else:

print("The square root of",a,"is {:.2f}".format(a\*\*0.5)) except:

print("Error: could not convert string to float")

# OUTPUT:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Inpu t** | **Expected** | **Got** |  |
|  | 16 | The square root of 16.0 is 4.00 | The square root of 16.0 is 4.00 |  |
|  | 0 | The square root of 0.0 is 0.00 | The square root of 0.0 is 0.00 |  |
|  | -4 | Error: Cannot calculate the square root of a negative number. | Error: Cannot calculate the square root of a negative number. |  |
| Passed all tests! | | | | |
| **Correct** | | | | |

**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 ≤ n ≤ 105

-104 ≤ nums[i] ≤ 104

0 ≤ k ≤ 104

## 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** | **Expecte d** | **Go t** |  |  |
|  | 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 == 4x.

## 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:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Inpu t** | **Expecte d** | **Got** |  |  |
|  | 16 | True | True |  |
|  | 5 | False | Fals e |  |
|  | 1 | True | True |  |
|  | -1 | False | Fals e |  |
| 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   1. John 78 Science 2. Doe 85 Math | 84.33 |

|  |  |
| --- | --- |
| **Input** | **Result** |
| 103 Smith 90 History |  |
| 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** | **Expecte d** | **Got** |  |  |
|  | 3  ID NAME MARKS CLASS   1. John 78 Science 2. Doe 85 Math 3. Smith 90 History | 84.33 | 84.3  3 |  |
|  | 3  MARKS CLASS NAME ID  78 Science John 101  85 Math Doe 102  90 History Smith 103 | 84.33 | 84.3  3 |  |
| 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 | 200 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | | | | **Result** |
| 6 55 | | | |  |
| 6 45 | | | |
| 6 55 | | | |
| 4 40 | | | |
| 18 60 | | | |
| 10 50 | | | |
| 5 |  |  |  | 50 |
| 5 | 5 5 | 5 | 5 |  |
| 5 |  |  |  |  |
| 5 | 10 |  |  |  |
| 5 | 10 |  |  |  |
| 5 | 10 |  |  |  |
| 5 | 10 |  |  |  |
| 5 | 10 |  |  |  |

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** | **Expecte d** | **Go t** |  |  |
|  | 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** | | | | |