**Creating simple LinkedList in Python**

[[Kapil Lanjewar](https://medium.com/@kapil.lanjewar.100?source=post_page-----e6561d3b088b--------------------------------)](https://medium.com/@kapil.lanjewar.100?source=post_page-----e6561d3b088b--------------------------------)

[Kapil Lanjewar](https://medium.com/@kapil.lanjewar.100?source=post_page-----e6561d3b088b--------------------------------)

4 min read

·

1 day ago

*Node.py*

class Node:  
 def \_\_init\_\_(self, value):  
 self.value = value  
 self.next = None

*LinkedList.py*

from Node import Node  
  
  
class LinkedList:  
 def \_\_init\_\_(self, value=0):  
 if value != 0:  
 new\_node = Node(value)  
 self.head = new\_node  
 self.tail = new\_node  
 self.length = 1  
 else:  
 self.head = None  
 self.tail = None  
 self.length = 0  
  
 # O(n)  
 def print\_list(self):  
 temp = self.head  
 while temp is not None:  
 print(temp.value)  
 temp = temp.next  
  
 # O(1)  
 # Edge case - If the list is empty  
 def append(self, value):  
 new\_node = Node(value)  
 if self.length == 0:  
 self.head = new\_node  
 self.tail = new\_node  
 else:  
 self.tail.next = new\_node  
 self.tail = new\_node  
 self.length += 1  
 return True  
  
 # O(n)  
 # Edge cases:  
 # 1. If the list is empty  
 # 2. If list contains only one element  
 def pop(self):  
 if self.length == 0:  
 return None  
  
 first = self.head  
 second = None  
 while first.next is not None:  
 second = first  
 first = first.next  
  
 self.length -= 1  
  
 if self.length == 0:  
 second = None  
 self.head = None  
 self.tail = None  
 return first.value  
  
 self.tail = second  
 self.tail.next = None  
 second = None  
 return first.value  
  
 # O(1)  
 # Edge case - If the list is empty  
 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 = new\_node  
 self.length += 1  
 return True  
  
 # O(1)  
 # Edge cases:  
 # 1. If the list is empty  
 # 2. If list contains only one element  
 def pop\_first(self):  
 if self.length == 0:  
 return None  
  
 if self.length == 1:  
 temp = self.head  
 self.head = None  
 self.tail = None  
 self.length -= 1  
 return temp.value  
  
 temp = self.head  
 self.head = self.head.next  
 temp.next = None  
 self.length -= 1  
 return temp.value  
  
 # O(n)  
 # Edge cases:  
 # 1. If the list is empty  
 # 2. If specified index is out of bound  
 def get\_value(self, index):  
 if index < 0 or index >= self.length or self.length == 0:  
 return None  
  
 temp = self.head  
 for \_ in range(index):  
 temp = temp.next  
 return temp.value  
  
 # O(n)  
 # Edge cases:  
 # 1. If the list is empty  
 # 2. If specified index is out of bound  
 def \_\_get\_node(self, index):  
 if index < 0 or index >= self.length or self.length == 0:  
 return None  
  
 temp = self.head  
 for \_ in range(index):  
 temp = temp.next  
 return temp  
  
 # O(n)  
 # Edge cases - If specified index is out of bound  
 def set\_value(self, index, value):  
 temp = self.\_\_get\_node(index)  
 if temp:  
 temp.value = value  
 return True  
 return False  
  
 # O(1) for 1st and last index  
 # O(n) for indexes other than 1st and last index  
 # Edge cases:  
 # 1. If specified index is out of bound  
 # 2. If the list is empty then the index must be 0  
 def insert(self, index, value):  
 if index < 0 or index > self.length:  
 return False  
 if self.length == 0 and index != 0:  
 return False  
 if index == 0:  
 return self.prepend(value)  
 if index == self.length-1:  
 return self.append(value)  
  
 new\_node = Node(value)  
 temp = self.\_\_get\_node(index-1)  
 new\_node.next = temp.next  
 temp.next = new\_node  
 self.length += 1  
 return True  
  
 # O(1) for 1st and last index  
 # O(n) for indexes other than 1st and last index  
 # Edge cases:  
 # 1. If specified index is out of bound  
 # 2. If the list is empty  
 def remove(self, index):  
 if self.length == 0:  
 return None  
 if index < 0 or index > self.length:  
 return None  
 if index == 0:  
 return self.pop\_first()  
 if index == self.length-1:  
 return self.pop()  
  
 prev = self.\_\_get\_node(index-1)  
 temp = prev.next  
 prev.next = temp.next  
 temp.next = None  
 self.length -= 1  
 return temp.value  
  
 # O(n)  
 # Edge cases:  
 # 1. If list is empty  
 # 2. If list has only one element  
 def reverse(self):  
 if self.length == 0 or self.length == 1:  
 return False  
  
 first = self.head  
 second = first.next  
  
 first.next = None  
 self.tail = first  
  
 while second is not None:  
 third = second.next  
  
 if third is not None:  
 self.head = third  
 else:  
 self.head = second  
  
 second.next = first  
 first = second  
 second = third  
  
 return True  
  
  
# -------------------------------------------------------------->  
  
new\_linked\_list = LinkedList(4)  
  
print("===============> append operations")  
print(new\_linked\_list.append(5))  
print(new\_linked\_list.append(6))  
print(new\_linked\_list.append(7))  
new\_linked\_list.print\_list()  
print("Length ", new\_linked\_list.length)  
  
print("\n===============> remove operation")  
print("Removed item -", new\_linked\_list.remove(2))  
new\_linked\_list.print\_list()  
print("Length ", new\_linked\_list.length)  
  
print("\n===============> get\_value operation")  
print(new\_linked\_list.get\_value(2))  
  
print("\n===============> set\_value operation")  
print("Is set\_value operation successful -", new\_linked\_list.set\_value(2, 10))  
new\_linked\_list.print\_list()  
  
print("\n===============> insert operation")  
print("Is insert operation successful -", new\_linked\_list.insert(0, 1))  
print("Is insert operation successful -", new\_linked\_list.insert(4, 13))  
print("Is insert operation successful -", new\_linked\_list.insert(1, 3))  
new\_linked\_list.print\_list()  
  
print("\n===============> pop operation")  
print("Popped element - ", new\_linked\_list.pop())  
new\_linked\_list.print\_list()  
print("Length ", new\_linked\_list.length)  
  
print("\n===============> prepend operation")  
print(new\_linked\_list.prepend(9))  
new\_linked\_list.print\_list()  
print("Length ", new\_linked\_list.length)  
  
print("\n===============> pop\_first operation")  
print("Popped element - ", new\_linked\_list.pop\_first())  
new\_linked\_list.print\_list()  
print("Length ", new\_linked\_list.length)  
  
print("\n===============> reverse operation")  
print("Is reverse operation successful -", new\_linked\_list.reverse())  
new\_linked\_list.print\_list()  
print("Length ", new\_linked\_list.length)

*Output:*

*===============> append operations  
True  
True  
True  
4  
5  
6  
7  
Length 4*

*===============> remove operation  
Removed item — 6  
4  
5  
7  
Length 3*

*===============> get\_value operation  
7*

*===============> set\_value operation  
Is set\_value operation successful — True  
4  
5  
10*

*===============> insert operation  
Is insert operation successful — True  
Is insert operation successful — True  
Is insert operation successful — True  
1  
3  
4  
5  
10  
13*

*===============> pop operation  
Popped element — 13  
1  
3  
4  
5  
10  
Length 5*

*===============> prepend operation  
True  
9  
1  
3  
4  
5  
10  
Length 6*

*===============> pop\_first operation  
Popped element — 9  
1  
3  
4  
5  
10  
Length 5*

*===============> reverse operation  
Is reverse operation successful — True  
10  
5  
4  
3  
1  
Length 5*

[Python](https://medium.com/tag/python?source=post_page-----e6561d3b088b---------------python-----------------)

[Algorithms](https://medium.com/tag/algorithms?source=post_page-----e6561d3b088b---------------algorithms-----------------)

[Data Structures](https://medium.com/tag/data-structures?source=post_page-----e6561d3b088b---------------data_structures-----------------)