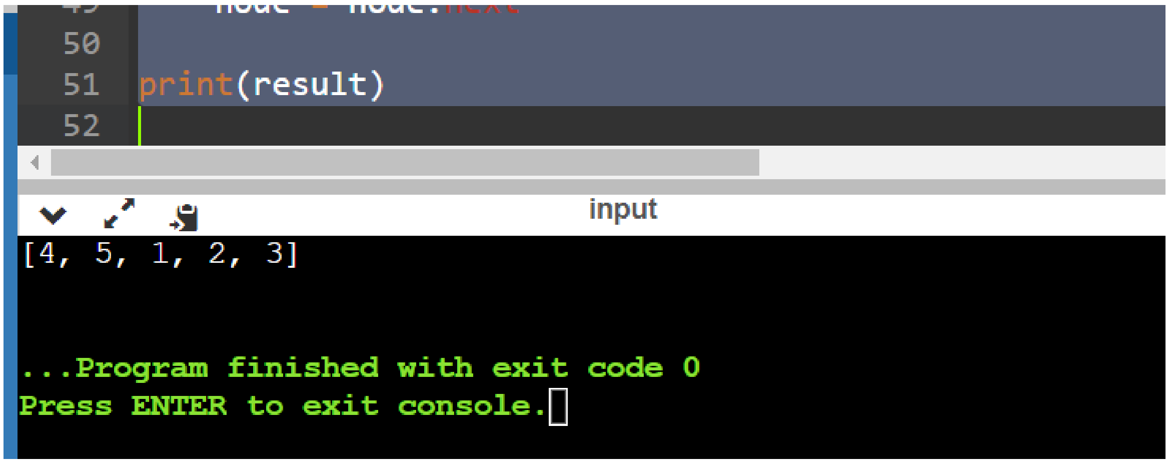
TANISH MITTAL
IT-C
2000290130174

Day – 7 Linked List & Arrays

Problem Statement: Given the head of a linked list, rotate the list to the right by k places.

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next


def rotateRight(head, k):
    if not head or not head.next or k == 0:
        return head



    length = 1
    tail = head
    while tail.next:
        tail = tail.next
        length += 1



    rotation_index = k % length


    if rotation_index == 0:
        return head



    new_tail = head
    for _ in range(length - rotation_index - 1):
        new_tail = new_tail.next
    new_head = new_tail.next
    new_tail.next = None
    tail.next = head

    return new_head

head = ListNode(1)
head.next = ListNode(2)
head.next.next = ListNode(3)
head.next.next.next = ListNode(4)
head.next.next.next.next = ListNode(5)

k = 2
rotated_head = rotateRight(head, k)



result = []
node = rotated_head
while node:
    result.append(node.val)
    node = node.next

print(result)
```

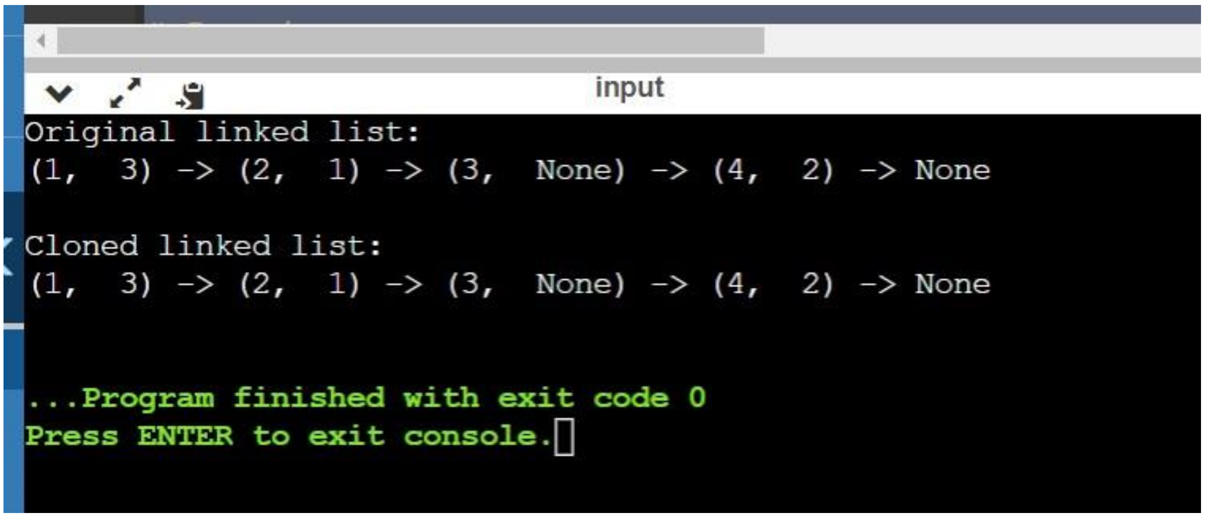


```
print(result)
```

```
[4, 5, 1, 2, 3]


...Program finished with exit code 0
Press ENTER to exit console.
```

Problem Statement: Given a Linked list that has two pointers in each node and one of which points to the first node and the other points to any random node. Write a program to clone the LinkedList.

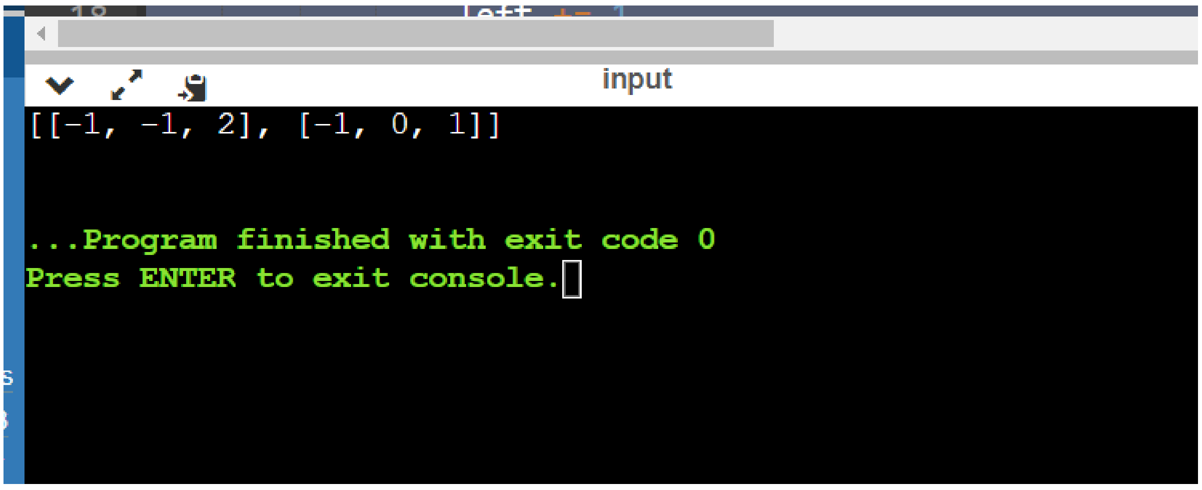
```
class Node:
    def __init__(self, val):
        self.val = val
        self.next = None
        self.random = None


def clone_linked_list(head):
    if not head:
        return None

    node_map = {}



    current = head
    while current:
        cloned_node = Node(current.val)
        node_map[current] = cloned_node
        current = current.next

    current = head
    while current:
        cloned_node = node_map[current]
        cloned_node.next = node_map.get(current.next, None)
        cloned_node.random = node_map.get(current.random, None)
        current = current.next
    return node_map[head]


def print_linked_list(head):
    current = head
    while current:
        random_val = current.random.val if current.random else None
        print(f"({current.val},  {random_val}) -> ", end="")
        current = current.next
    print("None")


    head = Node(1)
    node2 = Node(2)
    node3 = Node(3)
    node4 = Node(4)

    head.next = node2
    node2.next = node3
    node3.next = node4

    head.random  =  node3
    node2.random  =  head
    node3.random  =  None
    node4.random = node2


    cloned_head = clone_linked_list(head)


    print("Original linked list:")
    print_linked_list(head)
    print("\nCloned linked list:")
    print_linked_list(cloned_head)
```



```
Original linked list:
(1,  3) -> (2,  1) -> (3,  None) -> (4,  2) -> None

Cloned linked list:
(1,  3) -> (2,  1) -> (3,  None) -> (4,  2) -> None


...Program finished with exit code 0
Press ENTER to exit console.
```

Problem Statement: Given an array of N integers, your task is to find unique triplets that add up to give a sum of zero. In short, you need to return *an array of all the unique* triplets [arr[a], arr[b], arr[c]] such that i!=j, j!=k, k!=i, and their sum is equal to zero.

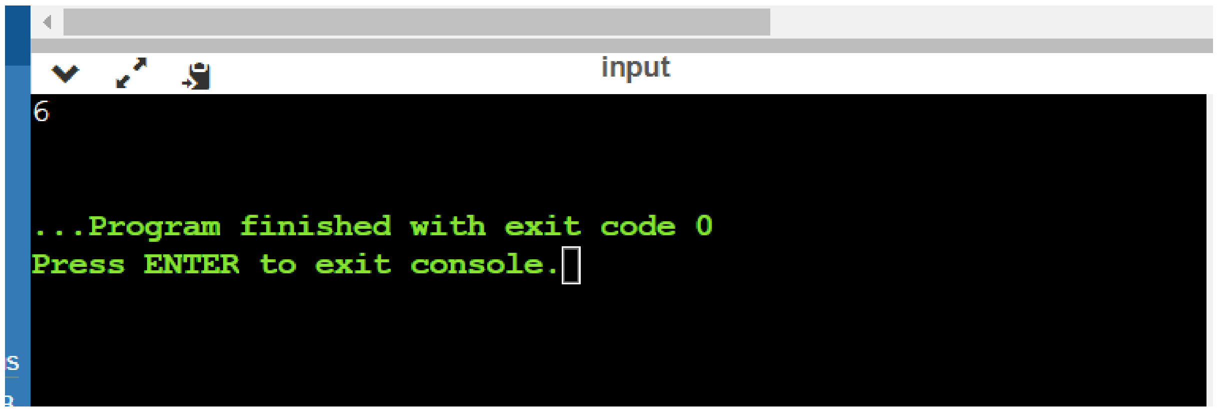
```
def threeSum(nums):
    nums.sort()
    result = []
    N = len(nums)

    for i in range(N - 2):
        if i > 0 and nums[i] == nums[i - 1]:
            continue

        left = i + 1
        right = N - 1

        while left < right:
            total = nums[i] + nums[left] + nums[right]
            if total == 0:
                result.append([nums[i], nums[left], nums[right]])
                left += 1
                right -= 1



                while left < right and nums[left] == nums[left - 1]:
                    left += 1
                while left < right and nums[right] == nums[right + 1]:
                    right -= 1

            elif total < 0:
                left += 1
            else:
                right -= 1

    return result
nums = [-1, 0, 1, 2, -1, -4]
print(threeSum(nums))
```



```
[[-1, -1, 2], [-1, 0, 1]]


...Program finished with exit code 0
Press ENTER to exit console.
```

Problem Statement: Given an array of non-negative integers representation elevation of ground. Your task is to find the water that can be trapped after rain.

```
def trap_water(height):
    left = 0
    right = len(height) - 1
    left_max = 0
    right_max = 0
    water_trapped = 0

    while left <= right:
        if height[left] <= height[right]:
            if height[left] > left_max:
                left_max = height[left]
            else:
                water_trapped += left_max - height[left]
            left += 1
        else:
            if height[right] > right_max:
                right_max = height[right]
            else:
                water_trapped += right_max - height[right]
            right -= 1

    return water_trapped
height = [0,1,0,2,1,0,1,3,2,1,2,1]
print(trap_water(height))
```

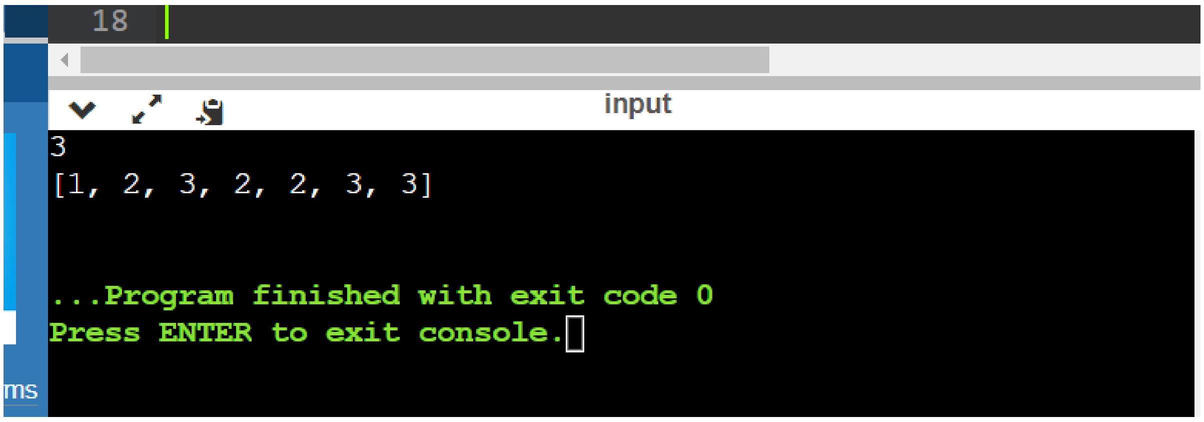


```
6


...Program finished with exit code 0
Press ENTER to exit console.
```

Problem Statement: Given an integer array sorted in non-decreasing order, remove the duplicates in place such that each unique element appears only once. The relative order of the elements should be kept the same.

If there are k elements after removing the duplicates, then the first k elements of the array should hold the final result. It does not matter what you leave beyond the first k elements.

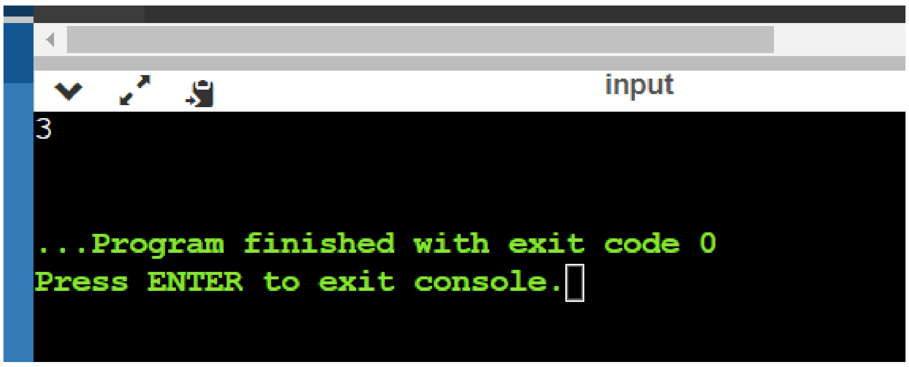
```
def removeDuplicates(arr):
    if len(arr) == 0:
        return 0

    k = 1  # Pointer to keep track of the position of the next unique element

    for i in range(1, len(arr)):
        if arr[i] != arr[k - 1]:
            arr[k] = arr[i]
            k += 1

    return k
arr = [1, 1, 2, 2, 2, 3, 3]
print(removeDuplicates(arr))  # Output: 3
print(arr)  # Output: [1, 2, 3, 2, 2, 3, 3]
```



```
3
[1, 2, 3, 2, 2, 3, 3]


...Program finished with exit code 0
Press ENTER to exit console.
```

Problem Statement: Given an array that contains **only 1 and 0** return the count of **maximum consecutive** ones in the array.

```
def find_max_consecutive_ones(nums):
    max_count = 0
    current_count = 0

    for num in nums:
        if num == 1:
            current_count += 1
            max_count = max(max_count, current_count)
        else:
            current_count = 0

    return max_count
prices = [1, 1, 0, 1, 1, 1]
print(find_max_consecutive_ones(prices))
```



```
3


...Program finished with exit code 0
Press ENTER to exit console.
```