**Assignment**

1\

class Node:  
def \_\_init\_\_(self, data=None, next=None):  
self.data = data  
self.next = next  
  
  
class SinglyLinkedList:  
def \_\_init\_\_(self):  
self.\_head = None  
self.\_size = 0  
  
def is\_empty(self):  
 return self.\_size == 0  
  
def \_\_len\_\_(self):  
 return self.\_size  
  
def add(self, item):  
new\_node = Node(item)  
new\_node.next = self.\_head  
self.\_head = new\_node  
self.\_size += 1  
  
def equals(self, other\_list):  
 if len(self) != len(other\_list):  
 return False  
  
 current\_self = self.\_head  
current\_other = other\_list.\_head  
  
 while current\_self is not None:  
 if current\_self.data != current\_other.data:  
 return False  
  
 current\_self = current\_self.next  
current\_other = current\_other.next  
  
 return True

2\

class Node:  
 def \_\_init\_\_(self, data=None):  
 self.data = data  
 self.next = None  
  
 def find\_second\_to\_last(head):  
 if head is None or head.next is None:  
 return None # List doesn't have enough nodes  
  
 current = head  
 previous = None  
  
 while current.next is not None:  
 previous = current  
 current = current.next  
  
 return previous

3\

class SinglyLinkedList:  
 def \_\_init\_\_(self):  
 self.\_head = None  
  
 def is\_empty(self):  
 return self.\_head is None  
  
 def add(self, item):  
 new\_node = Node(item)  
 new\_node.next = self.\_head  
 self.\_head = new\_node  
  
 def size(self):  
 count = 0  
 current = self.\_head  
  
 while current is not None:  
 count += 1  
 current = current.next  
  
 return count

4\

class SinglyLinkedList:  
 def \_\_init\_\_(self):  
 self.\_head = None  
  
 def is\_empty(self):  
 return self.\_head is None  
  
 def add(self, item):  
 new\_node = Node(item)  
 new\_node.next = self.\_head  
 self.\_head = new\_node  
  
 def remove\_first(self):  
 if self.is\_empty():  
 raise ValueError("Cannot remove from an empty list.")  
  
 removed\_item = self.\_head.data  
 self.\_head = self.\_head.next  
 return removed\_item  
  
 def rotate(self):  
 if self.is\_empty() or self.\_head.next is None:  
 return  
  
 old\_head = self.\_head  
 self.\_head = self.\_head.next  
  
 current = self.\_head  
 while current.next is not None:  
 current = current.next  
  
 current.next = old\_head  
 old\_head.next = None

5\

class Node:  
 def \_\_init\_\_(self, data=None):  
 self.data = data  
 self.next = None  
  
  
 def concatenate\_lists(L, M):  
 if L is None:  
 return M  
 if M is None:  
 return L  
  
 L\_prime = L  
 current = L\_prime  
  
 while current.next is not None:  
 current = current.next  
  
 current.next = M  
  
 return L\_prime

6\

class Node:  
 def \_\_init\_\_(self, data=None):  
 self.data = data  
 self.next = None  
  
  
 def reverse\_list(L):  
 if L is None or L.next is None:  
 return L  
  
 previous = None  
 current = L  
 next\_node = None  
  
 while current is not None:  
 next\_node = current.next  
 = previous  
 previous = current  
 current = next\_node  
  
 return previous