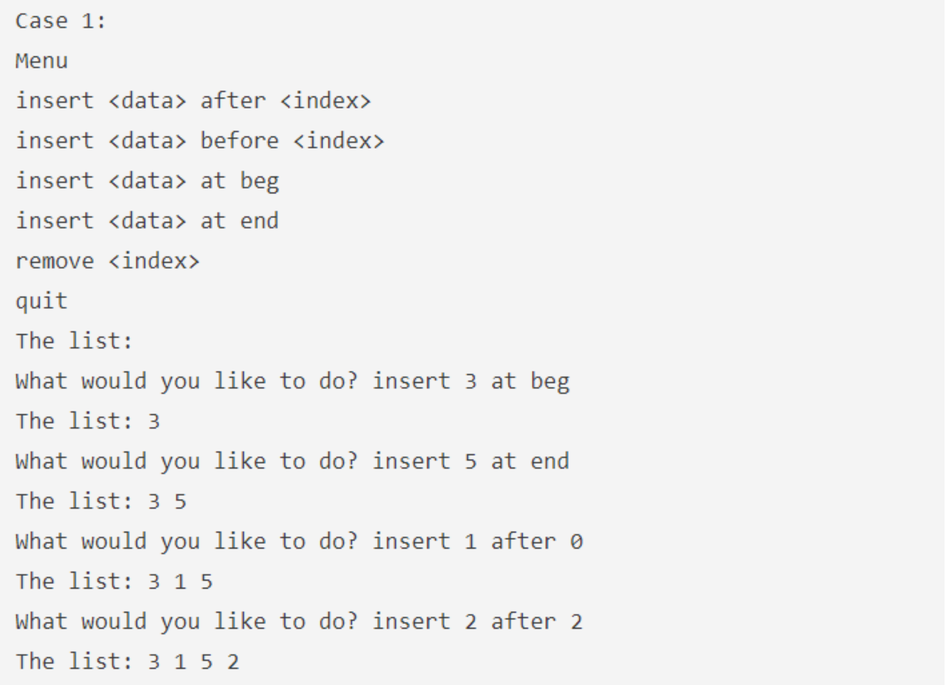
```
    elif operation == 'remove':
        index = int(do[1])
        node = a_cdllist.get_node(index)
        if node is None:
            print('No such index.')
            continue
        a_cdllist.remove(node)

    elif operation == 'quit':
        break
```

## Program Explanation

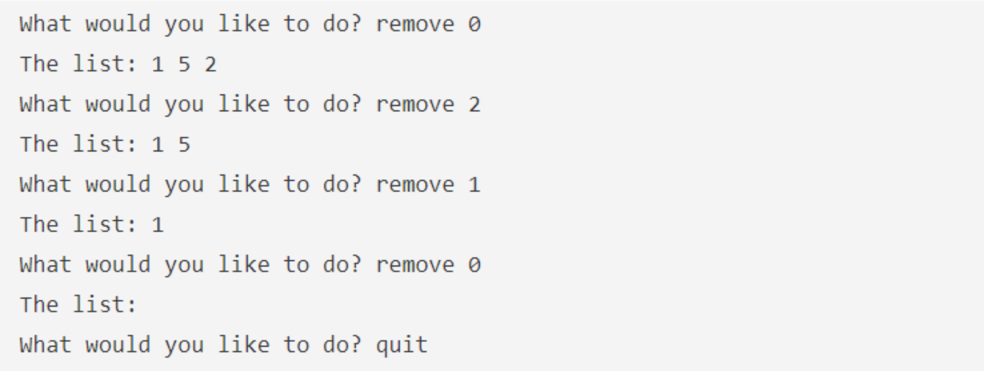
1. An instance of CircularDoublyLinkedList is created.

2. The user is presented with a menu to perform various operations on the list.

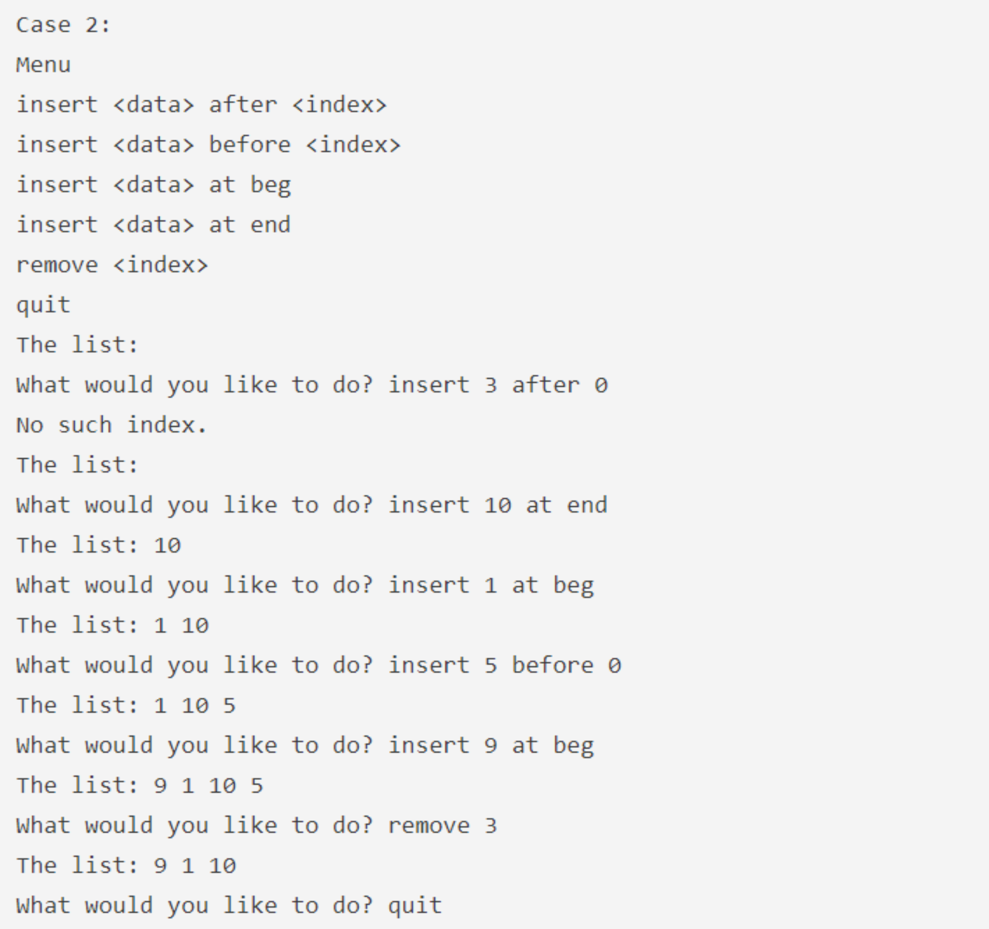
3. The corresponding methods are called to perform each operation.



```
Case 1:
Menu
insert <data> after <index>
insert <data> before <index>
insert <data> at beg
insert <data> at end
remove <index>
quit
The list:
What would you like to do? insert 3 at beg
The list: 3
What would you like to do? insert 5 at end
The list: 3 5
What would you like to do? insert 1 after 0
The list: 3 1 5
What would you like to do? insert 2 after 2
The list: 3 1 5 2
```



```
What would you like to do? remove 0
The list: 1 5 2
What would you like to do? remove 2
The list: 1 5
What would you like to do? remove 1
The list: 1
What would you like to do? remove 0
The list:
What would you like to do? quit
```



```
Case 2:
Menu
insert <data> after <index>
insert <data> before <index>
insert <data> at beg
insert <data> at end
remove <index>
quit
The list:
What would you like to do? insert 3 after 0
No such index.
The list:
What would you like to do? insert 10 at end
The list: 10
What would you like to do? insert 1 at beg
The list: 1 10
What would you like to do? insert 5 before 0
The list: 1 10 5
What would you like to do? insert 9 at beg
The list: 9 1 10 5
What would you like to do? remove 3
The list: 9 1 10
What would you like to do? quit
```