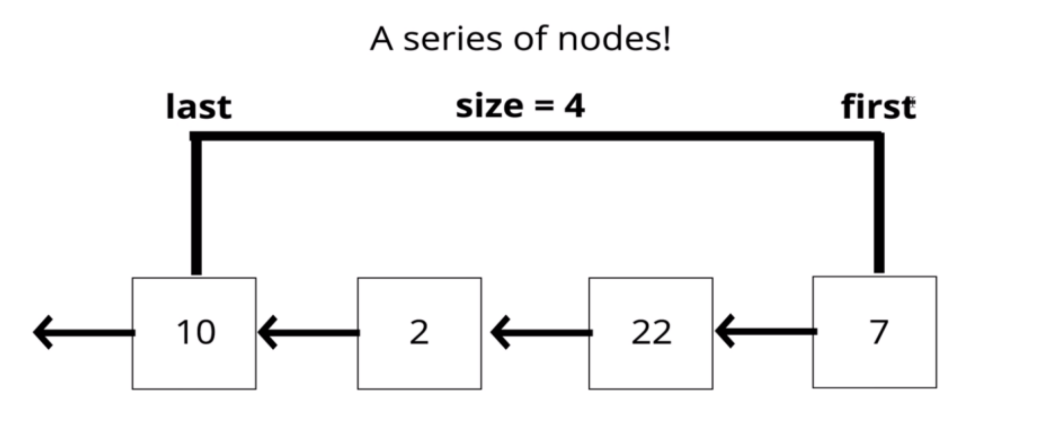
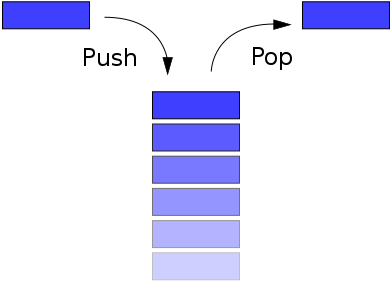
Stack Implementation

1. **class** Node:
2. **def** \_\_init\_\_(self, data):
3. self.data = data
4. self.next = None
6. **class** Stack:
7. **def** \_\_init\_\_(self):
8. self.first = None
9. self.last = None
10. self.length = 0
12. **def** append(self, data):
13. node = Node(data)
14. **if** **not** self.first:
15. self.first = node
16. self.last = node
17. **else**:
18. temp = self.first
19. self.first = node
20. self.first.next = temp
21. self.length += 1
22. **return** self
24. **def** pop(self):
25. **if** **not** self.first:
26. **return** None
27. temp = self.first
28. **if** self.first == self.last:
29. self.last = None
30. self.first = self.first.next
31. self.length -= 1
32. **return** temp.data



* Abides by **LIFO** (last in first out), **last element** added to the stack will be the **first removed** from the stack.
* Can use a list for a stack. list.append() to add onto the end and li.pop() to remove the item.
* Analogy: pile of dirty plates stacked up, the one on top will be removed first!



### Stack Big O

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