1. Fix the bugs v2

A student is trying to solve the following problem:
"Given an array of unique positive integers, find the number of pairs of integers that have a difference of exactly D".
Since each pair only needs to be counted once, he **correctly** reasons that the following pseudocode will work:

```
for each integer K in the array {
  if the array also contains K+D, pairs++
}
```

He wants to implement this by sorting the array and then scanning it with two indices: one that points to the integer K, and the other that searches forward for the integer K+D. Unfortunately, he gets distracted and his code doesn't work yet. Can you help him?

Language: Python 3 — Autocomplete Ready

```
#!/bin/python3 ...
# Preconditions:
#    array is sorted, smallest to largest, without duplicates

def countPairs(array, diff):
    """
    Returns the number of pairs in array that differ by diff.
    See problem stateeent for description of algorithm.
    array: list of unique integers
    diff: integer, the targeted difference
    return: number of pairs realizing the difference
    """
    pairs = 0
    n = len(array)
    i, j = 0, 1
    while j < len:
        for j in range(j, n):
            if array[j] == array[i] + diff:
                pairs += 1
    return pairs

if __name__ == '__main__': ...
```


1h 15m left

ALL

1

2

3

4

5

1. Find Doubles

This coding exercise is worth a total of 10 points, each test is worth one point.

Given a list of N integers, not necessarily unique, find all elements of the list for which there exists exactly one element of the list which is twice that number. The integers range from 0 to 1000. The list has no more than 100,000 elements.

Your code should find the appropriate values and print them to STDOUT in sorted order.

Examples:

`1 2 3 4 5 6 7 8 9 0 8 -> 0 1 2 3`
(8 is 4*2, but 8 is present twice; 0 is its own double, so it's part of the result)

`7 17 11 1 23 -> <blank>`
(nothing is exactly twice another element)

`1 1 2 -> 1 1`
(1 and 1 both have their double 2 present, and 2 is present in the list only once)

1h 8m left

ALL

1

2

3

4

2. Find Repetitions

This coding exercise is worth a total of 10 points, each test is worth one point.

You receive a short string `short_s` and a long string `long_s`, such that `short_s` can be found repeated a number of times in `long_s`. The goal is to compute the maximum number of *consecutive* repetitions of `short_s` within `long_s`, and return that number. If any of `short_s` or `long_s` are empty, then the answer is 0.

The following constraints apply:
0 <= len(short_s) < 10
0 <= len(long_s) < 1,000,000

Examples:
`short_s="AB" long_s="ABBAC"`
"AB" is only found once in "ABBAC" so the answer is 1.

`short_s="AB" long_s="ABCABCABAB"`
"AB" is found in long_s at index 0, 3, 6 and 8. Because it repeats twice consecutively at the end, the answer is 2.

2m 9s left

ALL

1

2

3

4

5

6

7

8

9

10

11

2. Racing Results

This coding exercise is worth a total of 10 points, each test is worth one point.

Unfortunately, the database containing all this year's racing results has crashed. The only thing left is a backup of database records and results for each race. **We urgently need to extract two pieces of information:**

- The winner of the championship and how many points they got so we can give them a trophy
- A list of racers who did not earn any point this year as they will be possibly relegated

Your program will receive input as a list of elements in the form of `[race, racer_name, position]`, where all elements are integers:

- `race` is a database record number between 2001 and 2000+N where N is the total number of races in the championship.
- `racer_name` is a database record number between 1001 and 1000+R where R is the total number of racers participating.
- `position` is a value between 1 (won the race) and R (arrived last).

Points are given according to the racers' position in a race, such that the 1st position is worth 10 points, 2nd is worth 6 points, 3rd is 4 points, 4th is 3 points, 5th is 2 points and 6th is worth 1 point. Positions further down earn no points.

In case of an equal number of points at the end of the championship, racers are sorted in record order for the purpose of classification. There are at most 100 racers, and at most 100 races in the championship.

Your program is expected to output two lines:

- one line with the record number of the winning racer (that is, the racer with the most points), followed by their point total
- one line with all record number of racers having earned zero points, in increasing order

Example:
Here's an example, with two races and three racers. The raw results are:

```
[2001, 1001, 3]
[2001, 1002, 2]
[2002, 1003, 1]
[2002, 1001, 2]
[2002, 1002, 3]
[2001, 1003, 1]
```

The two races are coded 2001 and 2002. The racers with records 1001, 1002 and 1003 competed.
Based on the raw results, they have the following points:
Racer 1001: 4+6=10 (3rd and 2nd positions)
Racer 1002: 6+4=10 (2nd and 3rd positions)
Racer 1003: 10+10=20 (1st in both races)

Your program is expected to output the following lines in that case:

```
1003 20
<empty line as no racers got 0 points>
```

Java 7

```
1 > import
10
11  class
12
13      /*
14       *
15       *
16       *
17       *
18
19      pu
20
21      }
22
23  }
24
25 > public
```

1. Remove All N Integers

Problem Statement:
Remove all elements having the value N from a singly linked list of integers. Return the head of the linked list.

Input format:
<N> <Linked List Size> <Linked List Elements>

Input sample *(Remove 3 from a linked list of size 5 containing elements 1, 3, 2, 3, 3)*:
3 5 1 3 2 3 3

Output expected:
1 2

```python
class Node:···

"""
For your reference, here's the node structure:
class Node:
    def __init__(self, val, next_node=None):
        self.val = val
        self.next_node = next_node
"""

def remove_all_n(head, n):
    if not head: return None
    # if head and not head.next: return None if head.val == n
    # Fill in the logic here
    prev = Node(None)
    prev.next_node = head
    ret = prev
    h = head
    while h:
        if h.val == n:
            prev.next_node = h.next_node
        else:
            prev.next_node = h
            prev = h
        h = h.next_node

    return ret.next_node

def main():···
```

Test Results **Custom Input**

21m left

ALL

2

3

4

5

6

7

8

9

7. Number Scores

You have developed a scoring system for positive integers that works as follows:

- +4 points for every 9 found in the number. For example, 9591 would score 8 points.
- +5 points for each pair of consecutive 1s. If there are more than two 1s in a row, add +5 for each additional 1, since it makes an additional pair (for example, four consecutive 1s gives +15).
- $+N^2$ points for a sequence of length N (N >= 1) where each digit is 1 more than the previous digit. For example, 9678562 (9-678-56-2) would be $1 + 3^2 + 2^2 + 1 = 15$ points.
- +1 if the entire number is a multiple of 7
- +2 for each even digit (note that 0 *is* even)

Each component of the score is evaluated separately, so a given digit may contribute to more than one component. For example, the number 789 would score 9 for the sequence of length 3, 2 for one even digit, and 4 for the '9' digit, for a total of 15 points.

Write a function compute_number_score that computes (and returns) a score for an integer passed to it. The number will be in the range 0<=*number*<1000000000.

Java 8

Autocomplete Ready

```
1  import java.io.*;
16
17     // Complete the compute_number_score function below.
18     static int compute_number_score(int number) {
19         if(number==0){
20             return 2;
21         }
22         // System.out.println( getDigitWiseScore(number) + " "  + countConsecutiveOnes(number) +" " +get(number));
23         return getDigitWiseScore(number)  + countConsecutiveOnes(number) +get(number);
24
25     }
26     public static int getDigitWiseScore(int num){
27         int score = 0;
28         if(num%7==0) score+=1;
```

Line: 16 Col: 1

Test Custom Run

22m left

ALL

1

2

3

4

5

6

7

8

8. Lock Use Analyzer

Suppose we want to monitor how locks are used in our system. As the first step, we log moments of acquire and release for each lock in the following format:

- ACQUIRE X
- RELEASE X

where X is some integer ID (1<=X<=1,000,000) of the lock.

All locks must be released in the reverse order of acquiring them; for example, this is a correct event sequence:

1. ACQUIRE 364
2. ACQUIRE 84
3. RELEASE 84
4. ACQUIRE 1337
5. RELEASE 1337
6. RELEASE 364

However, the following sequence violates this rule, because lock 84 is still acquired while releasing lock 364:

1. ACQUIRE 364
2. ACQUIRE 84
3. **RELEASE 364**

22m left

ALL

1

2

3

4

5

6

7

8

2. **RELEASE 84**

3. RELEASE 364

and it is as bad to acquire an already acquired lock (usually resulting in a deadlock):

1. ACQUIRE 364
2. ACQUIRE 84
3. **ACQUIRE 364**
4. RELEASE 364

Write a program that, given a list of **N (0<=N<=1,000,000)** lock acquire and release events (counting from 1), checks if there were any problems (acquire-release order violation, dangling acquired lock, acquiring a lock twice or releasing a free lock), and if so, tells the earliest time that could be detected. Note that there's no limit on how many nested locks may be acquired at any given moment.
More formally, you are given an array of strings where each string is either "ACQUIRE X" or "RELEASE X", where all **X**s are integers in the range [1..1000000].
Return:

- **0**, if there were no lock-related problems even after program termination
- **N+1**, if the only issue after program termination were dangling acquired locks
- **K**, in case event number **K** violated any of the principles (release a lock not acquired previously, acquire an already held lock OR violate lock acquire-release ordering).

Examples:
Input:

1. ACQUIRE 364

Pythor

1 >
10
11
12
13
14 >

Test
Resu

22m left

ALL

1

2

3

4

5

6

7

8

5. RELEASE 364

6. ACQUIRE 456

Output: 7 (upon terminating, not all locks were released, namely 123 and 456, but we can't know that until actually exiting)

Input:

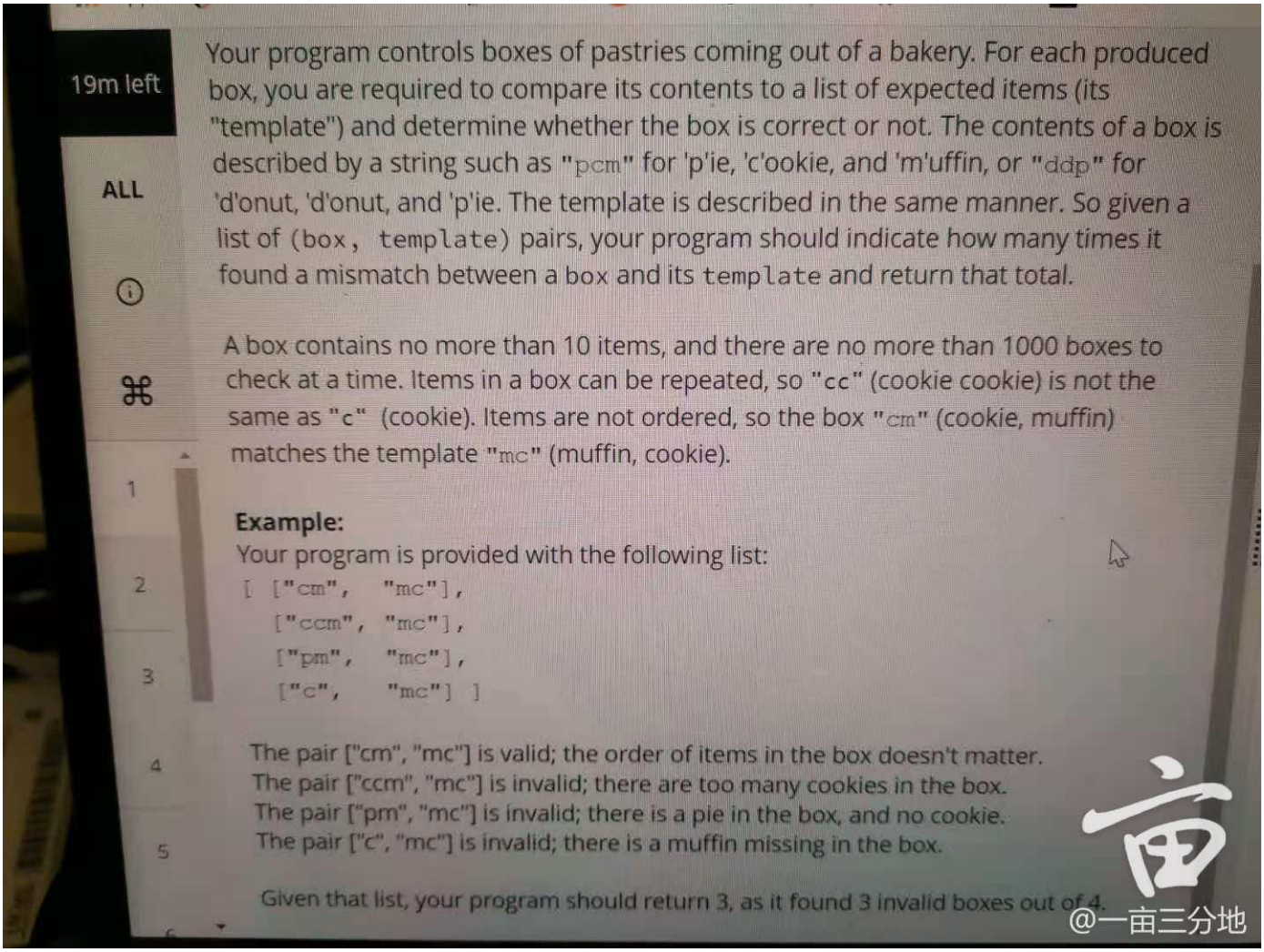
1. ACQUIRE 123
2. ACQUIRE 364
3. ACQUIRE 84
4. RELEASE 84
5. RELEASE 364
6. ACQUIRE 789
7. RELEASE 456
8. RELEASE 123

Output: 7 (releasing a lock not acquired before)

Input:

1. ACQUIRE 364
2. ACQUIRE 84
3. ACQUIRE 364
4. RELEASE 364

Output: 3 (acquiring an already held lock)



Push : begin S[i] = x; I = I + 1 end
Pop: begin i= I – 1; x = s[i] end
I = 1

Push : begin I = I + 1; S[i] = x end
Pop: begin x = s[i]; i= I – 1 end
I = 0

Push : begin S[i] := x ; I := I - 1;end
Pop: begin i:= I + 1; x := s[i] end
I = N

Push : begin I := I – 1; S[i] := x ; end
Pop: begin x := s[i] ; I := I + 1; end
I = N + 1