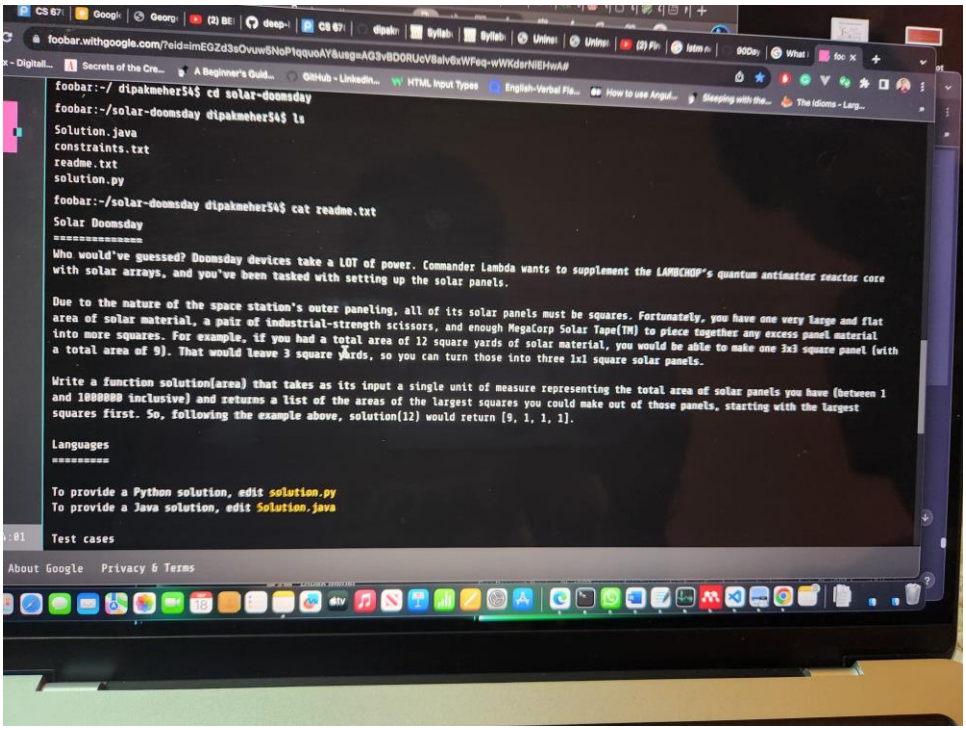
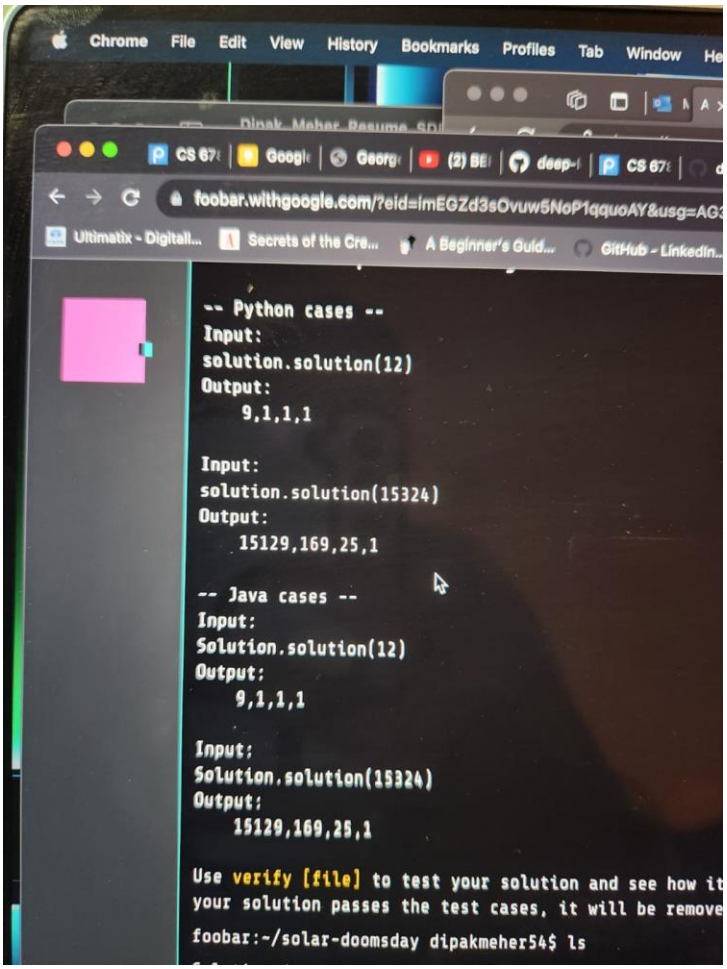
Level 1:

Task 1:

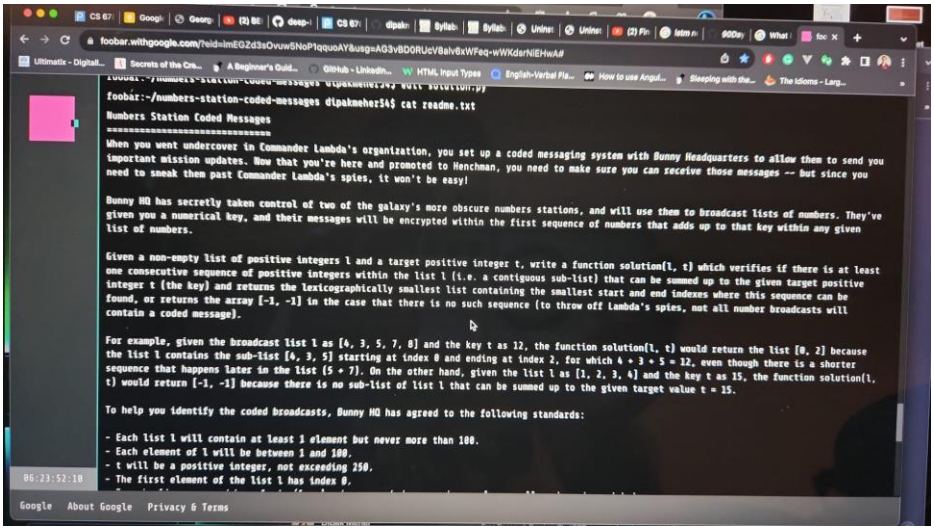


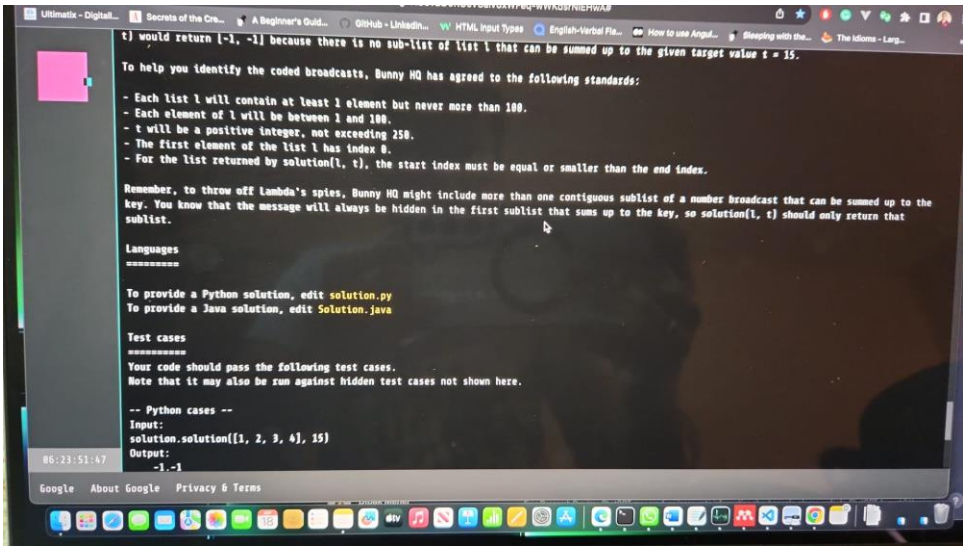


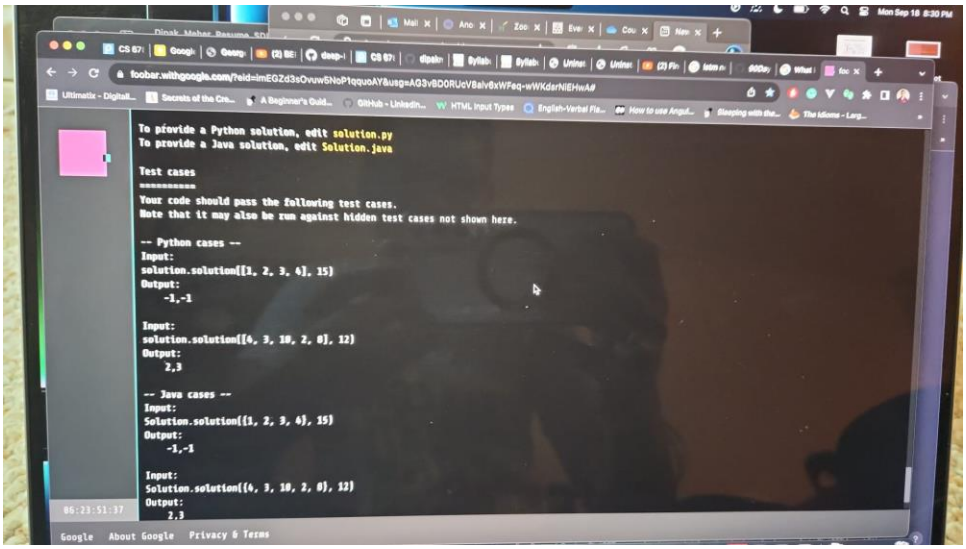

```
def calculate_square_tiles(area):
    square_sizes = []
    while area > 0:
        largest_square = int(area * 0.5) * 2
        square_sizes.append(largest_square)
        area -= largest_square
    return square_sizes


# Example usage:
area = 12
tiles = calculate_square_tiles(area)
print(tiles)  # Output: [9, 1, 1, 1]
```

Level 2:

Task 1:








```
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]
```