class Node():

def \_\_init\_\_(self, data):

self.data = data

self.next = None

def getData(self):

return self.data

def getNext(self):

return self.next

class LinkedList():

def \_\_init\_\_(self, head = None):

self.head = head

def insert(self, data):

if head == None:

head = Node(data)

else:

# insert at beginning of linked list

n = Node(data)

n.next = head

head = n

# insert at end of linked list

# n = Node(data)

# pointer = head

# while pointer:

# pointer = pointer.next

# pointer.next = n

def delete(self, data):

found = False

while head and not found:

if head.data == data:

found = True

head.next = head.next.next

else:

head = head.next

if head == None:

print "Not Found"

def length(self):

l = 0

while self.head:

l += 1

return l

#2.7

def intersect(ll1, ll2):

# if two linked lists are intersecting, then after the point of intersection,

# they have the same nodes, since the intersecting node cannot point to more than one node

# hence all nodes after intersection are same and hence the only difference in their lengths

# is before the intersecion

len1 = ll1.length()

len2 = ll2.length()

diff = abs(len2 - len1)

# seek ahead

while i < diff:

ll1 = ll1.next

while ll1 and ll2:

if ll1 == ll2:

return ll1

else:

ll1 = ll1.next

ll2 = ll2.next

if ll1 == None

#2.8

# very heavy on the system if large number of nodes are given

def loop(ll):

visited = []

current = ll

while current:

if current not in visited:

visited.append(current):

current = current.next

else:

break

if current == None:

print "No Loop"