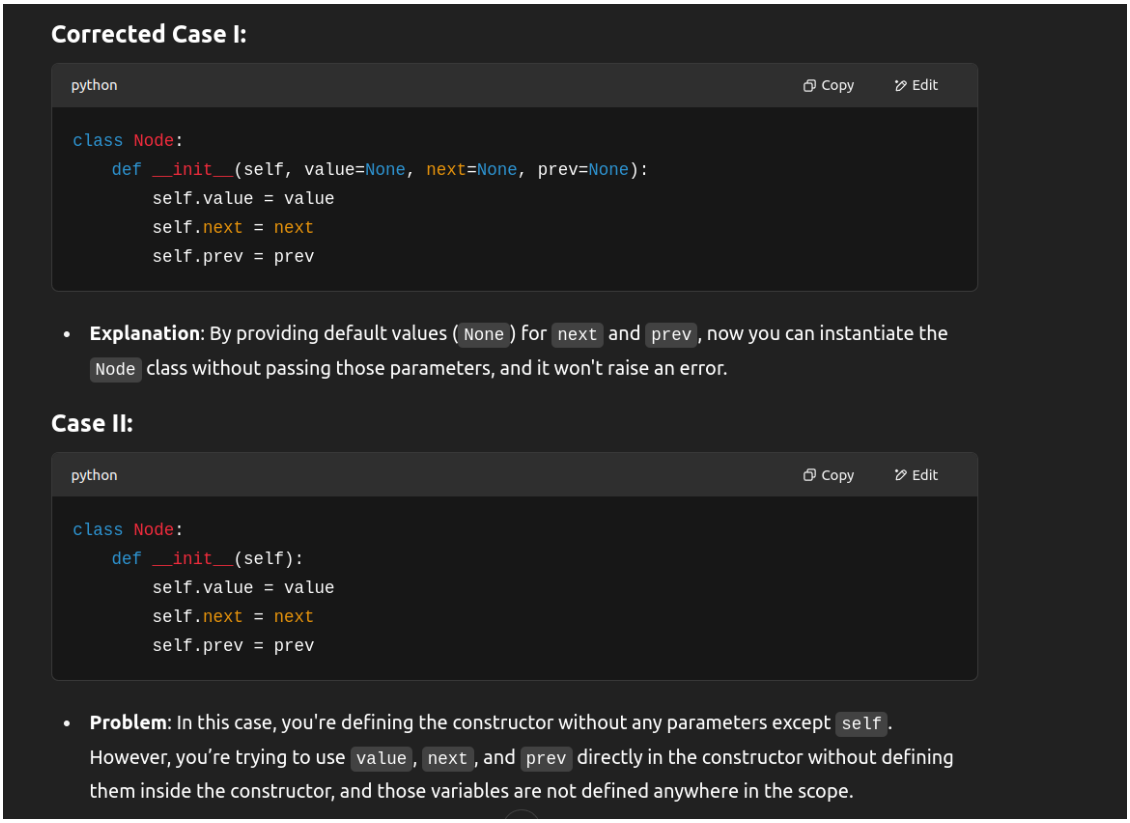
test

May 7, 2025

1 001 DLL Constructor.mp4

2 1. Constructor



Corrected Case I:

python Copy Edit

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

- **Explanation**: By providing default values (`None`) for `next` and `prev`, now you can instantiate the `Node` class without passing those parameters, and it won't raise an error.

Case II:

python Copy Edit

```
class Node:
    def __init__(self):
        self.value = value
        self.next = next
        self.prev = prev
```

- **Problem**: In this case, you're defining the constructor without any parameters except `self`. However, you're trying to use `value`, `next`, and `prev` directly in the constructor without defining them inside the constructor, and those variables are not defined anywhere in the scope.

2.0.1 Doubly Linked List (image) Visually:

Corrected Case I:

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

- **Explanation**: By providing default values (`None`) for `next` and `prev`, now you can instantiate the `Node` class without passing those parameters, and it won't raise an error.

Case II:

```python
class Node:
    def __init__(self):
        self.value = value
        self.next = next
        self.prev = prev
```

- **Problem**: In this case, you're defining the constructor without any parameters except `self`. However, you're trying to use `value`, `next`, and `prev` directly in the constructor without defining them inside the constructor, and those variables are not defined anywhere in the scope.

At first, we need to create a constructor;

- for that, we need to create a Node ||||| (Class)

This one is of singly linked list ———————————————————————- Doubly Linked List

Corrected Case I:

python Copy Edit

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

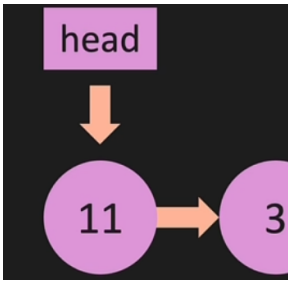
- **Explanation**: By providing default values (`None`) for `next` and `prev` , now you can instantiate the `Node` class without passing those parameters, and it won't raise an error.

Case II:

python Copy Edit

```python
class Node:
    def __init__(self):
        self.value = value
        self.next = next
        self.prev = prev
```

- **Problem**: In this case, you're defining the constructor without any parameters except `self` . However, you're trying to use `value` , `next` , and `prev` directly in the constructor without defining them inside the constructor, and those variables are not defined anywhere in the scope.



In dictnary wise, we can see this (in Doubly Linked List as)

Corrected Case I:

python Copy Edit

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

- **Explanation**: By providing default values (`None`) for `next` and `prev`, now you can instantiate the `Node` class without passing those parameters, and it won't raise an error.

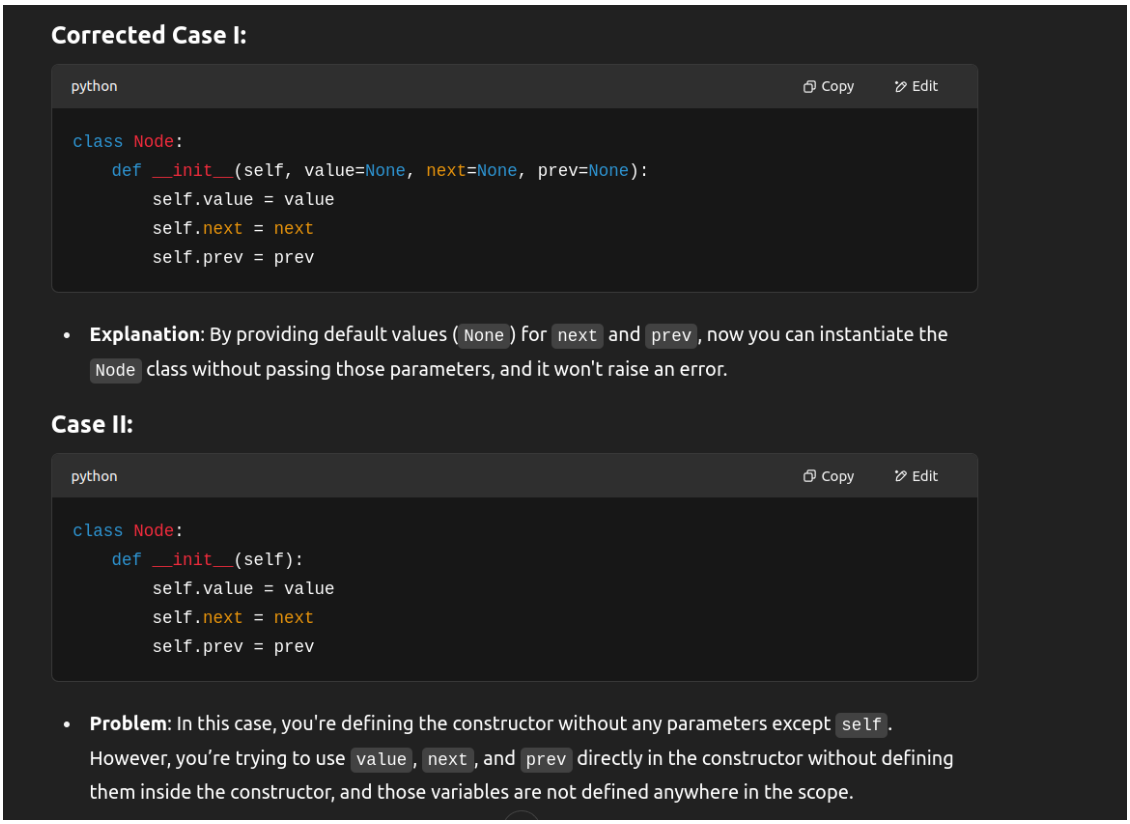
Case II:

python Copy Edit

```python
class Node:
    def __init__(self):
        self.value = value
        self.next = next
        self.prev = prev
```

- **Problem**: In this case, you're defining the constructor without any parameters except `self`. However, you're trying to use `value`, `next`, and `prev` directly in the constructor without defining them inside the constructor, and those variables are not defined anywhere in the scope.

2.0.2 Now building, DoublyLinkedList Class



Corrected Case I:

python Copy Edit

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

- **Explanation**: By providing default values (`None`) for `next` and `prev`, now you can instantiate the `Node` class without passing those parameters, and it won't raise an error.

Case II:

python Copy Edit

```python
class Node:
    def __init__(self):
        self.value = value
        self.next = next
        self.prev = prev
```

- **Problem**: In this case, you're defining the constructor without any parameters except `self`. However, you're trying to use `value`, `next`, and `prev` directly in the constructor without defining them inside the constructor, and those variables are not defined anywhere in the scope.

[81]:
```python
class Node:
    def __init__(self,value):
        self.value = value
        self.next = None
        self.prev = None

class DoublyLinkedList:
    def __init__(self, value):
        new_node = Node(value)
        self.head = new_node
        self.tail = new_node
        self.length = 1

    def print_list(self):
        temp = self.head
        while temp != None:
            print(temp.value)
            temp = temp.next

my_doubly_linked_list = DoublyLinkedList(7)
```

```
my_doubly_linked_list.print_list()
```

7

we have created a working constructor.

---

3 002 DLL Append.mp4

In singly linked list, when doing append;

We

- 1st create a node ;
- 2nd Point them ;
- 3rd Update Tail to it

Corrected Case I:

python Copy Edit

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

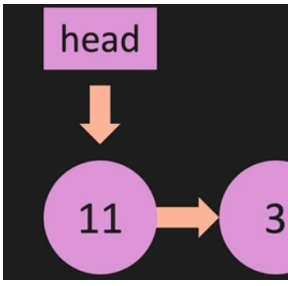
- **Explanation**: By providing default values (`None`) for `next` and `prev`, now you can instantiate the `Node` class without passing those parameters, and it won't raise an error.

Case II:

python Copy Edit

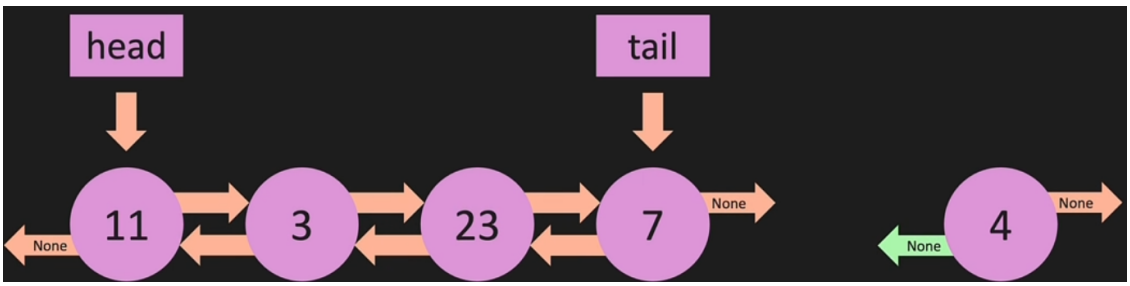
```
class Node:
    def __init__(self):
        self.value = value
        self.next = next
        self.prev = prev
```

- **Problem**: In this case, you're defining the constructor without any parameters except `self`. However, you're trying to use `value`, `next`, and `prev` directly in the constructor without defining them inside the constructor, and those variables are not defined anywhere in the scope.

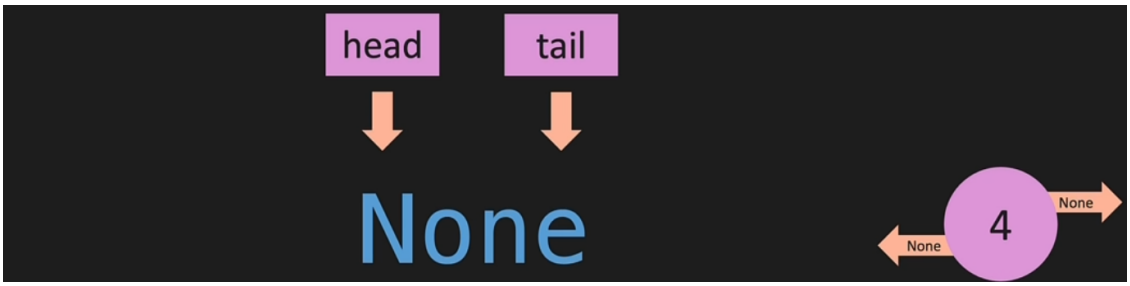


But in doubly linked list, when doing append:

We



Exception Case: (Edge Case)



[82]:
```python
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None
        self.prev = None

class DoublyLinkedList:
    def __init__(self, value):
        new_node = Node(value)
        self.head = new_node
        self.tail = new_node
        self.length = 1

    def print_list(self):
        temp = self.head
        while temp!=None:
            print(temp.value)
            temp = temp.next

    def append_list(self,value):
        new_node = Node(value)
        if self.length == 0 :
            self.head = new_node
            self.tail = new_node
        else:
            self.tail.next = new_node
```

```
            new_node.prev = self.tail
            self.tail = new_node
        self.length += 1
        return True

my_doubly_linked_list = DoublyLinkedList(7)

my_doubly_linked_list.append_list(14) ; my_doubly_linked_list.append_list(21)

my_doubly_linked_list.print_list()
```

```
7
14
21
```

Either you need to pass, the (fixed variable , value ; in that case you need to define it in the function constructor)

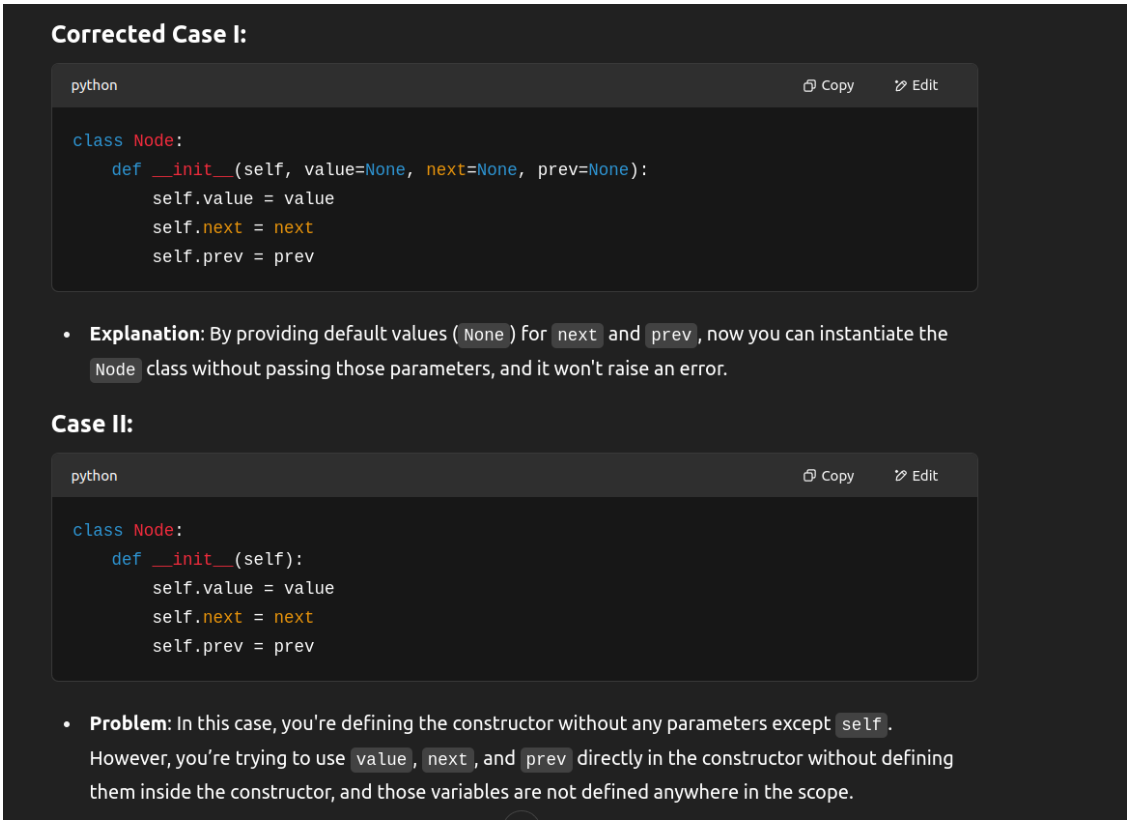
Else,

not fixed value, then you can put your parameter as None;

To fix this, you should set default values for next and prev, or you can provide a default value for value. Here's a corrected version:

Explanation:

By providing default values (None) for next and prev, now you can instantiate the Node class without passing those parameters, and it won't raise an error.



Why doesn't it give an error (and actually works as you expected)?

This might be a misunderstanding. If you're getting no error, that would be unusual because there should be an issue—since value, next, and prev are not defined anywhere in this code, Python should raise a NameError indicating that these variables are not defined. If no error is being raised, double-check to ensure that the code you're running actually matches what's written here.

[83]:
```
# Another way of creating the node ;

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

4 003 DLL Pop.mp4

[84]:
```
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None
        self.prev = None
```

```
class DoublyLinkedList:
    def __init__(self, value):
        new_node = Node(value)
        self.head = new_node
        self.tail = new_node
        self.length = 1

    def print_list(self):
        temp = self.head
        while temp!=None:
            print(temp.value)
            temp = temp.next

    def append_list(self,value):
        new_node = Node(value)
        if self.length == 0 :
            self.head = new_node
            self.tail = new_node
        else:
            self.tail.next = new_node
            new_node.prev = self.tail
            self.tail = new_node
        self.length += 1
        return True

    def pop(self):
        temp1 = self.head
        while temp1.next != None:
            temp2 = temp1
            temp1 = temp1.next

        self.tail = temp2
        temp2.next = None
        return temp1

my_doubly_linked_list = DoublyLinkedList(7)

my_doubly_linked_list.append_list(14) ; my_doubly_linked_list.append_list(21)

my_doubly_linked_list.append_list(28) ; my_doubly_linked_list.append_list(35)

my_doubly_linked_list.print_list()

print(f'-----------')

my_doubly_linked_list.pop()
my_doubly_linked_list.print_list()
```

```
7
14
21
28
35
-----------
7
14
21
28
```

5 004 DLL Prepend.mp4

[85]:
```python
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None
        self.prev = None

class DoublyLinkedList:
    def __init__(self, value):
        new_node = Node(value)
        self.head = new_node
        self.tail = new_node
        self.length = 1

    def print_list(self):
        temp = self.head
        while temp!=None:
            print(temp.value)
            temp = temp.next

    def append_list(self,value):
        new_node = Node(value)
        if self.length == 0 :
            self.head = new_node
            self.tail = new_node
        else:
            self.tail.next = new_node
            new_node.prev = self.tail
            self.tail = new_node
        self.length += 1
        return True

    def pop(self):
        temp1 = self.head
        temp2 = temp1
```

```
        while temp1.next != None:
            temp2 = temp1
            temp1 = temp1.next

        self.tail = temp2
        temp2.next = None
        return temp1

    def prepend(self, value):
        new_node = Node(value)
        temp = self.head
        temp.prev = new_node
        self.head = new_node
        self.head.next = temp




my_doubly_linked_list = DoublyLinkedList(7)

my_doubly_linked_list.append_list(14) ; my_doubly_linked_list.append_list(21)

my_doubly_linked_list.append_list(28) ; my_doubly_linked_list.append_list(35)

my_doubly_linked_list.print_list()

print(f'-----------')

my_doubly_linked_list.prepend(77)
my_doubly_linked_list.print_list()

print(f'-----------')
my_doubly_linked_list.pop()
my_doubly_linked_list.print_list()
```

```
7
14
21
28
35
-----------
77
7
14
21
28
35
```

```
----------
77
7
14
21
28
```

6 005 DLL Pop First.mp4

[86]:
```python
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None
        self.prev = None

class DoublyLinkedList:
    def __init__(self, value):
        new_node = Node(value)
        self.head = new_node
        self.tail = new_node
        self.length = 1

    def print_list(self):
        temp = self.head
        while temp!= None:
            print(temp.value)
            temp = temp.next

    def append_list(self, value):
        new_node = Node(value)
        if self.length == 0:
            self.head = new_node
            self.tail = new_node
        else:
            new_node.prev = self.tail # Because, since , upto here, self.tail␣
 ↪is pointing to the old node
            self.tail.next = new_node
            self.tail = new_node
        self.length += 1
        return True

    def pop(self):
        if self.length == 0 :
            return None
        elif self.length == 1:
            self.head = None
            self.tail = None
```

```
        else:
            temp1 = self.head
            temp2 = temp1
            while temp1.next != None:
                temp2 = temp1
                temp1 = temp1.next
            self.tail = temp2
            temp2.next = None
            return temp1

    def prepend(self, value):
        new_node = Node(value)
        if self.length == 0 :
            self.head = new_node
            self.tail = new_node
        else:
            new_node.next = self.head
            self.head.prev = new_node
            self.head = new_node
        return True

    def pop_first(self):
        if self.head is None:
            return None

        temp = self.head

        if temp.next is None:
            self.head = None
            self.tail = None
        else:
            self.head = self.head.next
            self.head.prev = None
            temp.next = None
        return temp
```

[87]:
```
my_doubly_linked_list = DoublyLinkedList(7)

my_doubly_linked_list.append_list(14) ; my_doubly_linked_list.append_list(21)

my_doubly_linked_list.append_list(28) ; my_doubly_linked_list.append_list(35)

my_doubly_linked_list.print_list()

print(f'-----------')

my_doubly_linked_list.prepend(77)
```

```
my_doubly_linked_list.print_list()

print(f'-----------')
my_doubly_linked_list.pop()
my_doubly_linked_list.print_list()

print(f'-----------')
my_doubly_linked_list.pop_first()
my_doubly_linked_list.print_list()
```

```
7
14
21
28
35
-----------
77
7
14
21
28
35
-----------
77
7
14
21
28
-----------
7
14
21
28
```

7 006 DLL Get.mp4

[88]:
```
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None
        self.prev = None

class DoublyLinkedList:
    def __init__(self, value):
        new_node = Node(value)
        self.head = new_node
```

```
        self.tail = new_node
        self.length = 1

    def print_list(self):
        temp = self.head
        while temp!= None:
            print(temp.value)
            temp = temp.next

    def append_list(self, value):
        new_node = Node(value)
        if self.length == 0:
            self.head = new_node
            self.tail = new_node
        else:
            new_node.prev = self.tail # Because, since , upto here, self.tail␣
↪is pointing to the old node
            self.tail.next = new_node
            self.tail = new_node
        self.length += 1
        return True

    def pop(self):
        if self.length == 0 :
            return None
        elif self.length == 1:
            self.head = None
            self.tail = None
        else:
            temp1 = self.head
            temp2 = temp1
            while temp1.next != None:
                temp2 = temp1
                temp1 = temp1.next
            self.tail = temp2
            temp2.next = None
            return temp1

    def prepend(self, value):
        new_node = Node(value)
        if self.length == 0 :
            self.head = new_node
            self.tail = new_node
        else:
            new_node.next = self.head
            self.head.prev = new_node
            self.head = new_node
```

```
        return True

    def pop_first(self):
        if self.head is None:
            return None

        temp = self.head

        if temp.next is None:
            self.head = None
            self.tail = None
        else:
            self.head = self.head.next
            self.head.prev = None
            temp.next = None
        return temp

    def get(self, index):
        temp = self.head
        for _ in range(index):
            temp = temp.next
        return temp.value
```

[89]:
```
my_doubly_linked_list = DoublyLinkedList(7)

my_doubly_linked_list.append_list(14) ; my_doubly_linked_list.append_list(21)

my_doubly_linked_list.append_list(28) ; my_doubly_linked_list.append_list(35)

my_doubly_linked_list.print_list()

print(f'-----------')

my_doubly_linked_list.prepend(77)
my_doubly_linked_list.print_list()

print(f'-----------')
my_doubly_linked_list.pop()
my_doubly_linked_list.print_list()

print(f'-----------')
my_doubly_linked_list.pop_first()
my_doubly_linked_list.print_list()

print(f'-----------')
my_doubly_linked_list.get(2)
```

```
7
14
21
28
35
-----------
77
7
14
21
28
35
-----------
77
7
14
21
28
-----------
7
14
21
28
-----------
```

[89]: 21

8 007 DLL Set.mp4

[90]:
```python
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None
        self.prev = None

class DoublyLinkedList:
    def __init__(self, value):
        new_node = Node(value)
        self.head = new_node
        self.tail = new_node
        self.length = 1

    def print_list(self):
        temp = self.head
        while temp!= None:
            print(temp.value)
```

```
            temp = temp.next

    def append_list(self, value):
        new_node = Node(value)
        if self.length == 0:
            self.head = new_node
            self.tail = new_node
        else:
            new_node.prev = self.tail # Because, since , upto here, self.tail␣
↪is pointing to the old node
            self.tail.next = new_node
            self.tail = new_node
        self.length += 1
        return True

    def pop(self):
        if self.length == 0 :
            return None
        elif self.length == 1:
            self.head = None
            self.tail = None
        else:
            temp1 = self.head
            temp2 = temp1
            while temp1.next != None:
                temp2 = temp1
                temp1 = temp1.next
        self.tail = temp2
        temp2.next = None
        self.length -=1
        return temp1

    def prepend(self, value):
        new_node = Node(value)
        if self.length == 0 :
            self.head = new_node
            self.tail = new_node
        else:
            new_node.next = self.head
            self.head.prev = new_node
            self.head = new_node
        self.length +=1
        return True

    def pop_first(self):
        if self.head is None:
            return None
```

```
        temp = self.head

        if temp.next is None:
            self.head = None
            self.tail = None
        else:
            self.head = self.head.next
            self.head.prev = None
            temp.next = None
        self.length -=1
        return temp

    def get(self, index):
        if index<0 or index>=self.length:
            return None
        else:
            temp = self.head
            for _ in range(index):
                temp = temp.next
            return temp.value

    def set(self, index, value):
        if index<0 or index>=self.length:
            return False
        else:
            temp = self.head
            for _ in range(index):
                temp = temp.next
            temp.value = value
        return True
```

[91]:
```
my_doubly_linked_list = DoublyLinkedList(7)

my_doubly_linked_list.append_list(14) ; my_doubly_linked_list.append_list(21)

my_doubly_linked_list.append_list(28) ; my_doubly_linked_list.append_list(35)

my_doubly_linked_list.print_list()

print(f'-----------')

my_doubly_linked_list.prepend(77)
my_doubly_linked_list.print_list()

print(f'-----------')
my_doubly_linked_list.pop()
```

```
my_doubly_linked_list.print_list()

print(f'-----------')
my_doubly_linked_list.pop_first()
my_doubly_linked_list.print_list()

print(f'-----------')
my_doubly_linked_list.set(2,55)
my_doubly_linked_list.print_list()
```

```
7
14
21
28
35
-----------
77
7
14
21
28
35
-----------
77
7
14
21
28
-----------
7
14
21
28
-----------
7
14
55
28
```

9 008 DLL Insert.mp4

[92]:
```
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None
        self.prev = None
```

```
class DoublyLinkedList:
    def __init__(self, value):
        new_node = Node(value)
        self.head = new_node
        self.tail = new_node
        self.length = 1

    def print_list(self):
        temp = self.head
        while temp!= None:
            print(temp.value)
            temp = temp.next

    def append_list(self, value):
        new_node = Node(value)
        if self.length == 0:
            self.head = new_node
            self.tail = new_node
        else:
            new_node.prev = self.tail # Because, since , upto here, self.tail␣
 ↪is pointing to the old node
            self.tail.next = new_node
            self.tail = new_node
        self.length += 1
        return True

    def pop(self):
        if self.length == 0 :
            return None
        elif self.length == 1:
            self.head = None
            self.tail = None
        else:
            temp1 = self.head
            temp2 = temp1
            while temp1.next != None:
                temp2 = temp1
                temp1 = temp1.next
        self.tail = temp2
        temp2.next = None
        self.length -=1
        return temp1

    def prepend(self, value):
        new_node = Node(value)
        if self.length == 0 :
```

```
            self.head = new_node
            self.tail = new_node
        else:
            new_node.next = self.head
            self.head.prev = new_node
            self.head = new_node
        self.length +=1
        return True

    def pop_first(self):
        if self.head is None:
            return None

        temp = self.head

        if temp.next is None:
            self.head = None
            self.tail = None
        else:
            self.head = self.head.next
            self.head.prev = None
            temp.next = None
        self.length -=1
        return temp

    def get(self, index):
        if index<0 or index>=self.length:
            return None
        else:
            temp = self.head
            for _ in range(index):
                temp = temp.next
            return temp.value

    def set(self, index, value):
        if index<0 or index>=self.length:
            return False
        else:
            temp = self.head
            for _ in range(index):
                temp = temp.next
            temp.value = value
        return True

    def insert(self, index, value):
        if index<0 or index>self.length:
            return False
```

```
        elif index==0:
            self.prepend(value)
        elif index==(self.length-1):
            self.append_list(value)
        else:
            new_node = Node(value)
            temp1 = self.head
            temp2 = temp1
            for _ in range(index):
                temp2 = temp1
                temp1 = temp1.next
            temp2.next = new_node
            new_node.next = temp1
            new_node.prev = temp2
            temp2.prev = new_node
            self.length +=1
        return True
```

[93]:
```
my_doubly_linked_list = DoublyLinkedList(7)

my_doubly_linked_list.append_list(14) ; my_doubly_linked_list.append_list(21)

my_doubly_linked_list.append_list(28) ; my_doubly_linked_list.append_list(35)

my_doubly_linked_list.print_list()

print(f'-----------')

my_doubly_linked_list.prepend(77)
my_doubly_linked_list.print_list()

print(f'-----------')
my_doubly_linked_list.pop()
my_doubly_linked_list.print_list()

print(f'-----------')
my_doubly_linked_list.pop_first()
my_doubly_linked_list.print_list()

print(f'-----------')
my_doubly_linked_list.insert(2,55)

my_doubly_linked_list.print_list()
```

```
7
14
21
```

```
28
35
-----------
77
7
14
21
28
35
-----------
77
7
14
21
28
-----------
7
14
21
28
-----------
7
14
55
21
28
```

10 009 DLL Remove.mp4

[94]:
```python
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None
        self.prev = None

class DoublyLinkedList:
    def __init__(self, value):
        new_node = Node(value)
        self.head = new_node
        self.tail = new_node
        self.length = 1

    def print_list(self):
        temp = self.head
        while temp!= None:
            print(temp.value)
            temp = temp.next
```

```
    def append_list(self, value):
        new_node = Node(value)
        if self.length == 0:
            self.head = new_node
            self.tail = new_node
        else:
            new_node.prev = self.tail # Because, since , upto here, self.tail␣
↪is pointing to the old node
            self.tail.next = new_node
            self.tail = new_node
        self.length += 1
        return True

    def pop(self):
        if self.length == 0 :
            return None
        elif self.length == 1:
            self.head = None
            self.tail = None
        else:
            temp1 = self.head
            temp2 = temp1
            while temp1.next != None:
                temp2 = temp1
                temp1 = temp1.next
        self.tail = temp2
        temp2.next = None
        self.length -=1
        return temp1

    def prepend(self, value):
        new_node = Node(value)
        if self.length == 0 :
            self.head = new_node
            self.tail = new_node
        else:
            new_node.next = self.head
            self.head.prev = new_node
            self.head = new_node
        self.length +=1
        return True

    def pop_first(self):
        if self.head is None:
            return None
```

```
        temp = self.head

        if temp.next is None:
            self.head = None
            self.tail = None
        else:
            self.head = self.head.next
            self.head.prev = None
            temp.next = None
        self.length -=1
        return temp

    def get(self, index):
        if index<0 or index>=self.length:
            return None
        else:
            temp = self.head
            for _ in range(index):
                temp = temp.next
            return temp.value

    def set(self, index, value):
        if index<0 or index>=self.length:
            return False
        else:
            temp = self.head
            for _ in range(index):
                temp = temp.next
            temp.value = value
        return True

    def insert(self, index, value):
        if index<0 or index>self.length:
            return False
        elif index==0:
            self.prepend(value)
        elif index==(self.length-1):
            self.append_list(value)
        else:
            new_node = Node(value)
            temp1 = self.head
            temp2 = temp1
            for _ in range(index):
                temp2 = temp1
                temp1 = temp1.next
            temp2.next = new_node
            new_node.next = temp1
```

```
            new_node.prev = temp2
            temp2.prev = new_node
            self.length +=1
        return True

    def remove(self, index):
        if index<0 or index>=self.length:
            return None
        elif index==0:
            self.pop_first()
        elif index == (self.length - 1):
            self.pop()
        else:
            temp1 = self.head
            temp2 = temp1
            for _ in range(index):
               temp2 = temp1
               temp1 = temp1.next
            temp2.next = temp1.next
            temp1.next.prev = temp2
            temp1.next = None
            self.length -=1
        return temp1
```

[95]:
```
my_doubly_linked_list = DoublyLinkedList(7)

my_doubly_linked_list.append_list(14) ; my_doubly_linked_list.append_list(21)

my_doubly_linked_list.append_list(28) ; my_doubly_linked_list.append_list(35)

my_doubly_linked_list.print_list()

print(f'-----------')

my_doubly_linked_list.prepend(77)
my_doubly_linked_list.print_list()

print(f'-----------')
my_doubly_linked_list.pop()
my_doubly_linked_list.print_list()

print(f'-----------')
my_doubly_linked_list.pop_first()
my_doubly_linked_list.print_list()

print(f'-------removing----')
my_doubly_linked_list.remove(2)
```

```
my_doubly_linked_list.print_list()
```

```
7
14
21
28
35
-----------
77
7
14
21
28
35
-----------
77
7
14
21
28
-----------
7
14
21
28
-------removing----
7
14
28
```

[96]:
```
# Corrected Version;

# Also, i am trying to use the extra variable

def remove(self, index):
    if index < 0 or index >= self.length:
        return None

    if index == 0:
        return self.pop_first()

    if index == self.length - 1:
        return self.pop()

    temp = self.head
    for _ in range(index):
        temp = temp.next
```

```
    temp.prev.next = temp.next
    temp.next.prev = temp.prev

    temp.next = None
    temp.prev = None

    self.length -= 1
    return temp
```

Assumptions:

This assumes you're working with a doubly linked list (since you're using .prev). If you're using a singly linked list, the logic is different — let me know if that's the case.

[ ]: