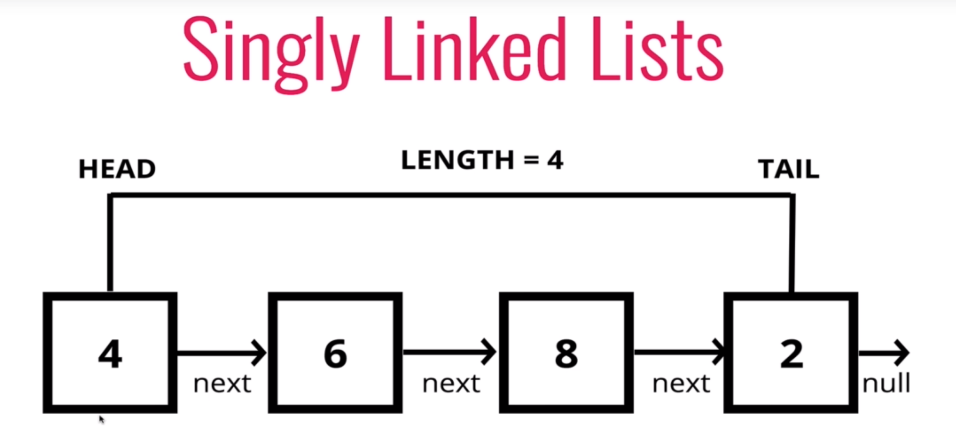
1. **class** Node:
2. # default value of data and next\_node is none if no data is passed
3. **def** \_\_init\_\_(self, data=None, next\_node=None):
4. self.data = data
5. self.next\_node = next\_node
7. # returns data of the node
8. **def** getData(self):
9. **return** self.data
11. # set the node data
12. **def** setData(self, data):
13. self.data = data
15. # get the next node
16. **def** getNext(self):
17. **return** self.next\_node
19. # set the next node
20. **def** setNext(self, node):
21. self.next\_node = node
23. **class** LinkedList:
24. **def** \_\_init\_\_(self):
25. self.head = None
27. # inserts a node at the head
28. **def** insertHead(self, data):
29. # create a temporary node
30. temp\_node = Node(data)
31. temp\_node.setNext(self.head)
32. self.head = temp\_node
33. **del** temp\_node
35. # inserts a node at the tail
36. **def** insertTail(self, data):
37. start = self.head
38. temp\_node = Node(data)
39. **while** start.getNext():
40. start = start.getNext()
41. start.setNext(temp\_node)
42. **del** temp\_node
43. **return** True
45. # prints the linked list
46. **def** **print**(self):
47. start = self.head
48. **if** start **is** None:
49. **print**("The linked list is currently empty.")
50. **return** False
52. **while** start:
53. **print**(str(start.getData()), end=" ")
54. start = start.next\_node
55. **if** start:
56. **print**("-->", end=" ")
57. **print**()
59. # gives the length of the linked list
60. **def** length(self):
61. start = self.head
62. size = 0
63. **while** start:
64. size += 1
65. start = start.getNext()
66. # print(size)
67. **return** size
69. # finds the index of the given data
70. **def** index(self, data):
71. start = self.head
72. index = 1
74. **while** start:
75. **if** start.getData() == data:
76. **return** index
77. **else**:
78. index += 1
79. start = start.getNext()
81. # removes an item from the linked list
82. # needs to check if data is in list---------------------------------------
83. **def** remove(self, item):
84. start = self.head
85. previous = None
86. found = False
88. # search element in list
89. **while** **not** found:
90. **if** start.getData() == item:
91. found = True
92. **else**:
93. previous = start
94. start = start.getNext()
96. # if previous is None then the data is found at first index
97. **if** previous **is** None:
98. self.head = start.getNext()
99. **else**:
100. previous.setNext(start.getNext())
101. **return** found
103. # finds the largest value in the linked list
104. **def** max(self):
105. start = self.head
106. largest = start.getData()
107. **while** start:
108. **if** largest < start.getData():
109. largest = start.getData()
110. start = start.getNext()
111. **return** largest
113. # gives the smallest value in the linked list
114. **def** min(self):
115. start = self.head
116. smallest = start.getData()
117. **while** start:
118. **if** smallest > start.getData():
119. smallest = start.getData()
120. start = start.getNext()
121. **return** smallest
123. # appends data to the end of the linked list
124. **def** append(self, data):
125. self.insertTail(data)
126. **return** True

129. # deletes and returns the tail of the linked list
130. **def** pop(self):
131. start = self.head
132. previous = None
134. **while** start.getNext():
135. previous = start
136. start = start.getNext()
138. **if** previous **is** None:
139. self.head = None
140. **else**:
141. previous.setNext(None)
142. data = start.getData()
143. **del** start
144. **return** data
146. # returns the index of the current node
147. **def** nodeIndex(self, index):
148. start = self.head
149. index = int(index)
150. pos = 1
151. **while** pos != index:
152. start = start.getNext()
153. pos += 1
155. data = start.getData()
156. **return** data
158. # copies the linked list
159. **def** copy(self):
160. temp = LinkedList()
161. start = self.head
163. temp.insertHead(start.getData())
164. start = start.getNext()
166. **while** start:
167. temp.insertTail(start.getData())
168. start = start.getNext()
170. **return** temp
172. # deletes and returns a node at the specified index
173. **def** popIndex(self, index):
174. data = self.nodeIndex(index)
175. self.remove(data)
176. **return** data
178. # counts how many of the given value are in a linked list
179. **def** count(self, element):
180. start = self.head
181. count1 = 0
182. **while** start:
183. **if** start.getData() == element:
184. count1 += 1
185. start = start.getNext()
186. **return** count1

189. # reverses the linked list
190. **def** reverse(self):
191. start = self.head
192. temp\_node = None
193. previous\_node = None
195. **while** start:
196. temp\_node = start.getNext()
197. start.setNext(previous\_node)
198. previous\_node = start
199. start = temp\_node
201. self.head = previous\_node
202. **return** True
204. # sorts the linked list
205. **def** sort(self):
206. start = self.head
207. begin\_node = start
208. **while** begin\_node:
209. temp\_node = begin\_node
210. temp\_node2 = begin\_node
211. smallest = begin\_node.getData()
212. **while** temp\_node:
213. **if** smallest > temp\_node.getData():
214. smallest = temp\_node.getData()
215. temp\_node2 = temp\_node
216. temp\_node = temp\_node.getNext()
218. # swap data of begin\_node and temp\_node2
219. temp = begin\_node.getData()
220. begin\_node.setData(temp\_node2.getData())
221. temp\_node2.setData(temp)
223. begin\_node = begin\_node.getNext()

# 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 elevat