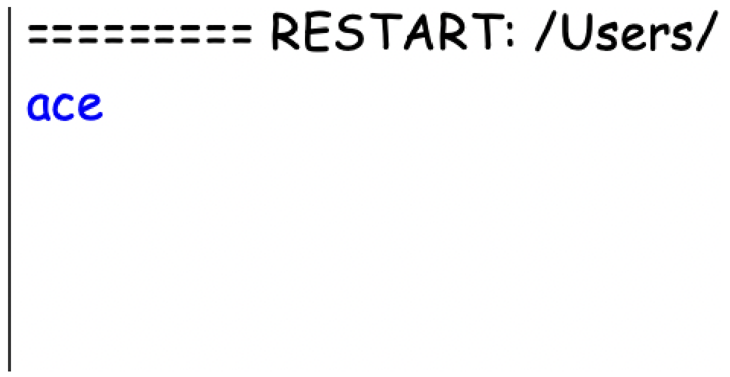
Assignment (19/06/2024)

1 . Odd String Difference

```
CODE : def find_unique_string(words):
    def get_difference_array(word):
        return [ord(word[i + 1]) - ord(word[i]) for i in range(len(word) - 1)]
    diff_arrays = [tuple(get_difference_array(word)) for word in words]
    diff_count = {}
    for diff in diff_arrays:
        if diff in diff_count:
            diff_count[diff] += 1
        else:
            diff_count[diff] = 1
    unique_diff = None
    for diff, count in diff_count.items():
        if count == 1:
            unique_diff = diff
            break
    for i, diff in enumerate(diff_arrays):
        if diff == unique_diff:
            return words[i]
words = ["abc", "bcd", "ace", "xyz"]
unique_string = find_unique_string(words)
print(unique_string)
```

OUTPUT :

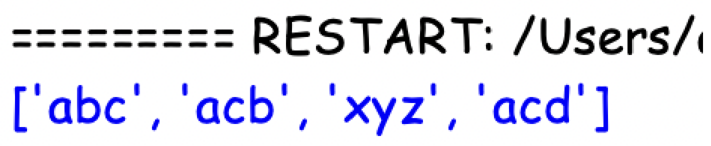


```
========== RESTART: /Users/
ace
```

2 . Words Within Two Edits of Dictionary

```
CODE :
def find_words_with_max_two_edits(queries, dictionary):
    def can_be_transformed_with_two_edits(word1, word2):
              count_diffs = sum(1 for a, b in zip(word1, word2) if a != b)
        return count_diffs <= 2

    result = []
    for query in queries:
        for dict_word in dictionary:
            if can_be_transformed_with_two_edits(query, dict_word):
                result.append(query)
                break
    return result
queries = ["abc", "acb", "xyz", "acd"]
dictionary = ["def", "acc", "abd", "xzz"]
matching_words = find_words_with_max_two_edits(queries, dictionary)
print(matching_words)  # Output should be ["abc", "acb", "acd"]
```

OUTPUT :



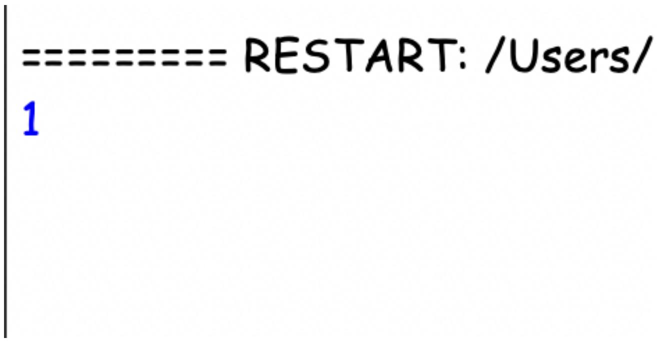
```
========= RESTART: /Users/
['abc', 'acb', 'xyz', 'acd']
```

3 . Destroy Sequential Targets

CODE :

```
def destroy_targets(nums, space):
    from collections import defaultdict
    remainder_groups = defaultdict(list)
    for num in nums:
        remainder = num % space
        remainder_groups[remainder].append(num)
    max_count = 0
    min_seed = float('inf')
    for remainder, group in remainder_groups.items():
        if len(group) > max_count:
            max_count = len(group)
            min_seed = min(group)
        elif len(group) == max_count:
            min_seed = min(min_seed, min(group))

    return min_seed
nums = [3, 7, 8, 1, 1, 5]
space = 3
result = destroy_targets(nums, space)
print(result)
```

OUTPUT :



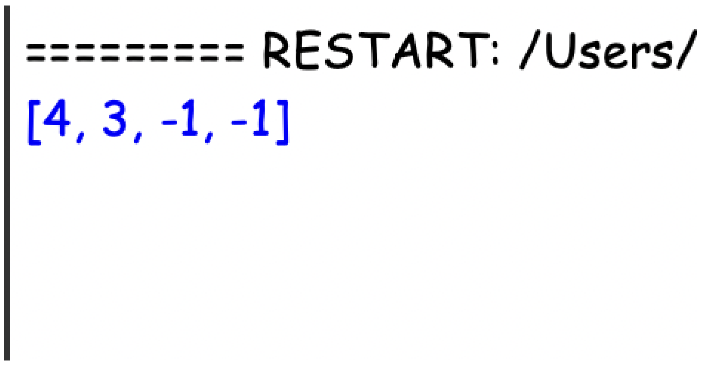
```
========= RESTART: /Users/
1
```

4 . Next Greater Element IV

CODE :

```
def find_second_greater(nums):
    n = len(nums)
    answer = [-1] * n
    stack1 = []  # Stack to keep track of first greater elements
    stack2 = []  # Stack to keep track of second greater elements

    for i in range(n):
            while stack2 and nums[stack2[-1]] < nums[i]:
            index = stack2.pop()
            answer[index] = nums[i]

                temp_stack = []
        while stack1 and nums[stack1[-1]] < nums[i]:
            temp_stack.append(stack1.pop())

        while temp_stack:
            stack2.append(temp_stack.pop())
              stack1.append(i)

    return answer
nums = [1, 2, 4, 3]
print(find_second_greater(nums))  # Output should be [4, 3, -1, -1]
```

OUTPUT :



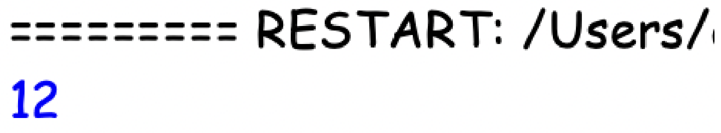
```
========= RESTART: /Users/
[4, 3, -1, -1]
```

5 . Average Value of Even Numbers That Are Divisible by Three

CODE :

```
def average_value_of_even_divisible_by_three(nums):
    filtered_nums = [num for num in nums if num % 2 == 0 and num % 3 == 0]
     if not filtered_nums:
        return 0
       total_sum = sum(filtered_nums)
    count = len(filtered_nums)
       average = total_sum // count  # Integer division to round down
        return average
nums = [1, 2, 3, 4, 6, 12, 18, 21]
print(average_value_of_even_divisible_by_three(nums))  # Output should be 12
```

OUTPUT :



```
========= RESTART: /Users/
12
```

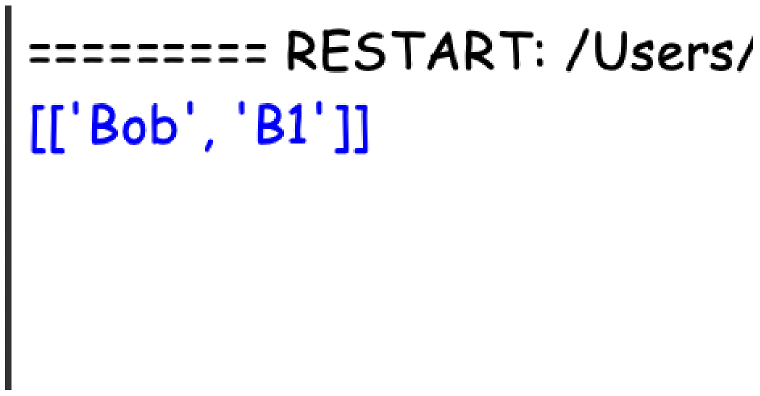
6 . Most Popular Video Creator

CODE :

```
def find_highest_popularity_creators(creators, ids, views):
    from collections import defaultdict
    total_views = defaultdict(int)
    most_viewed_videos = defaultdict(lambda: ("", -1))  # Stores (id, views)
    for creator, video_id, view_count in zip(creators, ids, views):
        total_views[creator] += view_count

        if view_count > most_viewed_videos[creator][1]:
            most_viewed_videos[creator] = (video_id, view_count)
        elif view_count == most_viewed_videos[creator][1]:
            if video_id < most_viewed_videos[creator][0]:
                most_viewed_videos[creator] = (video_id, view_count)
     max_popularity = max(total_views.values())
    result = []
    for creator in total_views:
        if total_views[creator] == max_popularity:
            result.append([creator, most_viewed_videos[creator][0]])

    return result
```

```
# Example usage:
creators = ["Alice", "Bob", "Alice", "Charlie", "Bob"]
ids = ["A1", "B1", "A2", "C1", "B2"]
views = [100, 200, 100, 50, 200]
print(find_highest_popularity_creators(creators, ids, views))
```

OUTPUT :



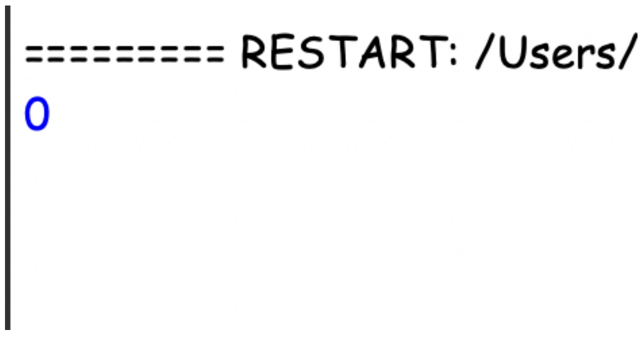
```
========= RESTART: /Users/
[['Bob', 'B1']]
```

7 . Minimum Addition to Make Integer Beautiful

CODE :

```
def sum_of_digits(number):
    return sum(int(digit) for digit in str(number))
def find_min_x(n, target):
    if sum_of_digits(n) <= target:
        return 0
    original_n = n
    increment = 1
    while True:
        if sum_of_digits(n + increment) <= target:
            return increment
        increment += 1
n = 123
target = 6
print(find_min_x(n, target))
```

OUTPUT :



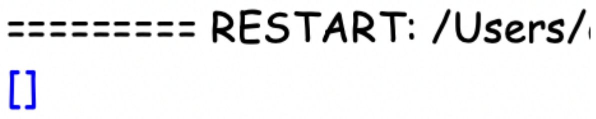
```
========= RESTART: /Users/
0
```

8 . Split Message Based on Limit

CODE :

```
def split_message(message, limit):
    n = len(message)
    max_parts = (n + limit - 1) // limit
    def create_parts(k):
        parts = []
        start = 0
        for i in range(1, k + 1):
            suffix = f"<{i}/{k}>"
            suffix_len = len(suffix)
            part_len = limit - suffix_len
            if start + part_len >= n:
                part = message[start:] + suffix
            else:
                part = message[start:start + part_len] + suffix
            parts.append(part)
            start += part_len
        return parts
    for b in range(1, max_parts + 1):
        suffix_len = len(f"<{b}/{b}>")
        available_len = limit - suffix_len
        if available_len <= 0:
            break
        required_parts = (n + available_len - 1) // available_len
        if required_parts <= b:
            return create_parts(b)

    return []
message = "thisisaverylongmessage"
limit = 10
result = split_message(message, limit)
print(result)
```

OUTPUT :



```
========= RESTART: /Users/
[]
```