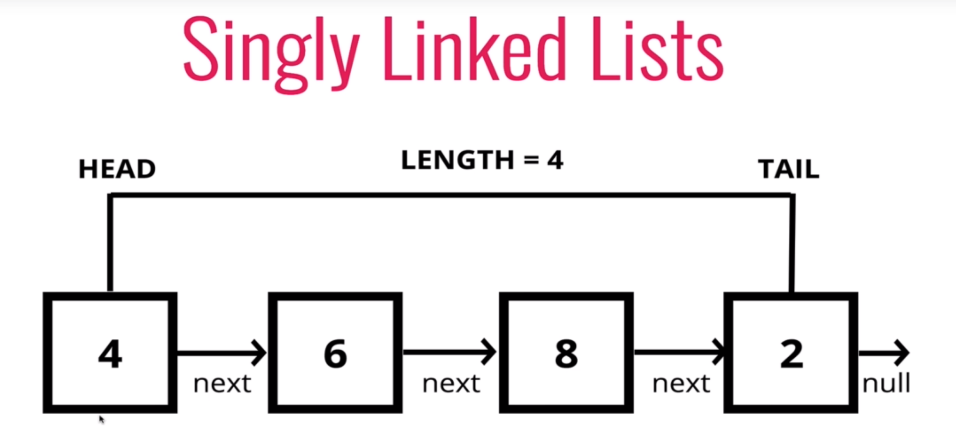
Singly-Linked List Implementation

1. **class** Node:
2. **def** \_\_init\_\_(self, data):
3. self.data = data
4. self.next = None
6. **class** LinkedList:
7. **def** \_\_init\_\_(self):
8. self.head = None
9. self.tail = None
10. self.length = 0
12. **def** append(self, data):
13. new\_node = Node(data)
14. **if** **not** self.head:
15. self.head = new\_node
16. self.tail = self.head
17. **else**:
18. self.tail.next = new\_node
19. self.tail = new\_node
20. self.length += 1
21. **return** self
23. **def** pop(self):
24. **if** **not** self.head:
25. **return** None
26. current = self.head
27. new\_tail = current
28. **while** current.next:
29. new\_tail = current
30. current = current.next
31. self.tail = new\_tail
32. self.tail.next = None
33. self.length -= 1
34. **if** self.length == 0:
35. self.head = None
36. self.tail = None
37. **return** current
39. **def** removeHead(self):
40. **if** **not** self.head:
41. **return** None
42. current\_head = self.head
43. self.head = current\_head.next
44. self.length -= 1
45. **if** self.length == 0:
46. self.tail = None
47. **return** current\_head
49. **def** addHead(self, data):
50. new\_node = Node(data)
51. **if** **not** self.head:
52. self.head = new\_node
53. self.tail = self.head
54. new\_node.next = self.head
55. self.head = new\_node
56. self.length += 1
57. **return** self
59. **def** get(self, index):
60. **if** index < 0 **or** index >= self.length:
61. **return** None
62. counter = 0
63. current = self.head
64. **while** counter != index:
65. current = current.next
66. counter += 1
67. **return** current
69. **def** set(self, index, data):
70. found\_node = self.get(index)
71. **if** found\_node:
72. found\_node.data = data
73. **return** True
74. **return** False
76. **def** insert(self, index, data):
77. **if** index < 0 **or** index > self.length:
78. **return** False
79. **if** index == self.length:
80. **return** self.append(data)
81. **if** index == 0:
82. **return** self.addHead(data)
83. new\_node = Node(data)
84. prev = self.get(index - 1)
85. temp = prev.next
86. prev.next = new\_node
87. new\_node.next = temp
88. self.length += 1
89. **return** True
91. **def** remove(self, index):
92. **if** index < 0 **or** index > self.length:
93. **return** None
94. **if** index == 0:
95. **return** self.removeHead()
96. **if** index == self.length - 1:
97. **return** self.pop()
98. previous\_node = self.get(index - 1)
99. removed = previous\_node.next
100. previous\_node.next = removed.next
101. self.length -= 1
102. **return** removed
104. **def** reverse(self):
105. node = self.head
106. self.head = self.tail
107. self.tail = node
108. prev = None
109. **for** i **in** range(self.length):
110. next = node.next
111. node.next = prev
112. prev = node
113. node = next
114. **return** self
116. **def** **print**(self):
117. li = ''
118. current = self.head
119. **while** current:
120. li = li + str(current.data) + '->'
121. current = current.next
122. **return** li

# 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 another node or None
* Analogy: skyscraper with only stairs to get from floor to floor (list has an elevator with indexes)

Linked Lists:

Python Lists:

* Do not have indexes
* Connected via nodes with a next pointer
* Random access is not allowed
* Indexed in order
* Insertion and deletion is expensive
* Can quickly be accessed at specific index

### Singly Linked Big O

* Insertion = O(1)
* Deletion = O(1)
* Searching = O(n)
* Access = O(n)