WEEK 12:

1.

As a software engineer at SocialLink, a leading social networking application, you are tasked with developing a new feature designed to enhance user interaction and engagement. The company aims to introduce a system where users can form connections based on shared interests and activities. One of the feature's components involves analyzing pairs of users based on the activities they've participated in, specifically looking at the numerical difference in the number of activities each user has participated in.

Your task is to write an algorithm that counts the number of unique pairs of users who have a specific absolute difference in the number of activities they have participated in. This algorithm will serve as the backbone for a larger feature that recommends user connections based on shared participation patterns.

Problem Statement

Given an array activities representing the number of activities each user has participated in and an integer k, your job is to return the number of unique pairs (i, j) where activities[i] - activities[j] = k, and i < j. The absolute difference between the activities should be exactly k.

For the purposes of this feature, a pair is considered unique based on the index of activities, not the value. That is, if there are two users with the same number of activities, they are considered distinct entities.

Input Format

The first line contains an integer, n, the size of the array nums.

The second line contains n space-separated integers, nums[i].

The third line contains an integer, k.

Output Format

Return a single integer representing the number of unique pairs (i, j)

where $|$ nums[i] - nums[j] $|$ = k and i < j.

Constraints:

$1 \leq n \leq 10^5$

$-10^4 \leq nums[i] \leq 10^4$

$0 \leq k \leq 10^4$

For example:

Input	Result
5 1 3 1 5 4 0	1
4 1 2 2 1 1	4

```
def count_pairs_with_difference_k(activities, k):
    count = 0
    n = len(activities)
    for i in range(n):
        for j in range(i + 1, n):
            if abs(activities[i] - activities[j]) == k:
                count += 1
    return count

# Reading input
n = int(input())
activities = list(map(int, input().split()))
k = int(input())

# Calling function and printing the result
print(count_pairs_)
```

OUTPUT:

	Input	Expected	Got	
	4 1 2 3 4 1	3	3	
	5 1 3 1 5 4 0	1	1	
	4 1 2 2 1 1	4	4	

Passed all tests!

Correct

2.

Given an integer `n`, print *`true` if it is a power of four. Otherwise, print `false`*.

An integer `n` is a power of four, if there exists an integer `x` such that `n == 4^x`.

For example:

Input	Result
16	True
5	False

```
def is_power_of_four(n):
    if n <= 0:
        return False
    while n > 1:
        if n % 4 != 0:
            return False
        n //= 4
    return True


# Test the function
n = int(input())
```


```
print(is_power_of_four(n))
```

OUTPUT:

	Input	Expected	Got	
	16	True	True	
	5	False	False	
	1	True	True	
	-1	False	False	

Passed all tests!

Correct

3. Background:

Dr. John Wesley maintains a spreadsheet with student records for academic evaluation. The spreadsheet contains various data fields including student IDs, marks, class names, and student names. The goal is to develop a system that can calculate the average marks of all students listed in the spreadsheet.

Problem Statement:

Create a Python-based solution that can parse input data representing a list of students with their respective marks and other details, and compute the average marks. The input may present these details in any order, so the solution must be adaptable to this variability.

Input Format:

The first line contains an integer N, the total number of students.

The second line lists column names in any order (ID, NAME, MARKS, CLASS).

The next N lines provide student data corresponding to the column headers.

Output Format:

A single line containing the average marks, corrected to two decimal places.

Constraints:

1≤N≤100

Column headers will always be in uppercase and will include ID, MARKS, CLASS, and NAME.

Marks will be non-negative integers.

For example:

Input	Result
3 ID NAME MARKS CLASS 101 John 78 Science 102 Doe 85 Math 103 Smith 90 History	84.33
3 MARKS CLASS NAME ID 78 Science John 101 85 Math Doe 102 90 History Smith 103	84.33

```
def calculate_average_marks(data):
    total_marks = 0
    num_students = 0

    for student in data:
        if 'MARKS' in student:
            total_marks += int(student['MARKS'])
            num_students += 1

    if num_students == 0:
        return 0
```


```
    return total_marks / num_students

# Read input
N = int(input())
columns = input().split()

# Initialize data structure to store student records
students = []

# Read student data
for _ in range(N):
    student_data = input().split()
    student_record = {columns[i]: student_data[i] for i in range(len(columns))}
    students.append(student_record)

# Calculate average marks
average_marks = calculate_average_marks(students)

# Print average marks with two decimal places
print("{:.2f}".format(average_marks))
```

OUTPUT:

	Input	Expected	Got	
	3 ID NAME MARKS CLASS 101 John 78 Science 102 Doe 85 Math 103 Smith 90 History	84.33	84.33	

	Input	Expected	Got	
	3 MARKS CLASS NAME ID 78 Science John 101 85 Math Doe 102 90 History Smith 103	84.33	84.33	

Passed all tests!

Correct

Marks for this submission: 1.00/1.00.

4.

Background:

Raghu owns a shoe shop with a varying inventory of shoe sizes. The shop caters to multiple customers who have specific size requirements and are willing to pay a designated amount for their desired shoe size. Raghu needs an efficient system to manage his inventory and calculate the total revenue generated from sales based on customer demands.

Problem Statement:

Develop a Python program that manages shoe inventory and processes sales transactions to determine the total revenue generated. The program should handle inputs of shoe sizes available in the shop, track the number of each size, and match these with customer purchase requests. Each transaction should only proceed if the desired shoe size is in stock, and the inventory should update accordingly after each sale.

Input Format:

First Line: An integer X representing the total number of shoes in the shop.

Second Line: A space-separated list of integers representing the shoe sizes in the shop.

Third Line: An integer N representing the number of customer requests.

Next N Lines: Each line contains a pair of space-separated values:

The first value is an integer representing the shoe size a customer desires.

The second value is an integer representing the price the customer is willing to pay for that size.

Output Format:

Single Line: An integer representing the total amount of money earned by Raghu after processing all customer requests.

Constraints:

1≤X≤1000 — Raghu's shop can hold between 1 and 1000 shoes.

Shoe sizes will be positive integers typically ranging between 1 and 30.

1≤N≤1000 — There can be up to 1000 customer requests in a single batch.

The price offered by customers will be a positive integer, typically ranging from $5 to $100 per shoe.

For example:

Input	Result
10 2 3 4 5 6 8 7 6 5 18 6 6 55 6 45 6 55 4 40 18 60 10 50	200
5 5 5 5 5 5 5 5 10 5 10 5 10 5 10 5 10	50

Answer:(penalty regime: 0 %)

```
class ShoeInventory:

    def __init__(self, shoe_sizes):

        self.inventory = {size: 0 for size in shoe_sizes}


    def add_shoes(self, size, quantity):

        self.inventory[size] += quantity
```


```
    def sell_shoes(self, size):
        if self.inventory.get(size, 0) > 0:
            self.inventory[size] -= 1
            return True
        return False

class TransactionManager:
    def __init__(self, shoe_inventory):
        self.shoe_inventory = shoe_inventory
        self.total_revenue = 0

    def process_transactions(self, customer_requests):
        for size, price in customer_requests:
            if self.shoe_inventory.sell_shoes(size):
                self.total_revenue += price

# Read input
X = int(input())  # Total number of shoes
shoe_sizes = list(map(int, input().split()))  # Shoe sizes available
N = int(input())  # Number of customer requests

# Initialize shoe inventory
inventory = ShoeInventory(shoe_sizes)

# Populate initial inventory
for size in shoe_sizes:
    inventory.add_shoes(size, X // len(shoe_sizes))
```


```
# Initialize transaction manager
transaction_manager = TransactionManager(inventory)

# Process transactions
for _ in range(N):
    size, price = map(int, input().split())
    transaction_manager.process_transactions([(size, price)])

# Print total revenue
print(transaction_manager.total_revenue)
```

OUTPUT:

Input	Expected	Got
10 2 3 4 5 6 8 7 6 5 18 6 6 55 6 45 6 55 4 40 18 60 10 50	200	200
5 5 5 5 5 5 5 5 10 5 10 5 10 5 10 5 10	50	50
4 4 4 6 6 5 4 25 4 25 6 30 6 55 6 55	135	135

Passed all tests!

Correct

Marks for this submission: 1.00/1.00.

5.

Background:

Rose manages a personal library with a diverse collection of books. To streamline her library management, she needs a program that can categorize books based on their genres, making it easier to find and organize her collection.

Problem Statement:

Develop a Python program that reads a series of book titles and their corresponding genres from user input, categorizes the books by genre using a dictionary, and outputs the list of books under each genre in a formatted manner.

Input Format:

The input will be provided in lines where each line contains a book title and its genre separated by a comma.

Input terminates with a blank line.

Output Format:

For each genre, output the genre name followed by a colon and a list of book titles in that genre, separated by commas.

Constraints:

Book titles and genres are strings.

Book titles can vary in length but will not exceed 100 characters.

Genres will not exceed 50 characters.

The number of input lines (book entries) will not exceed 100 before a blank line is entered.

For example:

Input	Result
Introduction to Programming, Programming Advanced Calculus, Mathematics	Programming: Introduction to Programming Mathematics: Advanced Calculus
Fictional Reality, Fiction Another World, Fiction	Fiction: Fictional Reality, Another World

```
def categorize_books_sorted():
    books = {}

    while True:
        try:
            user_input = input().strip()
        except EOFError:
            break

        if not user_input:
            break

        title, genre = user_input.split(',')
        genre = genre.strip()

        if genre in books:
            books[genre].append(title)
        else:
            books[genre] = [title]

    for genre in sorted(books.keys()):
        print(f"{genre}: {', '.join(books[genre])}")

categorize_books_sorted()
```


OUTPUT:

	Input	Expected	Got	
	Introduction to Programming, Programming Advanced Calculus, Mathematics	Programming: Introduction to Programming Mathematics: Advanced Calculus	Programming: Introduction to Programming Mathematics: Advanced Calculus	
	Fictional Reality, Fiction Another World, Fiction	Fiction: Fictional Reality, Another World	Fiction: Fictional Reality, Another World	

Passed all tests!

Correct