









# Solution Review: First Non-Repeating Integer in a list

This review provides a detailed analysis of the different ways to find the first non-repeating integer in a list.



- Solution #1: Using a Python dictionary to keep count of repetitions
  - Time Complexity
- Solution #2: Using collections
  - Time Complexity

## Solution #1: Using a Python dictionary to keep count of repetitions #

```
def findFirstUnique(lst):
        counts = {} # Creating a dictionary
2
        # Initializing dictionary with pairs like (lst[i],count)
3
        counts = counts.fromkeys(lst, 0)
5
        for ele in lst:
            # counts[ele] += 1 # Incrementing for every repitition
6
7
            counts[ele] = counts[ele]+1
        answer key = None
8
        # filter first non-repeating
        for ele in lst:
10
            if (counts[ele] is 1):
11
                answer_key = ele
12
13
                break
14
        return answer_key
15
```

```
16
17 print(findFirstUnique([1, 1, 1, 2]))
18

Company the state of coding metrics for coding metrics in Foundation (in the parameter for coding metrics in the parameter for coding metrics
```

The *keys* in the **counts** dictionary are the elements of the given list and the *values* are the number of times each element appears in the list. We return the element that appears at most once in the list on **line 23**. We return the first non-repeating element in the list after traversing **lst**.

**Caveat** Note that Python dictionaries do not maintain the order that elements were added to them so this solution will not necessarily display the FIRST non-repeating integer when traversing the dictionary! To get around this, we can use Python's ordered dictionary as follows.

#### Time Complexity#

Since the list is only iterated over only twice and the counts dictionary is initialized with linear time complexity, therefore the time complexity of this solution is linear, i.e., O(n).

### Solution #2: Using collections#

```
import collections

def findFirstUnique(lst):
    orderedCounts = collections.OrderedDict() # Creating an ordered dictionary
    # Initializing dictionary with pairs like (lst[i],0)
    orderedCounts = orderedCounts.fromkeys(lst, 0)
    for ele in lst:
```

```
orderedCounts[ele] += 1 # Incrementing for every repitition
for ele in orderedCounts:
   if orderedCounts[ele] == 1:
      return ele
   return None

print(findFirstUnique([1, 1, 1, 2, 3, 2, 4]))
```

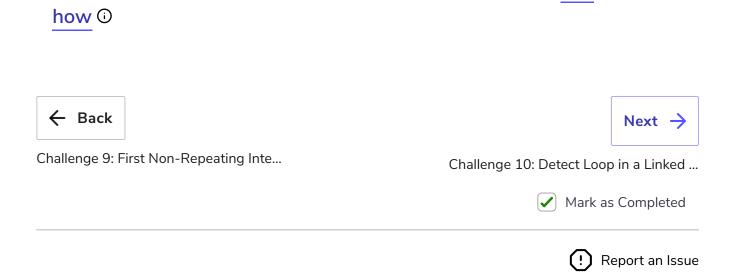
This solution is different from the previous as now the dictionary is maintained in a specific order in the orderedCounts variable.

Interviewing soon? We've partnered with Hired so that

companies apply to you instead of you applying to them. See

### Time Complexity#

Since the list is only iterated over only once, therefore the time complexity of this solution is linear, i.e., O(n).



X







