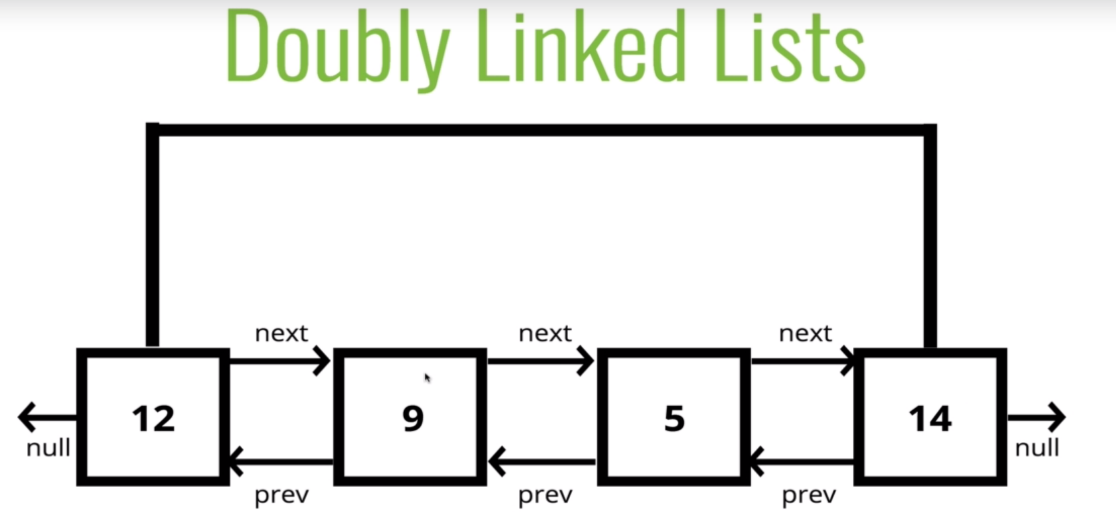
Doubly-Linked List Implementation

1. **class** Node:
2. **def** \_\_init\_\_(self, data):
3. self.data = data
4. self.next = None
5. self.prev = None
7. **class** DoublyLinkedList:
8. **def** \_\_init\_\_(self):
9. self.head = None
10. self.tail = None
11. self.length = 0
13. **def** append(self, data):
14. new\_node = Node(data)
15. **if** self.length == 0:
16. self.head = new\_node
17. self.tail = new\_node
18. **else**:
19. self.tail.next = new\_node
20. new\_node.prev = self.tail
21. self.tail = new\_node
22. self.length += 1
23. **return** self
25. **def** pop(self):
26. **if** **not** self.head:
27. **return** None
28. pop\_node = self.tail
29. **if** self.length == 1:
30. self.head = None
31. self.tail = None
32. **else**:
33. self.tail = pop\_node.prev
34. self.tail.next = None
35. pop\_node.prev = None
36. self.length -= 1
37. **return** pop\_node
39. **def** removeHead(self):
40. **if** self.length == 0:
41. **return** None
42. old\_head = self.head
43. **if** self.length == 1:
44. self.head = None
45. self.tail = None
46. **else**:
47. self.head = old\_head.next
48. self.head.prev = None
49. old\_head.next = None
50. self.length -= 1
51. **return** old\_head
53. **def** addHead(self, data):
54. new\_node = Node(data)
55. **if** self.length == 0:
56. self.head = None
57. self.tail = None
58. **else**:
59. self.head.prev = new\_node
60. new\_node.next = self.head
61. self.head = new\_node
62. self.length += 1
63. **return** self
65. **def** get(self, index):
66. **if** index < 0 **or** index >= self.length:
67. **return** None
68. count = 0
69. current = self.head
70. **if** index <= self.length / 2:
71. count = 0
72. current = self.head
73. **while** count != index:
74. current = current.next
75. count += 1
76. **else**:
77. count = self.length -1
78. current = self.tail
79. **while** count != index:
80. current = current.prev
81. count -= 1
82. **return** current
84. **def** set(self, index, data):
85. found\_node = self.get(index)
86. **if** found\_node != None:
87. found\_node.data = data
88. **return** True
89. **return** False
91. **def** insert(self, index, data):
92. **if** index < 0 **or** index > self.length:
93. **return** False
94. **if** index == 0:
95. **return** self.addHead(data)
96. **if** index == self.length:
97. **return** self.append(data)
98. new\_node = Node(data)
99. b4\_node = self.get(index-1)
100. after\_node = b4\_node.next
102. b4\_node.next = new\_node
103. new\_node.prev = b4\_node
105. new\_node.next = after\_node
106. after\_node.prev = new\_node
108. self.length += 1
109. **return** True

# Doubly Linked List Cheat Sheet



* Consists of a **head**, a **tail**, and has a **length** property
* Consists of **nodes**, and each node has a **data** **value** and a **pointer** to the previous and next node.
* Both the **head** and the **tail** point to None
* Analogy: skyscraper with only stairs to get from floor to floor (list has an elevator with indexes)

**Advantages over singly linked list**  
**1)** A DLL can be traversed in both forward and backward direction.  
**2)** The delete operation in DLL is more efficient if pointer to the node to be deleted is given.  
**3)**We can quickly insert a new node before a given node.  
In singly linked list, to delete a node, pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using previous pointer.

**Disadvantages over singly linked list**  
**1)** Every node of DLL Require extra space for an previous pointer. It is possible to implement DLL with single pointer though.  
**2)** All operations require an extra pointer previous to be maintained. For example, in insertion, we need to modify previous pointers together with next pointers. For example in following functions for insertions at different positions, we need 1 or 2 extra steps to set previous pointer.