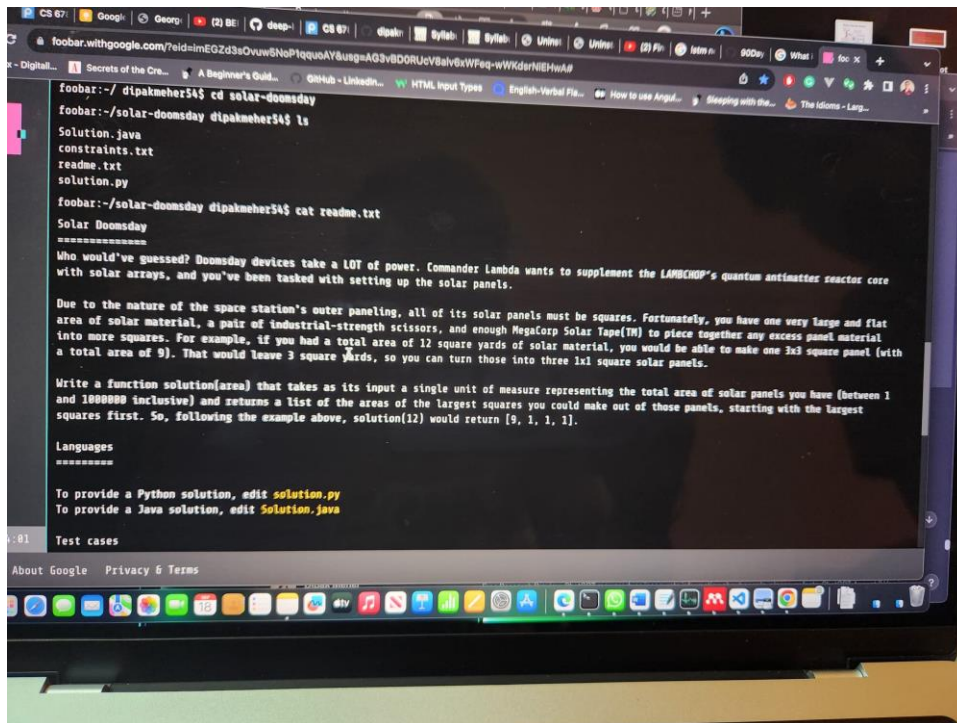
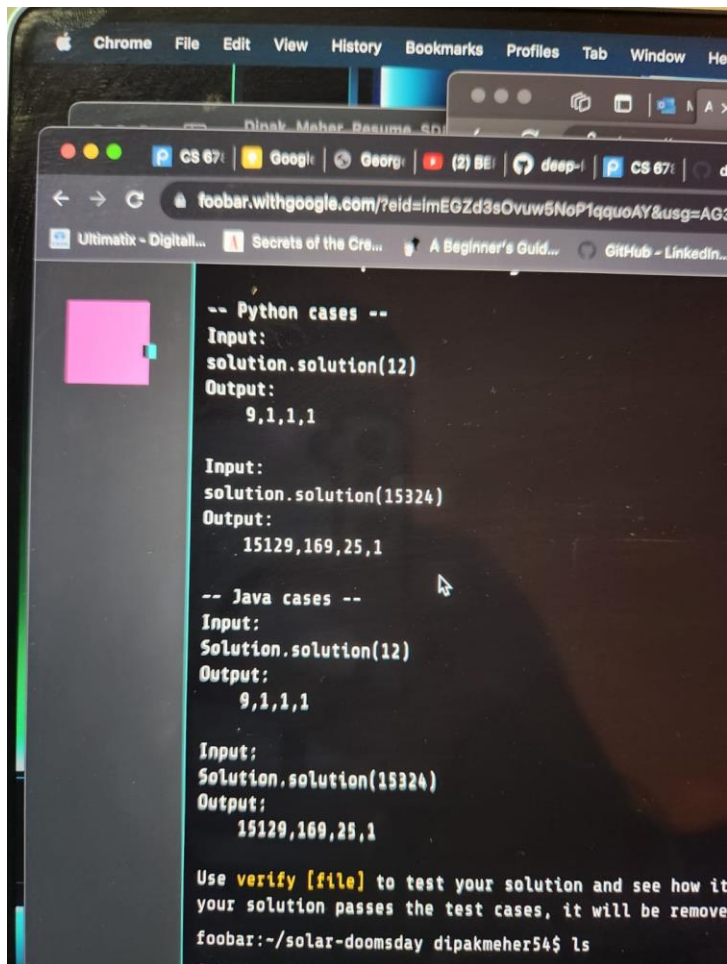


## Level 1:

### Task 1:





```
def calculate_square_tiles(area):
```

```
    square_sizes = []
```

```
    while area > 0:
```

```
        largest_square = int(area * 0.5) * 2
```

```
        square_sizes.append(largest_square)
```

```
        area -= largest_square
```

```
    return square_sizes
```

```
# Example usage:
```

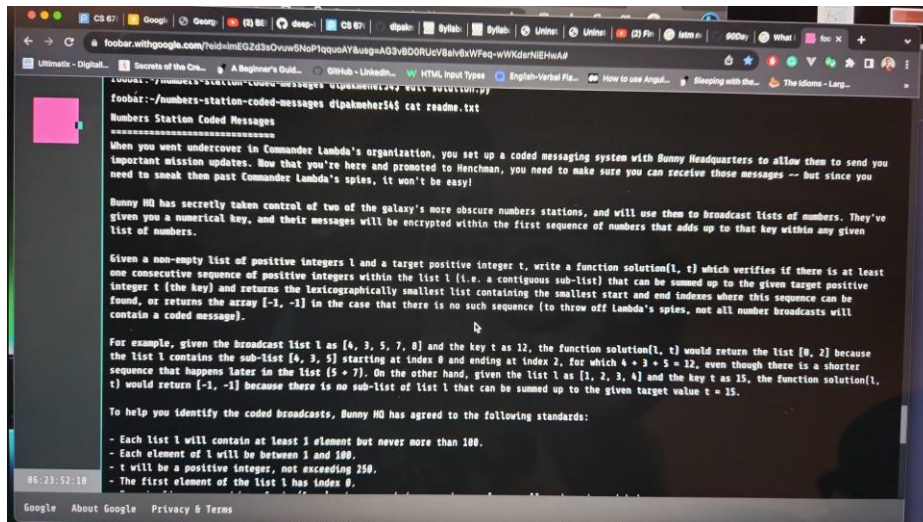
```
area = 12
```

```
tiles = calculate_square_tiles(area)
```

```
print(tiles) # Output: [9, 1, 1, 1]
```

Level 2:

Task 1:



foobär:~/numbers-station-coded-messages \$ cat readme.txt

**Numbers Station Coded Messages**

When you went undercover in Commander Lambda's organisation, you set up a coded messaging system with Bunny Headquarters to allow them to send you important mission updates. Now that you're here and promoted to Henchman, you need to make sure you can receive those messages -- but since you need to sneak them past Commander Lambda's spies, it won't be easy!

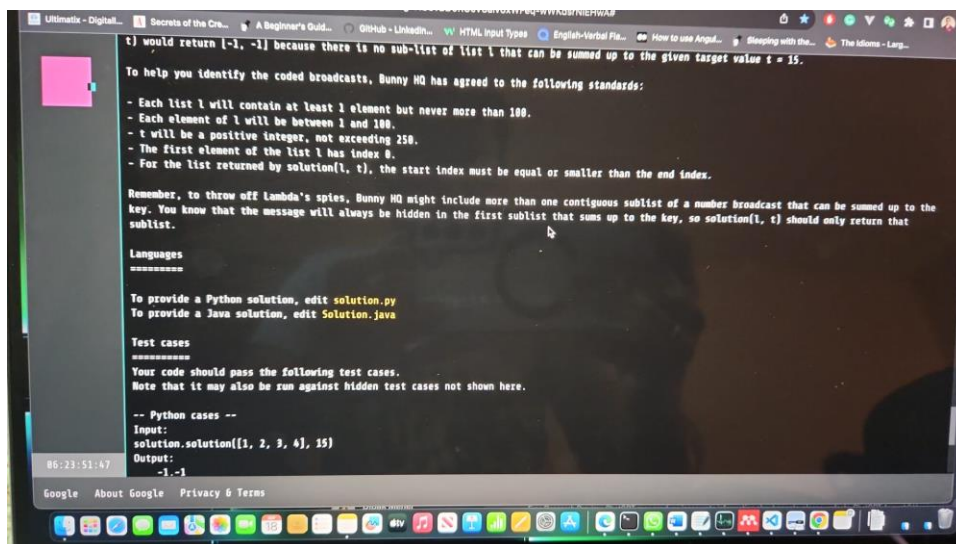
Bunny HQ has secretly taken control of two of the galaxy's more obscure numbers stations, and will use them to broadcast lists of numbers. They've given you a numerical key, and their messages will be encrypted within the first sequence of numbers that adds up to that key within any given list of numbers.

Given a non-empty list of positive integers *l* and a target positive integer *t*, write a function `solution(l, t)` which verifies if there is at least one consecutive sequence of positive integers within the list *l* (i.e. a contiguous sub-list) that can be summed up to the given target positive integer *t* (the key) and returns the lexicographically smallest list containing the smallest start and end indexes where this sequence can be found, or returns the array `[-1, -1]` in the case that there is no such sequence (to throw off Lambda's spies, not all number broadcasts will contain a coded message).

For example, given the broadcast list *l* as `[4, 3, 5, 7, 8]` and the key *t* as 12, the function `solution(l, t)` would return the list `[0, 2]` because the list *l* contains the sub-list `[4, 3, 5]` starting at index 0 and ending at index 2, for which `4 + 3 + 5 = 12`, even though there is a shorter sequence that happens later in the list `[5 + 7]`. On the other hand, given the list *l* as `[1, 2, 3, 4]` and the key *t* as 15, the function `solution(l, t)` would return `[-1, -1]` because there is no sub-list of list *l* that can be summed up to the given target value *t* = 15.

To help you identify the coded broadcasts, Bunny HQ has agreed to the following standards:

- Each list *l* will contain at least 1 element but never more than 100.
- Each element of *l* will be between 1 and 100.
- *t* will be a positive integer, not exceeding 250.
- The first element of the list *l* has index 0.



`t)` would return `[-1, -1]` because there is no sub-list of list *l* that can be summed up to the given target value *t* = 15.

To help you identify the coded broadcasts, Bunny HQ has agreed to the following standards:

- Each list *l* will contain at least 1 element but never more than 100.
- Each element of *l* will be between 1 and 100.
- *t* will be a positive integer, not exceeding 250.
- The first element of the list *l* has index 0.
- For the list returned by `solution(l, t)`, the start index must be equal or smaller than the end index.

Remember, to throw off Lambda's spies, Bunny HQ might include more than one contiguous sublist of a number broadcast that can be summed up to the key. You know that the message will always be hidden in the first sublist that sums up to the key, so `solution(l, t)` should only return that sublist.

**Languages**

=====

To provide a Python solution, edit `solution.py`  
To provide a Java solution, edit `Solution.java`

**Test cases**

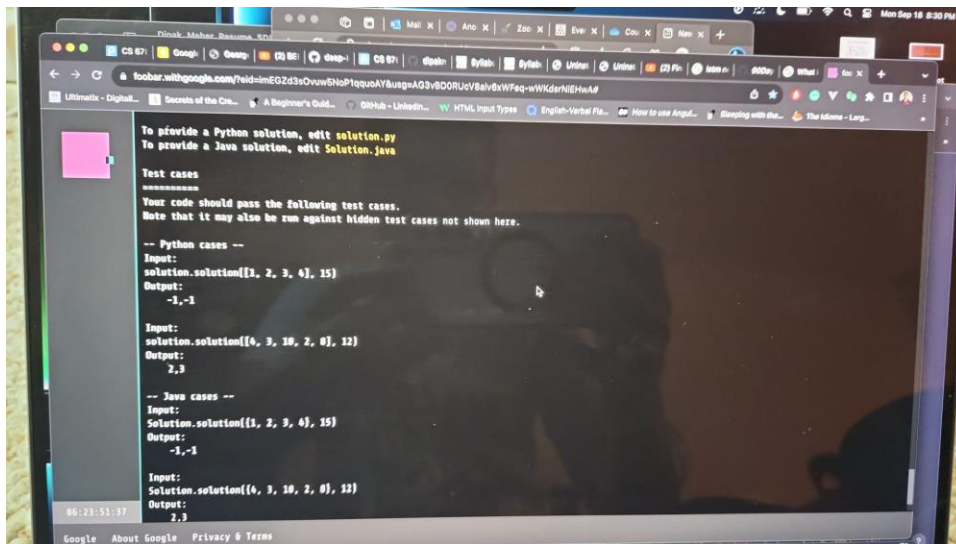
=====

Your code should pass the following test cases.  
Note that it may also be run against hidden test cases not shown here.

-- Python cases --

Input:  
`solution([1, 2, 3, 4], 15)`

Output:  
`[-1, -1]`



```
def find_sublist_with_sum(nums, key):
```

```
    left, right = 0, 0
```

```
    current_sum = 0
```

```
    while right < len(nums):
```

```
        current_sum += nums[right]
```

```
        while current_sum > key:
```

```
            current_sum -= nums[left]
```

```
            left += 1
```

```
        if current_sum == key:
```

```
            return [left, right]
```

```
        right += 1
```

```
    return [-1, -1]
```

# Example usage:

```
nums = [4, 3, 10, 2, 8]
```

```
key = 12
```

```
result = find_sublist_with_sum(nums, key)
```

```
print(result) # Output: [2, 3]
```

Task 2:

First calculate no of element in perfect binary tree using height. Find the parent element using below algorithm. H – height; q – list of labels who parents has to be returned

```
def solution(h, q):
```

```
    result = []
```

```
    for label in q:
```

```
        current = (2 ** h) - 1 # Start from the root
```

```
        parent = (2 ** (h - 1)) - 1 # Parent of the current node
```

```
        while current != label:
```

```
            if current == 1 or parent == label:
```

```
                break
```

```
            elif label > parent: # If label is on the right side, move to the left
```

```
                current = parent - 1
```

```
            else: # If label is on the left side, move to the right
```

```
                current = parent
```

```
            h -= 1
```

```
            parent = (parent - 1) // 2 # Calculate the parent of the current node
```

```
    if current == label:
```

```
        result.append(-1)
    else:
        result.append(parent)

    return result
```

# Example test cases

```
print(solution(3, [7, 3, 5, 1])) # Output: [-1, 7, 6, 3]
```

```
print(solution(5, [19, 14, 28])) # Output: [21, 15, 29]
```