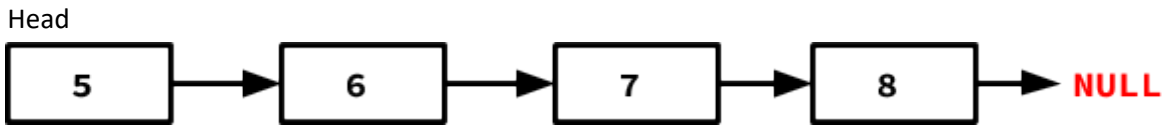
1. Write a function getAverage(Head) that will take head of a singly linked list and return the average.



Here the call getAverage(Head) will return 8.667

2. Write a function numOfOccurrences(Head, value) that will take Head of a singly linked list and return the number of occurrences of the value in the list.

3. You're given the pointer to the head nodes of two singly linked lists. Compare the data in the nodes of the linked lists to check if they are equal. The lists are equal only if they have the same number of nodes and corresponding nodes contain the same data.

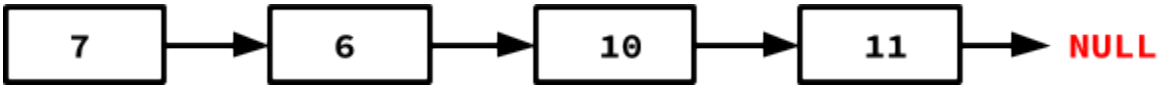


Figure: List A

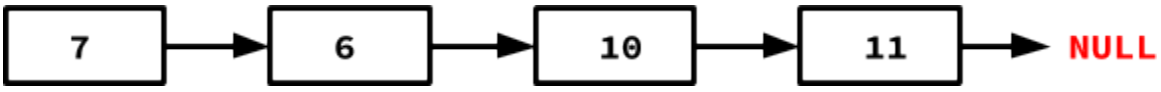


Figure: List B

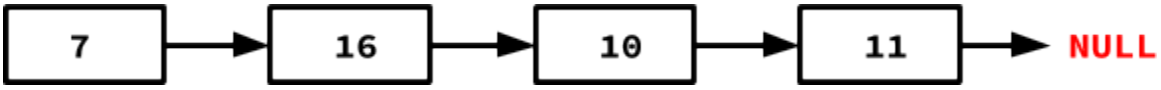


Figure: List C

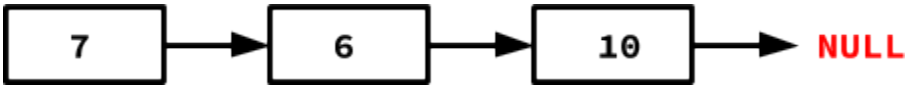
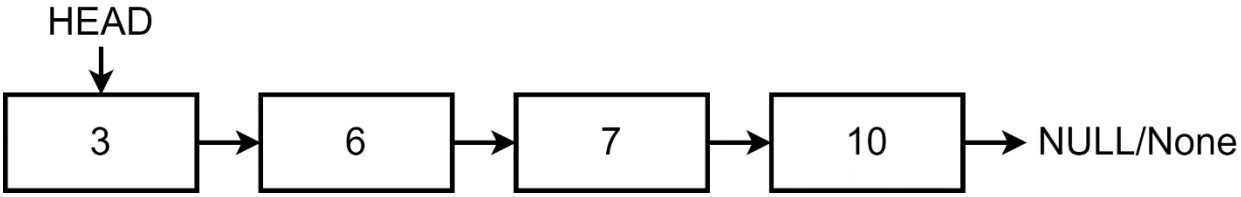


Figure: List D

For example a function call to equalNode(A,B) will return **true**, equalNode(A,A) will return **true**, equalNode(A,C) will return **false**, equalNode(A,D) will return **false**.

4. Assume, Head is the head-node of a singly linked list (**sll**). Write a utility function addNodeBeforeValue (self, givenValue, newValue) to insert a new value in this **sll** just

before the given value.



a. If the given value is found in the sll, insert the new value just before the given value.
b. If the given value is not found, don't add the node. Just print "Not found"

5. Remove the even numbers from the following singly linked list. Write a utility function deleteeven(self) that will delete all even nodes from the singly linked list.

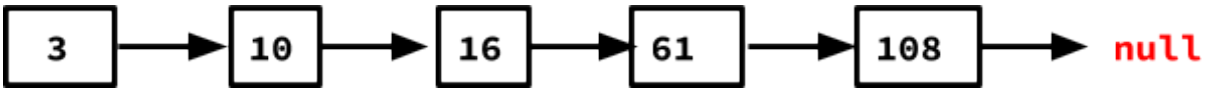
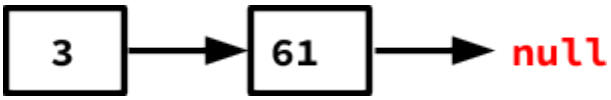
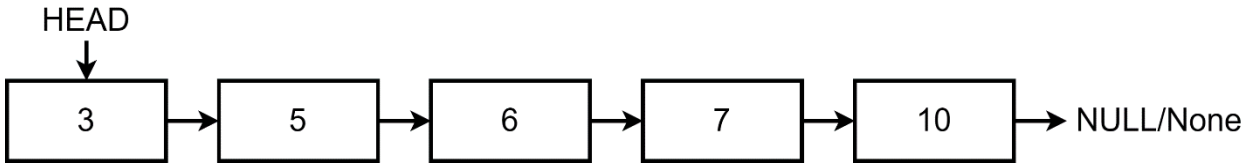


Fig A: List A



6. Write a **utility** function function named sorted_insert(self, v) that takes a parameter v and inserts it into a singly linked list in ascending order. For example, if the list contains the following items:

then, after calling sorted_insert(5), it will be as follows:



More specifically, complete the sorted_insert function in the following code.

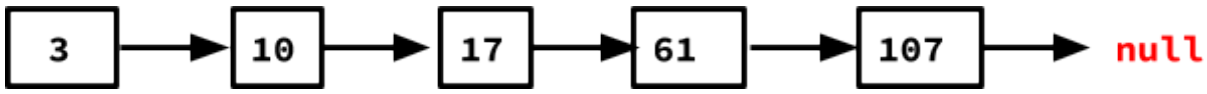
```
class Node:
  def __init__(self, d):
    self.data = d
    self.next = None

class LinkedList:
  def __init__(self):
    self.head = None

  def sorted_insert(self, v):
    # write your code here
```

```
sll = LinkedList()
sll.sorted_insert(3)
sll.sorted_insert(6)
sll.sorted_insert(10)
sll.sorted_insert(7)
sll.sorted_insert(5)
sll.display()
```

7. Assume there is a singly linked list. HEAD is the pointer for the head node. Write a function reverseList(HEAD) to reverse the linked list. It means, the function will return the new head.

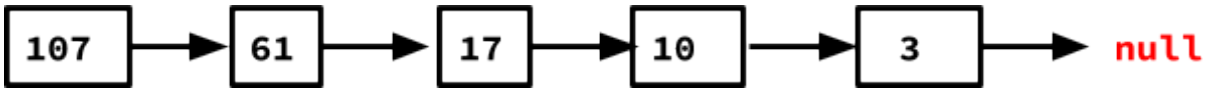
Then print the reversed list.

Fig A: List A



Fter calling reverseList(A) it will return B such that

Fig B: List B



For example if 3 represents the head node in Fig A, it will represent the tail in Fig B.

8. Assume, HEAD is the head-node of a doubly linked list (**dll**). Write a utility function addNodeAfterValue(self, givenValue, newValue) to insert a new value in this **dll** just after a given value.
   a. If the given value is found in the dll, insert the new value just after the given value.
   b. If the given value is not found, don't add the node. Just print "Not found"

9. Given a sorted doubly linked list and a value to insert, write a function to insert the value in a sorted way (and in an efficient way). Initial doubly linked list

After calling insertSorted(9) the list will be as follows: