ANNIVERSARY

BCN ▪ LEO ▪ MAD ▪ VAL

Valencia 2025

Problems
and solutions

#	Problem	Points
1	Moviegoers	1
2	Fizz Buzz	3
3	Robocup Ball Position	3
4	DAC Conversion	4
5	Recover Information	4
6	Climbing And Descending	4
7	The Easter Bunny Problem	4
8	Happy Pi Day	4
9	La Barbacoa	4
10	Compound Interest Calculator	5
11	Ancient Cryptex	5
12	Sophie Germain	6
13	New Printer	6
14	The Sudoku Redemption Challenge	7
15	Adler 32	7
16	TriPizza	7
17	The riddle of the Golden Numbers	7
18	Single Neuron	8
19	Vigenere Cipher Encoder	8
20	A Polynomial Spell	10
21	Viking ascent	10
22	Undo Redo Actions	12
23	Gladiators Clash	12
24	EHB League	13
25	Ancient time Traveller	13
26	The automatic coach	13
27	Lagrange's Four Squares	13
28	Ready Player One	17
29	Minesweeper	19
30	Time Corridors	30

1 Moviegoers

1 points

Introduction

Stanley and Alfred are planning to attend a screening of a classic movie. They have already purchased their tickets online and do not intend to stop in the lobby for snacks and drinks. Consequently, they will be able to proceed directly to the screening room. Stanley is currently uncertain about the appropriate time to depart. He is at his residence now and must first pick up Alfred. It will take him several minutes to drive to Alfred's home, and then additional minutes to travel from Alfred's home to the cinema. If the film is scheduled to begin at a specific minute of the day, at what minute should Stanley commence his journey to ensure they do not miss the beginning of the movie?

Input

The input is composed of three lines. The first line contains a single integer indicating the duration in minutes to reach Alfred's residence. The second line includes a single integer denoting the travel time in minutes from Alfred's home to the cinema. The third line presents a single integer specifying the minute of the day when the movie is set to begin.

Output

The output is a single integer representing the latest minute of the day that Stanley can begin driving to ensure they arrive on time.

Example

Input

```
10
4
1335
```

Output

```
1321
```

Python

```python
timeToAlfred = int(input())
timeToCinema = int(input())
timeMovieStarts = int(input())
# Calculate the time Alfred needs to leave to get to the cinema on time
print(timeMovieStarts - (timeToAlfred + timeToCinema))
```

2 FizzBuzz

3 points

Introduction

You are playing FizzBuzz with your friend, but he claims that you missed saying the correct word several times. To prove that you were correct, you will implement a function that counts from *m* to *n* playing the game with the following conditions:

- For numbers that are multiples of 3, you will print Fizz instead of the number.
- For numbers that are multiples of 5, you will print Buzz instead of the number.
- For numbers that are multiples of 3 and 5, you will print FizzBuzz.

Input

The first input will be a line with m, the number you must count from.
The second input will be a line with n, the number you must count to.

Output

The output will be the list of numbers with Fizz, Buzz or FizzBuzz in the right place.

Example 1

Input

```
1
5
```

Output

```
1
2
Fizz
4
Buzz
```

Example 2

Input

```
7
15
```

Output

```
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
```

Python

```python
# Read input image
m = int(input())
n = int(input())

for i in range(m,n+1):
    if(i%3==0 and i%5==0):
        print("FizzBuzz")
    elif(i%3==0):
        print("Fizz")
    elif(i%5==0):
        print("Buzz")
    else:
        print(i)
```

3 RoboCup Ball Position

3 points

Introduction

This is a true story. All characters depicted in this story have been modified to protect their identity.

In 2011, while participating in the RoboCup soccer competition (an autonomous robot competition to play soccer), we developed stereo cameras (i.e., two cameras) mounted on the ceiling that could track the ball and provide the 3D coordinates of the ball (X, Y, Z). There was just one problem: the position was given using one of the cameras as a reference, but we needed the center of the field as the reference.

Therefore, we took some measurements:

- When the ball was at the position (0, 0, 0) of the field, the stereo cameras system marked (-0.378094, -0.044534, 3.54707).
- When the ball was at the position (0, 0, 1) of the field, the stereo cameras system marked (-0.460408, -0.74951, 2.91714).
- When the ball was at the position (-1, 0, 0) of the field, the stereo cameras system marked (0.427701, -0.555208, 3.98178).
- When the ball was at the position (0, 1, 0) of the field, the stereo cameras system marked (0.288779, 0.451449, 2.93639).
- When the ball was at the position (1, 1, 0) of the field, the stereo cameras system marked (-0.517016, 0.962123, 2.50168).
- When the ball was at the position (1, 0, 0) of the field, the stereo cameras system marked (-1.183889, 0.46614, 3.11236).
- When the ball was at the position (-1, -1, 0) of the field, the stereo cameras system marked (0.77, -0.208, 1.8878).
- When the ball was at the position (1, -1, 1) of the field, the stereo cameras system marked (1.111, 0.872364, -1.6786).

These values are the real measured values, so do not expect the results to be nicely rounded integers.

You are asked to write a program that, given a point P (X, Y, Z) measured from the camera, returns the position of the ball on the field.

Input

Your program will receive three comma-separated integers ranging from -1 to 1 and will return the corresponding 3D stereo camera values with six decimal places.

Output

These integer numbers may or may not correspond to specific field positions. If they do correspond (for instance, 0,0,1), the 3D stereo camera values will be displayed. Note that all stereo camera values will be shown with six decimal places. If the input values do not correspond to any 3D camera values, the text "INVALID POSITION" will be displayed.

Example 1

Input

```
0,0,1
```

Output

```
-0.460408,-0.749510,2.917140
```

Example 2

Input

```
0,1,0
```

Output

```
0.288779,0.451449,2.936390
```

Example 3

Input

```
-1,0,1
```

Output

```
INVALID POSITION
```

Python

```
# Mapping of field positions to stereo camera coordinates
field_to_camera = {
    (0, 0, 0): (-0.378094, -0.044534, 3.547070),
    (0, 0, 1): (-0.460408, -0.749510, 2.917140),
    (-1, 0, 0): (0.427701, -0.555208, 3.981780),
    (0, 1, 0): (0.288779, 0.451449, 2.936390),
    (1, 1, 0): (-0.517016, 0.962123, 2.501680),
    (1, 0, 0): (-1.183889, 0.466140, 3.112360),
    (-1, -1, 0): (0.770000, -0.208000, 1.887800),
    (1, -1, 1): (1.111000, 0.872364, -1.678600),
}

def get_camera_coordinates(x, y, z):
    # Check if the given position exists in the dictionary
    if (x, y, z) in field_to_camera:
        # Return the corresponding stereo camera coordinates, formatted to 6
decimal places
        camera_coords = field_to_camera[(x, y, z)]
        return
f"{camera_coords[0]:.6f},{camera_coords[1]:.6f},{camera_coords[2]:.6f}"
    else:
        # If the position is not found, return "INVALID POSITION"
        return "INVALID POSITION"

# Read the input from the user
x, y, z = map(int, input("").split(','))

# Get the camera coordinates or invalid message
result = get_camera_coordinates(x, y, z)

# Print the result
print(result)
```

4 DAC Conversion

4 points

Introduction

In electronics, a digital-to-analog converter (DAC) is a device that transforms a binary signal into an analog output. For example, an 8-bit DAC can represent up to 256 different values, with each step differing by 1/256 of the full-scale value, which defines the system's resolution.

Let's create a program that converts a decimal representation of a signal into the corresponding analog voltage level produced by a DAC. Given a value range of 0-1023 and a reference range of 0-5.00 volts, the value and reference are directly proportional.

Input

The input is a positive integer with 10-bit resolution.

Output

The output is the reference voltage rounded to two decimal places.

Example 1

Input

```
0
```

Output

```
0.00 V
```

Example 2

Input

```
1023
```

Output

```
5.00 V
```

Example 3

Input

```
356
```

Output

```
1.74 V
```

Example 4

Input

```
2
```

Output

```
0.01 V
```

Python

```python
def dac_output(decimal_value):
    # Calculate the analog voltage
    analog_voltage = (decimal_value / 1023) * 5.00
    # Round the result to two decimal places
    return round(analog_voltage, 2)

# Example usage
digitalInput = int(input())
print(f"{dac_output(digitalInput):.2f} V")
```

5 Recover Information

4 points

Introduction

You are working on data recovery from a RAID system that has experienced corruption on one of its hard drives. In this system, redundant information is stored on an additional disk to ensure data integrity. Your task is to restore the information from the damaged hard drive using the redundancy disk.

The first line of input represents the contents of the functioning RAID hard drive (data disk), while the second line represents the contents of the redundancy disk (parity disk). Your goal is to combine the data from these two disks using a NAND operation. The logic of the NAND operation is defined as follows:

```
0 NAND 0 = 1
0 NAND 1 = 1
1 NAND 0 = 1
1 NAND 1 = 0
```

Can you write a program that combines the data from the data and parity disks according to this NAND operation?

Input

You will be provided with two lines of input, each representing the contents of one of the disks in binary format. Both lines will have the same length.

Output

Output a single line that represents the result of applying the NAND operation between the two input lines.

Example

Input

```
11001
10110
```

Output

```
01111
```

Python

```python
disk1 = input()
disk2 = input()
result = ""
for i in range(0,len(disk1)):
    if disk1[i] == '1' and disk2[i] == '1':
        result += '0'
    else:
        result += '1'
print(result)
```

6 Climbing And Descending

4 points

Introduction

Imagine you are on a hiking adventure, climbing a mountain. Your journey starts at the base, and you steadily ascend, step by step, until you reach the peak. At the peak, you take a moment to enjoy the breathtaking view. Afterward, you begin your descent, carefully making your way down the other side of the mountain until you reach the base again.

In this problem, a sequence of numbers represents your journey. The numbers should first increase, symbolizing your climb up the mountain, and then decrease, symbolizing your descent. Your task is to determine if the sequence of numbers follows this pattern of climbing to a peak and then descending, just like a perfect mountain hike. If the sequence does, it means you successfully climbed and descended the mountain. If not, it means your journey did not follow the expected path.

Input

The input consists of:

- The first line contains an integer N, the number of elements that composes a journey
- The next N lines each contain a positive integer, representing the elements of the sequence.

Output

"True" if the sequence first strictly increases and then strictly decreases. Otherwise, print "False".

Example 1

Input

```
3
4
5
6
```

Output

```
False
```

Example 2

Input

```
6
1
3
4
12
10
9
```

Output

```
True
```

Example 3

Input

```
5
1
3
2
1
5
```

Output

```
False
```

Example 4

Input

```
5
4
3
2
1
2
```

Output

```
False
```

Python

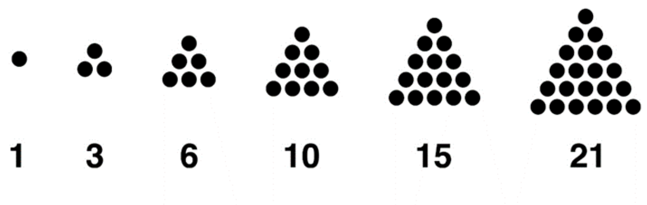
```python
# Check if a sequence of positive integers is first increasing, arrives to a
peak, and then decreases.
def checkSequence(sequence):
    increasing = True
    for i in range(1, len(sequence)):
        if increasing:
            # Beware of the case the sequence starts decreasing
            if sequence[i] < sequence[i-1] and i == 1:
                return False
            if sequence[i] < sequence[i-1]:
                increasing = False
        if not increasing:
            if sequence[i] > sequence[i-1]:
                return False
    return not increasing

# Read input
numElements = int(input())
elements = []
for _ in range(numElements):
    elements.append(int(input()))

# Check if the sequence is climbing and descending
if checkSequence(elements):
    print("True")
else:
    print("False")
```

7 The Easter Bunny Problem

4 points

Introduction

This year, the Easter Bunny visited the Egyptian pyramids and was fascinated. To pay homage to them, he wants to hide the eggs in a perfectly triangular shape. This is the idea he has in mind:



With 1 egg, you CAN form a perfectly triangular structure, and with 3, and with 6, ... But it would not work with 2 eggs, nor with 4, nor with 5, ... Can you help him determine if, with a number N of eggs, he can build what he wants?

Input

The input consists of two parts:

- The first part is a line with a single integer T.
- The second part contains T integers, each representing the number N of eggs to be hidden.

Output

Following the same order of input, output T lines with "YES" if the Nth number in the list serves the purpose or "NO" otherwise.

Example

Input

```
3
6
10
7
```

Output

```
YES
YES
NO
```

Python

```python
numLines = int(input())
for _ in range(numLines):
    S = int(input())
    N = (-1 + (1+8*S)**(1/2))/2
    if N == int(N) and N>0:
        print("YES")
    else:
        print("NO")
```

8 Happy Pi Day

4 points

Introduction

In a few days it will be March 14th, such date is the annual celebration of the mathematical constant Pi (π) because it is the 3rd month followed by 14 which are the first three digits of this number. As you may know Pi is the ratio of the circle's circumference to its diameter and it is an infinite decimal number. In 2021, Pi calculation reached up to 100 trillion decimal digits. But let's not go that far and consider only the first hundred decimal digits:

3.1415926535897932384626433832795028841971693993751058209749445923078164062862089986280348253421170679

We want to explore Pi and look for specific numeric sequences inside the decimal digits. Given a number sequence of up to 10 digits, can you write a program that will search for this sequence in the first 100 decimal digits of Pi? Your program should output the position of the first occurrence of the number sequence in Pi, or -1 if the sequence is not found.

Input

The input is an integer up to 10 digits long.

Output

The output is an integer reporting the position of the first ocurrence starting to count from the decimal point or -1 if the sequence is not found.

Example 1

Input

```
14159
```

Output

```
1
```

Example 2

Input

```
65
```

Output

```
7
```

Example 3

Input

```
1416
```

Output

```
-1
```

Python

```python
piDecimals =
".14159265358979323846264338327950288419716939937510582097494459230781640628620 8
9986280348253421170679"

pattern = input()

#With the find method we can get the position where the pattern is found. If
not, a -1 is returned
position = piDecimals.find(pattern)

print(str(position))
```

9 La Barbacoa

4 points

Introduction

As every summer, your friend Georgie has fun cooking meat and vegetables on his barbecue grill. To do so, there are two kinds of skewers (thin metal sticks of variable length) that hold pieces of food together while they are being cooked. A vegetarian skewer is a skewer that has only vegetables (represented by 'o'). A non-vegetarian skewer is a skewer with at least one piece of meat (represented by 'x').

For example, the grill below has 4 non-vegetarian skewers and 1 vegetarian skewer (the one in the middle):

```
--xo--x--ox--
--xx--x--xx--
--oo--o--oo--          <– vegetarian skewer
--xx--x--ox--
--xx--x--ox--
```

Given a barbecue grill, can you write a program that prints the number of vegetarian skewers and the number of non-vegetarian skewers?

Input

The input consists of variable number of lines, one per skewer. The input ends with the character '#'.

Output

The output is a pair of integers: the first is the number of vegetarian skewers, and the second is the number of non-vegetarian skewers.

Example 1

Input

```
--xo--x--ox--
--xx--x--xx--
--oo--o--oo--
--xx--x--ox--
--xx--x--ox--
#
```

Output

```
1 4
```

Example 2

Input

```
--o-ooo--
--xx--x--xx--
--oo--oo--
--xx--x--ox--
#
```

Output

```
2 2
```

Python

```python
# Read skewer from input
skewer = input()

# Initialize counters
vegetables = 0
nonVegetables = 0

while skewer != "#":
    if skewer.find("x") != -1:
        nonVegetables += 1
    else:
        vegetables += 1
    skewer = input()

print(vegetables, nonVegetables)
```

10 Compound interest calculator

5 points

Introduction

You have 1000 € in your bank account. The bank uses your money to lend it to other people and make a profit. In return, the bank will increase the money in your account by a percentage each year. This percentage is called an interest rate. The higher it is the more money you will have at the end of the year.

You want to find out how much money you will have in the future by comparing the interest rate your bank offers versus other banks. You might end much richer if you move your money to a bank with higher interest rates, especially in the long term.

To do so, you will use the following compound interest formula to calculate the final money (M_f) considering your initial money (M_i), the interest rate (r) and the number of years you want to keep the money in the bank (n).

$$\boldsymbol{M_f = M_i \times (1 + r)^n}$$

For example, to calculate the final money after 15 years in a bank that gives you a 5% interest rate you would have to do the following calculation:

$$\boldsymbol{M_f = 1000 \times (1 + 0.05)^{15}}$$

Input

The input consists of two lines:

- The first line will be the interest rate (r). It will be a positive integer number greater than zero.
- The second line will be the number of years the money will be in this bank (n). It will be a positive integer number greater than zero.

Output

The output should be the final money you will have in your account (M_f), printed out with 2 decimals rounded to the nearest hundredth.

Example

Input	Output
5 15	2078.93

Python

```python
def calculate_compound_interest(initial_money, interest_rate, years):
    # Convert interest rate to decimal form
    rate = interest_rate / 100.0
    # Calculate final money using the compound interest formula
    final_money = initial_money * (1 + rate) ** years
    return final_money

# Fixed initial money
initial_money = 1000.0

# Read inputs
interest_rate = int(input())
years = int(input())

# Calculate final money
final_money = calculate_compound_interest(initial_money, interest_rate, years)

# Output the result with 2 decimal places
print(f"{final_money:.2f}")
```

C++

```cpp
#include <iostream>
#include <cmath>
#include <iomanip>

int main() {
    const double initialMoney = 1000.0; // Fixed initial money
    int interestRate; // Input interest rate as an integer percentage
    int years; // Input number of years
    std::cin >> interestRate;
    std::cin >> years;
    double rate = interestRate / 100.0;
    double finalMoney = initialMoney * std::pow(1 + rate, years);

    // Output the result with 2 decimal places
    std::cout << std::fixed << std::setprecision(2) << finalMoney << std::endl;

    return 0;
}
```

11 Ancient Cryptex

5 points

Introduction

In the distant future, a new type of human has come into existence on planet Earth: the so-called "Homo psychic." You, a renowned Homo psychic, have been tasked with investigating an artifact from an ancient civilization.

As you wire your psyche around the intricate patterns on the artifact, you discover that to unlock its secrets, you must break the sigil and passcode. Any mistake in the spell will render the artifact unusable.

For this task, you have learned that any phrase will suffice. However, the input string must be at least 16 characters long. If the string is shorter than 16 characters, the remaining spaces must be filled with zeroes. Spaces between words should also be replaced with zeroes. Any characters beyond 16 will be discarded.

Input

The input consists of a string.

Output

The output should be a 4x4 matrix containing the requested string, filling blank spaces with '0'.

Example 1

Input

```
The secret within
```

Output

```
T h e 0
s e c r
e t 0 w
i t h i
```

Example 2

Input

```
Hello world
```

Output

```
H e l l
o 0 w o
r l d 0
0 0 0 0
```

Python

```python
def main():
    # Leer la frase de entrada
    input_text = input("")

    # Asegurarse de que no tenga más de 16 caracteres
    if len(input_text) > 16:
        input_text = input_text[:16]

    # Si tiene menos de 16 caracteres, rellenamos con '0'
    input_text = input_text.ljust(16, '0')

    # Crear la matriz 4x4
    matriz = [['' for _ in range(4)] for _ in range(4)]

    # Llenar la matriz con los caracteres de la frase
    index = 0
    for i in range(4):
        for j in range(4):
            # Reemplazar los espacios con '0'
            if input_text[index] == ' ':
                matriz[i][j] = '0'
            else:
                matriz[i][j] = input_text[index]
            index += 1

    # Mostrar la matriz
    for i in range(4):
        print(" ".join(matriz[i]) + " ")

# Ejecutar el programa
if __name__ == "__main__":
    main()
```

12 Sophie Germain

6 points

Introduction

Sophie Germain was a pioneering French mathematician known for her contributions to number theory, including her work on prime numbers, particularly those now termed Sophie Germain prime pairs. She overcame social and gender barriers in her pursuit of mathematical knowledge, leaving a lasting impact on the field.

A Sophie Germain prime pair is a pair of prime numbers (p, q) such that:

- p is a prime number and
- $2 \cdot p + 1 = q$ is also a prime number.

For example, (2, 5) is a Sophie Germain prime pair because 2 is prime and 2·2 + 1 = 5 is also prime.

Write a program to find out whether there is a Sophie Germain prime pair for a given positive integer.

Input

The input is composed by a p positive integer.

Output

The output will return q if p and q form a Sophie Germain prime pair. Otherwise, the output will be: p is not a Sophie Germain prime.

Example 1

Input

```
2
```

Output

```
5
```

Example 2

Input

```
7
```

Output

```
7 is not a Sophie Germain prime
```

Python

```python
import math

def isPrime(n):
  for i in range(2,int(math.sqrt(n))+1):
    if (n%i) == 0:
      return False
  return True

# Main program

# Read number p from standard input
p = int(input())

# Calculate number q
q = 2 * p + 1

# Check whether p and q form a Sophie Germain primes pair
if isPrime(p) and isPrime (q):
  print(str(q))
else:
  print(str(p) + " is not a Sophie Germain prime")
```

13 New Printer

6 points

Introduction

We are designing a new printer in HP and need your help to print images. Our new printer will use three inks of colors RGB, meaning Red, Green and Blue.

The printer can print the following colors:

- Red (R)
- Green (G)
- Blue (B)
- Yellow (Y) = Red (R) + Green (G)
- Purple (P) = Blue (B) + Red (R)
- Turquoise (T) = Green (G) + Blue (B)
- White (W) The printer will not print anything for white, as we assume the paper is white.

Given an image represented by a line of 10 characters indicating the colors, such as

```
RWBYWWWPTW
```

we need to determine the passes the printer will need to make to print the image. For composed colors like Yellow, Purple, and Turquoise, the printer will need to make two passes.

When the printer is not printing in a position (because the color is white or not needed in that pass), we will represent it with an underscore _.

For the input `RWBYWWWPTW`, the output should be:

```
R_BR___BG_

___G___RB_
```

Input

The input will be a single line with the colors to print.

Output

The output will be the number of passes needed and the sequences for each pass.

Example 1

Input

```
WWRGBBGRWW
```

Output

```
__RGBBGR__
```

Example 2

Input

```
WYRGBPTYYW
```

Output

```
_RRGBBGRR_
_G___RBGG_
```

Python

```python
def print_image(image):
    # Initialize two passes with underscores
    pass1 = ['_'] * len(image)
    pass2 = ['_'] * len(image)
    for i, color in enumerate(image):
        if color == 'R':
            pass1[i] = 'R'
        elif color == 'G':
            pass1[i] = 'G'
        elif color == 'B':
            pass1[i] = 'B'
        elif color == 'Y':  # Yellow = Red + Green
            pass1[i] = 'R'
            pass2[i] = 'G'
        elif color == 'P':  # Purple = Blue + Red
            pass1[i] = 'B'
            pass2[i] = 'R'
        elif color == 'T':  # Turquoise = Green + Blue
            pass1[i] = 'G'
            pass2[i] = 'B'
        elif color == 'W':  # White, do nothing
            pass
    pass1_str = ''.join(pass1)
    pass2_str = ''.join(pass2)
    return pass1_str, pass2_str
image = input()
pass1, pass2 = print_image(image)
print(pass1)
if pass2 != "__________" :
    print(pass2)
```

14 The Sudoku Redemption Challenge

7 points

Introduction

In a dystopian future where justice is swift and unforgiving, society has turned to an unconventional means of rehabilitation. Convicted programmers are no longer confined to dark, cold prison cells. Instead, they are sent to the Algorithm Redemption Facility, where their skills are put to the ultimate test: solving puzzles that even the most advanced machines fail to crack.

You are one of these programmer convicts, branded as a "red code rogue" for your past transgressions. Your only hope for freedom lies in your ability to create an efficient algorithm capable of solving a seemingly simple, yet devilishly constrained problem: a 3x3 Sudoku puzzle.

But there's a twist.

Each number from 1 to 9 must appear exactly once in the entire grid. The puzzle is partially filled, and the remaining cells must be completed without violating the rules of Sudoku. You are given a specific starting grid, and your solution must respect this initial configuration. The Redemption Board watches your every keystroke, ready to grant your freedom if you succeed—or condemn you to a lifetime of computational servitude if you fail. This is your chance to prove your genius and earn back your place in society.

The challenge is clear: write an algorithm that solves the given Sudoku puzzle while ensuring no number repeats in any row, column, or across the entire grid. Time is limited, and the pressure is immense. Your fellow developers are counting on you to lead the charge.

Can you rise above your past mistakes, code your way to freedom, and become the hero of your team? The clock is ticking...

Note: As there are many different solutions. The good one will have the unknown numbers the lowest to the left and to the top.

Input

The input consists of 9 characters, that could be numbers between 1 and 9 or 'x', separated by white spaces.

Output

The output should be the sudoku solved, as there are many different solutions. To good one will have the unknown numbers the lowest to the left and to the top.

Example 1

Input

```
9 8 7
x 3 x
x x 1
```

Output

```
9 8 7
2 3 4
5 6 1
```

Example 2

Input

```
x 8 x
x x x
x x x
```

Output

```
1 8 2
3 4 5
6 7 9
```

Python

```python
def leer_matriz():
    matriz = []

    for _ in range(3):
        # Lee una línea, separa los elementos por espacio y los agrega como una
lista de elementos
        linea = input().split()
        matriz.append(linea)

    return matriz

def obtener_numeros_faltantes(matriz):
    # Obtiene los números ya usados en la matriz
    usados = set()
    for fila in matriz:
        for elem in fila:
            if elem != 'x':
                usados.add(int(elem))

    # Los números que faltan (del 1 al 9), excluyendo los que ya están en la
matriz
```

```python
    faltantes = [i for i in range(1, 10) if i not in usados]

    # Ordenamos los números faltantes de menor a mayor
    return sorted(faltantes)

def sustituir_x(matriz, numeros_faltantes):
    # Reemplazamos las 'x' por los números faltantes
    idx = 0
    for i in range(3):
        for j in range(3):
            if matriz[i][j] == 'x':
                matriz[i][j] = str(numeros_faltantes[idx])
                idx += 1

def imprimir_matriz(matriz):
    for fila in matriz:
        print(" ".join(fila))

# Leer las tres líneas de entrada
matriz = leer_matriz()

# Obtener los números que faltan (del 1 al 9) y ordenarlos
numeros_faltantes = obtener_numeros_faltantes(matriz)

# Sustituir las 'x' por los números faltantes
sustituir_x(matriz, numeros_faltantes)

# Imprimir la matriz final
imprimir_matriz(matriz)
```

15 Adler-32

7 points

Introduction

On the planet Zorvak, an advanced civilization of beings known as the Zorvaks have long been famous for their expertise in mining valuable resources deep within the planet. For years, they have been collecting minerals and developing technologies to improve their quality of life. However, they now face a new challenge: data integrity.

The Zorvaks rely on an interplanetary network to store and transmit their mining and resource data. This information is critical to their mining operations, but due to the vastness of their network and cosmic interference from nearby galaxies, some of the data may be altered or corrupted during transmission. To ensure that the data they receive is correct and unmodified, the Zorvaks need to implement a reliable data integrity verification mechanism.

As a programmer specialized in the interplanetary communications network, you have been tasked with helping the Zorvaks develop an algorithm that will ensure their data is complete and unaltered. To achieve this, the algorithm must use a hash type capable of representing a sequence of data in a compact and efficient way.

The Adler-32 algorithm is ideal for this purpose due to its efficiency and simplicity. This algorithm generates a unique signature for a set of data, allowing the Zorvaks to verify whether the information has been altered or corrupted during transmission.

Your mission is to write a program that implements the Adler-32 algorithm. The program will take a text string (representing the data) as input and return the checksum (unique signature) generated by the Adler-32 algorithm.

The Adler-32 algorithm works as follows:

The text must be processed in such a way that two different sums are obtained simultaneously: S1 and S2. To do so the ASCII values of the characters of the text are taken and added cumulatively, in the following way:

$$S1 = 1 + D1 + D2 + \ldots + Dn \pmod{65521}$$

$$S2 = 0 + (1 + D1) + (1 + D1 + D2) + \ldots + (1 + D1 + D2 + \ldots + Dn) \pmod{65521} = n \times D1 + (n-1) \times D2 + (n-2) \times D3 + \ldots + Dn + n \pmod{65521}$$

Both S1 and S2 are then reduced using the modulus 65521 to avoid data overflow. Finally, the value of Adler-32 is calculated by combining S2 with S1, and this value is returned as the checksum:

$$\text{Checksum} = (S2 \times 65536) + S1$$

Let's see an example: The Adler-32 sum of the ASCII string "Codewars" would be calculated as follows:

Character	ASCII code	S1	S2
C	67	1 + 67 = 68	0 + 68 = 68
o	111	68 + 111 = 179	68 + 179 = 247
d	100	179 + 100 = 279	247 + 279 = 526
e	101	279 + 101 = 380	526 + 380 = 906
w	119	380 + 119 = 499	906 + 499 = 1405
a	97	499 + 97 = 596	1405 + 596 = 2001
r	114	596 + 114 = 710	2001 + 710 = 2711
s	115	710 + 115 = 825	2711 + 825 = 3536

```
S1 =  825 mod 65521 = 825

S2 = 3536 mod 65521 = 3536

Output = (3536 * 65536) + 825 = 231735296 + 825 = 231736121
```

Input

The input consists of a text representing the data that the Zorvaks wish to verify.

Output

The output should be an integer numeric value representing the integrity signature of the text, according to the steps described.
Note: The algorithm must return unsigned integer positive values.

Example 1

Input

```
Wikipedia
```

Output

```
300286872
```

Example 2

Input

```
Codewars
```

Output

```
231736121
```

Example 3

Input

```
To infinity and beyond!
```

Output

```
1705445475
```

Python

```python
def adler32(text):
    # Constants for Adler-32
    MOD_ADLER = 65521
    s1 = 1
    s2 = 0
    for char in text:
        # Update s1 and s2 according to the Adler-32 formula
        s1 = (s1 + ord(char)) % MOD_ADLER
        s2 = (s2 + s1) % MOD_ADLER
#         print (char, "--> ", ord(char), " --> ",s1, " --> ",s2)
    checksum = (s2 << 16) | s1
    return checksum
text = input().strip()

# Calculate the Adler-32 checksum for the given text
checksum = adler32(text)

# Output the checksum
print(checksum)
```

16 TriPizza

7 points

Introduction

Have you ever heard of "Triggy's Pizzeria"? Triggy, the owner, is known for creating the best TriPizza (triangular-shaped pizzas). Programmers loved his pizzas so much that they wanted to calculate the area of these peculiar pizzas. However rulers are forbidden in the pizzeria, so there is no way to find out the lengths of the sides of the pizzas. Fortunately your mathematician friend brought a centimeter coordinate plane printed. After placing the pizza over the coordinate plane it was possible to find the X, Y points corresponding to the vertices of the triangle.

To help finding out the area you can count on Hero of Alexandria. He was a Greek mathematician that derived the formula for the calculation of the area of a triangle using the length of all three sides:

$$\boldsymbol{Area} = \sqrt{\boldsymbol{s}(\boldsymbol{s}-\boldsymbol{a})(\boldsymbol{s}-\boldsymbol{b})(\boldsymbol{s}-\boldsymbol{c})}$$

where s is the semi-perimeter of the triangle and a, b and c are the lengths of its sides.

Can you code a program to calculate the area of a TriPizza?

Input

The coordinates will be provided in three lines, where each line has a pair of integer values representing coordinates (x, y) that match a triangle vertex.

Output

The output is the area of the TriPizza in squared centimeters with one single decimal.

Example

Input

```
-4 0
2 0
0 4
```

Output

```
12.0
```

Python

```python
import math

# Read the three points
x1, y1 = input().split()
x2, y2 = input().split()
x3, y3 = input().split()

# Convert string to float
x1 = float(x1)
x2 = float(x2)
x3 = float(x3)
y1 = float(y1)
y2 = float(y2)
y3 = float(y3)

# Find out the triangle segments
a = math.sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2))
b = math.sqrt(pow(x3 - x2, 2) + pow(y3 - y2, 2))
c = math.sqrt(pow(x3 - x1, 2) + pow(y3 - y1, 2))

# Calculate the semiperimeter
s = (a + b + c) / 2

# Apply the Heron's formula to find out the area
area = math.sqrt(s*(s-a)*(s-b)*(s-c))

# Print out just one decimal
print(f"{area:.1f}")
```

17 The riddle of the Golden Numbers

7 points

Introduction

In the distant kingdom of Numeria, wise mathematicians believed that prime numbers were fragments of an ancient language connecting mortals to the secrets of the universe. These numbers, known as the Golden Numbers, held special significance, and it was said that their sum would reveal the key to unlocking a powerful relic called the *Orb of Truth*.

However, the Orb could only be activated if the exact sum of all the Golden Numbers less than or equal to a specific number, *N*, was found. No one knew how to calculate this sum accurately, as manual calculations were tedious and error-prone. The Orb remained still and mysterious in the center of the Temple of Eternal Sequences.

The Challenge of the Young Programmer

Young apprentice named Auron was passionate about solving problems using modern tools. While the wise mathematicians debated and wrote endless lists of numbers, Auron had a revolutionary idea: to use the newly invented calculation machine to decipher the riddle.

Auron devised a method to identify whether a number was "golden" (a prime). He also created a system to sum them quickly. The key was to reduce the time spent manually reviewing each number.

Acting as Auron, your task is to create an application that takes a number as input and returns the exact sum of all prime numbers less than the specified number.

Input

The input consists of a positive integer

Output

The output should be the golden number

Example 1

Input

```
10
```

Output

```
17
```

Example 2

Input

```
20
```

Output

```
77
```

Python

```python
def es_primo(num):
    if num < 2:
        return False
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False
    return True

def suma_primos(N):
    if N <= 2:
        print(f"0")
        return
    suma = 0
    for i in range(2, N + 1):
        if es_primo(i):
            suma += i
    print(f"{suma}")

# Solicitar entrada al usuario
N = int(input(""))
suma_primos(N)
```

18 Single Neuron

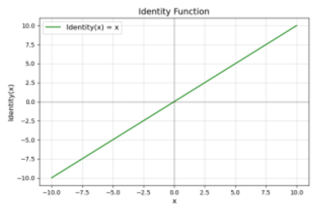
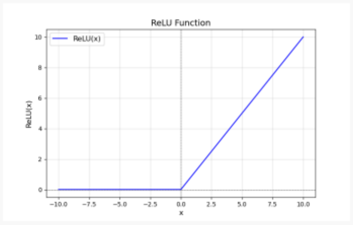
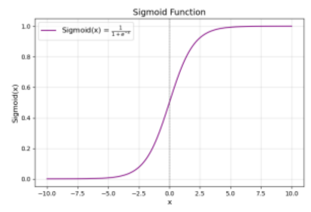
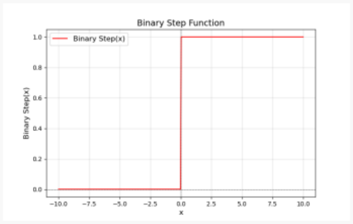
8 points

Introduction

Have you heard about Artificial Intelligence and Deep Learning? Technologies like ChatGPT are powered by Neural Networks (NN), a powerful tool in the AI field. Neural Networks are made up of connected neurons, which process information to generate insights. While super large Neural Networks can understand and emulate human-like conversations, the foundation of these systems is the single neuron.

In this problem, you are tasked to program a single neuron of a Neural Network. A neuron takes multiple inputs, processes them using weights and biases, and applies an activation function to produce an output.

The activation function can be as simple as a linear function, or more complex as the sigmoid function. Find below examples of activation functions.

Name	**Activation Function**	**Plot**
Identity	$g(x) = x$	
Rectified Linear Unit (ReLU)	$\mathrm{ReLU}(x) = \begin{cases} x, & \text{if } x \geq 0, \\ 0, & \text{if } x < 0. \end{cases}$	
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	
Binary Step	$\mathrm{BinStep}(x) = \begin{cases} 1, & \text{if } x \geq 0, \\ 0, & \text{if } x < 0. \end{cases}$	

The Single Neuron must accomplish the following requirements:

- Must be able to accept from 1 to 100 inputs.
- The number of weights must equal the number of inputs

To calculate the weighted sum to be used in the function, you must add the multiplication of each input with its corresponding weight and add the bias:

$$weighted\_sum = \left(\sum i * w\right) + bias$$

Input

The input of this problem consists in 4 lines:

- First line is the neuron input.
- Second line are neuron weights.
- Third line corresponds the neuron bias.
- Fourth line corresponds to the activation function. The allowed activation functions are: *identity, relu, sigmoid.*

All numeric inputs may contain decimal values.

Output

The single neuron will output the result of the weighted sum and the function activation.

Example 1

Input

```
3,10,-4
0.1,0.2,0.7
1
identity
```

Output

```
Output: 0.50
```

Example 2

Input

```
3,10,-4
0.1,0.2,0.7
2
relu
```

Output

```
Output: 1.50
```

Python

```python
import sys
import math

def identity(x):
    return x

def relu(x):
    return max(0, x)

def sigmoid(x):
    return 1 / (1 + math.exp(-x))

# Read inputs
line = input().strip()
line2 = line.split(",")
inputs = list(map(float, line2))
#print(f'line={line}, line2={line2}, inputs={inputs}')

line = input().strip()
line2 = line.split(",")
weights = list(map(float, line2))
#print(f'line={line}, line2={line2}, weights={weights}')
bias = float(input().strip())

activation_function = input().strip()

# Ensure the number of inputs and weights are the same
if len(inputs) != len(weights):
    print("Error: The number of inputs must be equal to the number of weights.")
    sys.exit(1)

# Compute the weighted sum
weighted_sum = sum(i * w for i, w in zip(inputs, weights)) + bias
if activation_function == "identity":
    output = identity(weighted_sum)
elif activation_function == "relu":
    output = relu(weighted_sum)
elif activation_function == "sigmoid":
    output = sigmoid(weighted_sum)
else:
    print("Error: Unsupported activation function.")
    sys.exit(1)

# Output the result
print(f'Output: {output:.2f}')
```

19 Vigenère Cipher Encoder

8 points

Introduction

During World War II, coded messages were often used to keep sensitive information secret. One such method of encoding messages is the Vigenère cipher. This cipher uses a keyword to encrypt a message by shifting the letters of the alphabet. The result is a message that appears scrambled and is difficult to read without the correct keyword.

In this problem, you will create a program to encrypt a plaintext message using the Vigenère cipher. The encryption process involves repeating the keyword to match the length of the message and shifting each letter according to the corresponding letter in the keyword. The output is an encrypted message that appears scrambled to anyone without the keyword.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Keyword: `LEMON`
Message: `ATTACKATDAWN`

	A	T	T	A	C	K	A	T	D	A	W	N
	0	19	19	0	2	11	0	19	3	0	22	13
	L	E	M	O	N	L	E	M	O	N	L	E
+	11	4	12	14	13	11	4	12	14	13	11	4
MOD(26)	11	23	31	14	15	22	4	31	17	13	33	17
	11	23	5	14	15	22	4	5	17	13	7	17
	L	X	F	O	P	V	E	F	R	N	H	R

```
A (0)  + L (11) → L (11)
T (19) + E (4)  → X (23)
T (19) + M (12) → F (5)
                  (since 31 % 26 = 5)
A (0)  + O (14) → O (14)
C (2)  + N (13) → P (15)
K (10) + L (11) → V (21)
A (0)  + E (4)  → E (4)
T (19) + M (12) → F (5)
                  (since 31 % 26 = 5)
D (3)  + O (14) → R (17)
A (0)  + N (13) → N (13)
W (22) + L (11) → H (7)
                  (since 33 % 26 = 7)
N (13) + E (4)  → R (17)
```

Input

The input will be provided in the following format:

- The first line contains the keyword, consisting of uppercase letters (A-Z).
- The second line contains the plaintext message, also consisting of uppercase letters (A-Z).

Output

The output should be a single line containing the encrypted message. Ensure the output is correctly formatted, as the judge will compare your output with the expected output using exact matching.

Example 1

Input

```
LEMON
ATTACKATDAWN
```

Output

```
LXFOPVEFRNHR
```

Example 2

Input

```
KEY
HELLOAGAIN
```

Output

```
RIJVSYQEGX
```

Python

```python
keyword = str(input())
message = str(input())

alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
keyword = keyword.upper()
message = message.upper()
encrypted_message = []

keyword_length = len(keyword)
for i, char in enumerate(message):
    if char in alphabet:
        key_char = keyword[i % keyword_length]
        key_shift = alphabet.index(key_char)
        plain_index = alphabet.index(char)
        enc_index = (plain_index + key_shift) % 26
        encrypted_message.append(alphabet[enc_index])
    else:
        encrypted_message.append(char)  # Non-alphabetic characters are not
changed

print(''.join(encrypted_message))
```

20 A Polynomial Spell

10 points

Introduction

In the magical land of Algebrania, the kingdom's prosperity depends on the harmony of its polynomial spells. The Royal Mathemagician, tasked with maintaining this harmony, has discovered an ancient scroll that reveals a powerful spell for multiplying polynomials. However, the scroll is incomplete, and the Mathemagician needs your help to decipher the missing part.

You are given two polynomials, each represented by a list of coefficients. Your task is to write a program that computes the product of these two polynomials and returns the resulting polynomial in the same format. By doing so, you will help restore balance to the kingdom and ensure its continued prosperity.

Input

The input consists of two lines. Each line contains a space-separated list of integers representing the coefficients of a polynomial. The first integer corresponds to the coefficient of the highest degree term, and the last integer corresponds to the constant term.

Output

The output should be a single line containing a space-separated list of integers representing the coefficients of the resulting polynomial after multiplication.

In the example 1, the first polynomial is ($3x^2 + 2x + 1$) and the second polynomial is ($x^2 - 1$). So, the product of these polynomials is ($3x^4 + 2x^3 - 2x^2 - 2x - 1$).

Example 1

Input

```
3 2 1
1 0 -1
```

Output

```
3 2 -2 -2 -1
```

Example 2

Input

```
-5 0
4 -2
```

Output

```
-20 10 0
```

Python

```python
# Reading polynomial coefficients from the user
firstPol = input().split()
secondPol = input().split()

# Finding out the grades of the polynomials
firstGrade = len(firstPol) - 1
secondGrade = len(secondPol) - 1
resGrade = firstGrade + secondGrade

# Creating a list for the result polynomial
resPol = [0] * (resGrade + 1)

# Multiplying the polynomials
for i in range(len(firstPol)-1, -1, -1):
    for j in range(len(secondPol)-1, -1, -1):
        resPol[i+j] += int(firstPol[i]) * int(secondPol[j])

# Print the result polynomial
print(*resPol)
```

21 Viking Ascent

10 points

Introduction

Young Vikings, to entertain themselves while their elders were at war, played this ancient game. It consisted of each player rolling dice a certain number of times, which they defined before starting to play. The first roll of each player's dice always added to their score, but subsequent rolls were added if the previous number was even and subtracted if it was odd. The lowest value a score can have at any time is zero. The player with the most points at the end of the rounds wins.

If several players end up with the same score, the winner is the one who has reached the highest number of points in any round.

Write a program that calculates each player score and shows which one is the winner.

Input

The input consists of multiple lines, with numbers separated by commas, representing the rolls of each player.

Output

The output should be a string containing the winner's number and score, formatted as: "Player *x* is the winner with *y* points".

Example

In this example, these are the totals:

- Player 1: 7 total points, reaching a maximum of 13 points.
- Player 2: 5 total points, reaching a maximum of 21 points.
- Player 3: 10 total points, reaching a maximum of 13 points.

Input

```
1,3,4,2,6,5,6,3,4,1
2,4,6,4,5,5,5,3,4,1
1,4,5,3,2,4,6,3,1,2
```

Output

```
Player 3 is the winner with 10 points
```

Python

```python
class User:
    def __init__(self):
        self.dices = 0  # Número de dados lanzados
        self.points = 0  # Puntos actuales
        self.max = 0  # Puntos máximos alcanzados


def calculate_user(line):
    user = User()
    tokens = line.split(",")
    l_substract = False

    for token in tokens:
        user.dices += 1  # Incrementar el contador de dados
        value = int(token.strip())  # Convertir el token a número entero
        if l_substract:
            user.points -= value
            if user.points < 0:
                user.points = 0  # No permitir puntos negativos
        else:
            user.points += value  # Sumar los puntos

        # Cambiar la lógica de restar/sumar dependiendo si el valor es par o
impar
        l_substract = value % 2 != 0

        # Actualizar el puntaje máximo si es necesario
        if user.points > user.max:
            user.max = user.points

    return user


def main():
    lines = []
    users = []

    # Leer múltiples líneas hasta que se ingrese una línea vacía
    while True:
        line = input("")
        if not line:  # Si la línea está vacía, salir del bucle
            break
        lines.append(line)
        users.append(calculate_user(line))
    # Verificar cuál línea tiene más puntos
    max_points = -1
    max_line_index = -1

    for i, user in enumerate(users):
        if user.points > max_points:
            max_points = user.points
            max_line_index = i

    # Mostrar el resultado
    if max_line_index != -1:
        print(f"Player {max_line_index + 1} is the winner with
{users[max_line_index].points} points")
    else:
        print("No se ingresaron líneas válidas.")


if __name__ == "__main__":
    main()
```

22 Undo-Redo Action

12 points

Introduction

Imagine you're developing a text editor and need to implement robust undo and redo functionalities. Your task is to create a program that efficiently handles a sequence of operations, allowing users to undo their last action or redo it if needed. This feature is crucial for enhancing user experience by providing flexibility in editing.

Input

The input will consist of a sequence of commands. The commands can be:

- A string containing a line of text you want to write.
- The word "undo" to revert the last action if any.
- The word "redo" to repeat the last undone action if any.
- The character '#' to print the ending result and stop the inputs.

Output

The output should be the resulting string after performing all the actions.

Example 1

Input

```
Welcome to Codewars
2024
undo
2025
#
```

Output

```
Welcome to Codewars 2025
```

Example 2

Input

```
Welcome
Codewars
undo
to
redo
#
```

Output

```
Welcome to Codewars
```

Python

```python
class UndoRedoStack:
    def __init__(self):
        self.history = []
        self.redo_stack = []
        self.current_state = ""

    def perform(self, action):
        self.history.append(action)
        self.update_state()

    def undo(self):
        if self.history:
            action = self.history.pop()
            self.redo_stack.append(action)
            self.update_state()

    def redo(self):
        if self.redo_stack:
            action = self.redo_stack.pop()
            self.history.append(action)
            self.update_state()

    def update_state(self):
        self.current_state = " ".join(self.history)

    def get_state(self):
        return self.current_state

stack = UndoRedoStack()
while True:
    command = input().strip()
    if command == "#":
        print(stack.get_state())
        break
    elif command == "undo":
        stack.undo()
    elif command == "redo":
        stack.redo()
    else:
        stack.perform(command)
```

23 Gladiators Clash

12 points

Introduction

Two groups of gladiators are fighting over a territory. To avoid a large-scale battle, each group decides to send the same number, n, of their best gladiators. Each group leader submits a list of size n representing the power of each of their gladiators.

In each round, one gladiator fights against another. The gladiator with less power is defeated, while the one with more power wins but loses part of their strength (equal to the power of the defeated gladiator) for future battles. Gladiators fight in the order they are listed (from left to right).

The group with any gladiators still standing with power at the end will be victorious. If all gladiators in both groups have 0 power at the end, the result is a draw. Similarly, if both teams choose not to send any fighters (an empty list, `[ ]`), the result is also a draw.

Input

Two lines, each containing a list of the power of each fighter.

Output

The winner and a list of the remaining power for each fighter in the winning group after the battle.

Example 1

Input

```
[2, 3, 4]
[3, 5, 6]
```

Output

```
Group 2 Wins! [0, 0, 5]
```

Example 2

Input

```
[2, 3, 4, 5]
[4, 3, 2, 5]
```

Output

```
Draw!
```

Example 3

Input

```
[10, 1, 8]
[2, 3, 4]
```

Output

```
Group 1 Wins! [1, 1, 8]
```

Python

```python
group1 = input()
group2 = input()
index1 = 0
index2 = 0
# Convert the input strings to lists of integers
size = len(group1)
if size>2:
    group1 = [int(x) for x in group1[1:-1].split(",")]
    group2 = [int(x) for x in group2[1:-1].split(",")]
    size = len(group1)
else:
    size=0
# Loop through the fighters of both groups
while index1 < size and index2 < size:
    if group1[index1] > group2[index2]:
        # If group1's fighter has more power, they win this round
        # Reduce group1's fighter power by group2's fighter's power
        group1[index1] -= group2[index2]
        # Set group2's fighter's power to 0 (they are defeated)
        group2[index2] = 0
        # Move to the next fighter in group2
        index2 += 1
    elif group2[index2] > group1[index1]:
        group2[index2] -= group1[index1]
        group1[index1] = 0
        index1 += 1
    else:
        group1[index1] = 0
        group2[index2] = 0
        index1 += 1
        index2 += 1
# Determine the winner based on which group has remaining fighters
if index1 < index2:
    print("Group 1 Wins! ", group1)
elif index2 < index1:
    print("Group 2 Wins! ", group2)
else:
    print("Draw!")
```

24 EHB League

13 points

Introduction

The college e-handball league (EHB) enters its final phase, with the top eight teams battling for the championship ring awarded to the number 1 team in the internal collegiate e-handball league. In this phase (the quarterfinals), teams from different conferences (West and East) will face each other, with four teams in each conference.

The organization seeks the cooperation of all teams participating in CodeWars 2025 tournament to proceed with the draw. Teams will be ordered alphabetically within each conference. Then, the first team from each conference will play against each other, the second team from each conference will play against each other, and so on.

However, there is still one step remaining: associating a date with each match. The quarterfinal phase will be played over four days: Monday, Tuesday, Thursday, and Friday. (Note: There will be no games on Wednesdays.) One match will be played each day. The pairing of each match with its corresponding day will be based on the position of the first letter of each team's name in the alphabet. Therefore, the match scheduled for Monday will involve the teams whose sum of the positions of their first letters is closest to the beginning of the alphabet.

The teams whose first letters appear in second position alphabetically will play on Tuesday, and so on for the third and fourth positions, which will play on Thursday and Friday, respectively.

For example, let's imagine that Madrid plays against Barcelona and Valencia plays against Bilbao. The letter positions are as follows:

- Barcelona (B = 2) vs. Madrid (M = 13), giving a total of 2 + 13 = 15.
- Valencia (V = 23) vs. Bilbao (B = 2), giving a total of 23 + 2 = 25.

In this case, the match with the lowest sum (15) will be played on Monday, so the game between Barcelona and Madrid will take place on Monday, and the game between Valencia and Bilbao will take place on Tuesday.

You are asked to develop a program that receives the list of the eight participating teams along with their conference and formalizes the quarterfinal draw.

Input

The input data will consist of a single line containing:

- Team name.
- Conference to which it belongs (Este-Oeste, North South, East-West)…. Whatever is the conference, but always there will be only two alternatives

Example: Team1-Conf1, Team2-Conf2, Team3-Conf1… until 8 teams are introduced.

Output

The output will consist of three lines with the following messages:

1. Conference XXX (The first one received in the input): Alphabetically ordered list of teams belonging to this conference, separated by commas and a space.
2. Conference YYY (The second one received in the input): Alphabetically ordered list of teams belonging to this conference, separated by commas and a space.
3. Pairings: Alphabetically ordered list of pairings (match day), separated by commas and a space.

Each line should display the respective conference and match pairings clearly.
See examples for clarity and spacing issues

Example 1

Input

```
Captors-Este, Bullets-Oeste, Wizards-Este, Parrots-Oeste, Flippers-Oeste,

Goals-Este, Sockets-Oeste, Lentils-Este
```

Output

```
Conference Este: Captors, Goals, Lentils, Wizards
Conference Oeste: Bullets, Flippers, Parrots, Sockets
Pairings: Captors-Bullets (Monday), Goals-Flippers (Tuesday), Lentils-Parrots
(Thursday), Wizards-Sockets (Friday)
```

Example 2

Input

```
Unixes-North, HackersLove-North, GigabyteTM-South, Anonymous-North,

CodeWarriors-South, ErrorsTeam-South, Mandalorians-North, Devils-South
```

Output

```
Conference North: Anonymous, HackersLove, Mandalorians, Unixes
Conference South: CodeWarriors, Devils, ErrorsTeam, GigabyteTM
Pairings:  Anonymous-CodeWarriors  (Monday),  HackersLove-Devils  (Tuesday),
Mandalorians-ErrorsTeam (Thursday), Unixes-GigabyteTM (Friday)
```

Python

```python
def parse_input(input_string: str) -> list:
    """Parse input string into list of team and conference tuples."""
    teams_data = []
    teams = input_string.strip().split(',')

    # Check for uniqueness of starting letters
    first_letters = set()

    for team in teams:
        try:
            team_name, conference = team.strip().split('-')
            team_name = team_name.strip()
            conference = conference.strip()

            # Check if the first letter of the team name is unique
            first_letter = team_name[0].upper()
            if first_letter in first_letters:
                print(f"Error: Team '{team_name}' starts with the same letter as 
another team.")
                return []
            first_letters.add(first_letter)

            teams_data.append((team_name, conference))

        except ValueError:
            print(f"Error: Invalid input format for team '{team}'. Expected 
'team-conference'.")
            return []

    return teams_data

def create_match_schedule(teams_data: list) -> None:
    """Creates and prints match schedule for teams from two conferences."""
    # Separate teams by conference
    conference_teams = {}

    for team, conference in teams_data:
        if conference not in conference_teams:
            conference_teams[conference] = []
        conference_teams[conference].append(team)

    # Ensure there are exactly two distinct conferences
    if len(conference_teams) != 2:
        print("Error: The input must contain exactly two distinct conferences.")
        return
    # Extract conference names and teams
    conference_names = list(conference_teams.keys())
    conference1, conference2 = conference_names[0], conference_names[1]

    # Sort teams alphabetically in each conference
    teams1, teams2 = sorted(conference_teams[conference1]),
sorted(conference_teams[conference2])

    # Print conferences with sorted teams
    print(f"Conference {conference1}: {', '.join(teams1)}")
    print(f"Conference {conference2}: {', '.join(teams2)}")

    # Handle case where the number of teams in each conference is not equal
    min_teams = min(len(teams1), len(teams2))
    matches = []

    # Create matches by pairing teams from both conferences
    for i in range(min_teams):
        matches.append((teams1[i], teams2[i]))

    if len(teams1) > len(teams2):
        print(f"Warning: {len(teams1) - len(teams2)} teams from {conference1}
will not have an opponent.")
    elif len(teams2) > len(teams1):
        print(f"Warning: {len(teams2) - len(teams1)} teams from {conference2}
will not have an opponent.")

    # Calculate letter scores and assign days
    match_scores = []
    for team1, team2 in matches:
        # Convert first letters to their position in the alphabet (1-based)
        score = (ord(team1[0].upper()) - 64) + (ord(team2[0].upper()) - 64)
        match_scores.append((score, team1, team2))

    # Sort matches by score
    match_scores.sort(key=lambda x: x[0])

    # Assign days to matches
    days = ["Monday", "Tuesday", "Thursday", "Friday"]

    # Create pairings string
    pairings = []
    for i, (score, team1, team2) in enumerate(match_scores):
        if i < len(days):
            pairings.append(f"{team1}-{team2} ({days[i]})")
        else:
```

```python
            pairings.append(f"{team1}-{team2} (Additional Day)")

    # Print pairings
    print(f"Pairings: {', '.join(pairings)}")

def main():
    """Main function to drive the program."""
    # Get input from user
    input_string = input("")
    teams_data = parse_input(input_string)

    # Only proceed if input is valid (no empty list returned)
    if teams_data:
        create_match_schedule(teams_data)

if __name__ == "__main__":
    main()
```

25 Ancient Time Traveler

13 points

Introduction

As a time traveler, you find yourself stranded in ancient Rome during one of its most pivotal moments in history. The only way to prove your allegiance and earn the trust of the Roman Senate, besides speaking Latin, is to master their numeric system. Without this, you risk being labeled an outsider—or worse, a spy sent by their enemies.

To gain the Senate's trust, you must write a program capable of:

- Converting decimal numbers into Roman numerals, to speak the language of the Empire's accounts and decrees.
- Validating Roman numerals, ensuring that they adhere to the rules of the Roman numeric system, as even a single mistake could reveal your foreign origins.
- Translating Roman numerals back into decimal numbers, so you can accurately decipher ancient texts and military orders.

Rules of the Roman Numeric System:

Roman numerals are formed using the following symbols:

```
I = 1, V = 5, X = 10, L = 50, C = 100, D = 500, M = 1000
```

And numbers are created following some rules:

- Smaller values that follow larger values are added:

```
VI = 5 + 1 = 6
XII = 10 + 1 + 1 = 12
```

- A smaller value that precedes a larger value is subtracted:

```
IV = 5 - 1 = 4
IX = 10 - 1 = 9
```

- No symbol may be repeated more than three times consecutively.

```
III = 3 is valid, but IIII is not
```

- Only specific combinations are allowed for subtraction:

```
I precedes V or X

X precedes L or C

C precedes D or M
```

The program will allow to:

- Translate a decimal number (1-3999) into its Roman numeral equivalent.
- Translate a roman number into decimal equivalent.

Input

The input consists of a single line, that can a numeric value or an upper-case text. If the input is a numeric value, the expected action is to convert this number into roman numbers. The number will be a positive integer lower than 4000. If the input is a text value, the expected output is a positive numeric decimal value.

Output

The output should be a numeric value or an upper-case text with the appropriate roman equivalent. It can be also ERROR if the numeric value is higher than 3999 or the text representing the roman number is incorrectly written.

Example 1

Input

```
1990
```

Output

```
MCMXC
```

Example 2

Input

```
1234
```

Output

```
MCCXXXIV
```

Example 3

Input

```
4000
```

Output

```
ERROR
```

Example 4

Input

```
MCMI
```

Output

```
1901
```

Example 5

Input

```
XXXX
```

Output

```
ERROR
```

Python

```python
import re

def int_to_roman(num):
    if num <= 0 or num >= 4000:
        return "ERROR"

    val = [
        1000, 900, 500, 400,
        100, 90, 50, 40,
        10, 9, 5, 4,
        1
    ]
    syb = [
        "M", "CM", "D", "CD",
        "C", "XC", "L", "XL",
        "X", "IX", "V", "IV",
        "I"
    ]

    roman_num = ''
    i = 0
    while num > 0:
        for _ in range(num // val[i]):
            roman_num += syb[i]
            num -= val[i]
        i += 1
    return roman_num

def roman_to_int(roman):
    # Regular expression to validate a correct Roman numeral
    roman_pattern = r"^M{0,3}(CM|CD|D?C{0,3})(XC|XL|L?X{0,3})(IX|IV|V?I{0,3})$"

    # Validate if the Roman numeral string matches the pattern
    if not re.match(roman_pattern, roman):
        return "ERROR"

    roman_dict = {
        'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100,
        'D': 500, 'M': 1000
    }
    total = 0
    prev_value = 0

    for char in reversed(roman):
        if char not in roman_dict:
            return "ERROR"

        value = roman_dict[char]

        if value < prev_value:
            total -= value
        else:
            total += value
        prev_value = value

    return total

def process_input(input_value):
    # Check if the input is numeric
    if input_value.isdigit():
        num = int(input_value)
        if 0 <= num <= 3999:
            return int_to_roman(num)
        else:
            return "ERROR"

    # Check if the input is a valid Roman numeral
    elif input_value.isalpha() and input_value.isupper():
        return roman_to_int(input_value)

    else:
        return "ERROR"

# Test the function with some examples
text=input("")
print(process_input(text))
```

26 The Automatic Coach

13 points

Introduction

As it is well known, new technologies combined with statistics are the future of any sport. We were asked by a futbasket team (a new sport we have just invented) to develop a program that determines the line up for each game based on the knowledge of each player's skills.

The skills are as follows:

- Near shooting
- Long-range shooting
- Passing
- Dribbling
- Defense
- Attack

Each player will have a value between 0 and 10 for each skill. Using these values, we must determine the best possible line up by selecting one player for each position. Each position has associated coefficients for each skill. The positions and their associated coefficients are as follows:

Skill	**Guard**	**Shooting Guard**	**Small Forward**	**Power Forward**	**Center**
Near Shooting	0	0	0.2	0.4	0.2
Long-range Shooting	0.2	0.45	0.3	0	0
Passing	0.45	0.15	0	0	0
Dribbling	0.15	0.35	0.3	0.05	0
Defense	0.2	0.05	0.1	0.25	0.3
Attack	0	0	0.1	0.3	0.5

Input

The input data will consist of several lines. The first line will contain the name of the team. The second will be an integer, N, the number of players on the team. Then, there will be N lines with data for each player, following this structure:

- The first part of each line contains the player's name.
- The following six integers represent the player's ratings for the six skills listed above (each value between 0 and 10).

The team's name and the number of players will be enclosed in angle brackets (< and >), as will the player names and their skill values. The program should process this data and calculate the best possible squad by selecting players for each position in the following order:

1. Best Guard
2. Best Shooting Guard
3. Best Small Forward
4. Best Power Forward
5. Best Center

Once the line up is obtained, the overall team rating should be calculated as an average of the ratings of the players in each position. This rating will range from 0 to 10 and should be rounded to the nearest whole number (without decimal places).

Output

With this data, the program should display a single line with following result on the screen: "Line up of teamName: Guard Name, Shooting Guard Name, Small Forward Name, Power Forward Name, Center Name. Qualification teamValue." Where teamName will be the name of the team, PointGuardName, ShootingGuardName, etc will be the names of the players chosen for each position followed by the overall team qualification.

Example 1

Input

```
<Barsa>
<7>
<TerStegen 8 10 8 9 8 7>
<Lewandowski 10 6 7 7 9 10>
<Rafinha 9 10 8 9 8 9>
<Pedri 10 4 4 5 9 10>
<LamineYamal 6 9 10 10 8 4>
<Gavi 5 10 8 10 7 4>
<Casado 7 10 6 4 5 8>
```

Output

```
Line up of Barsa: LamineYamal, Gavi, Rafinha, Lewandowski, Pedri. Qualification 9.
```

Example 2

Input

```
<Madrid>
<8>
<Vinicius 8 10 8 9 8 7>
<Carbajal 7 10 6 4 5 8>
<Mbappe 10 6 7 7 9 10>
<Modric 9 10 8 9 8 9>
<Guler 6 9 10 10 8 4>
<Asencio 5 10 8 10 7 4>
<Bellingham 10 4 4 5 9 10>
<Camavinga 5 8 10 9 9 4>
```

Output

```
Line up of Madrid: Guler, Asencio, Modric, Mbappe, Bellingham. Qualification 9.
```

Python

```
def calcular_calificacion(jugador, coeficientes):
    """
    Calcula la calificación total de un jugador para una posición dada,
basándose
    en las habilidades del jugador y los coeficientes correspondientes.
    """
    calif=0
    for i in range(6):  # Suma ponderada de habilidades
        calif=calif+jugador[i]*coeficientes[i]
#        print (jugador[i], " * ", coeficientes[i] , " = ", calif)
    #calificacion = sum(jugador[i] * coeficientes[i] for i in range(6))  # Suma
ponderada de habilidades
    #return calificacion
    #print (calif)
    return calif


def main():
    # Coeficientes por cada habilidad para las posiciones
    coeficientes = {
        'POINT GUARD': [0, 0.2, 0.45, 0.15, 0.2, 0],
        'SHOOTING GUARD': [0, 0.45, 0.15, 0.35, 0.05, 0],
        'SMALL FORWARD': [0.2, 0.3, 0, 0.3, 0.1, 0.1],
        'POWER FORWARD': [0.4, 0, 0, 0.05, 0.25, 0.3],
        'CENTER': [0.2, 0, 0, 0, 0.3, 0.5]
    }

    # Lectura de datos
    team_name = input().strip()[1:-1]  # Eliminar los signos '<' y '>'
    num_jugadores = int(input().strip()[1:-1])  # Número de jugadores, quitar
'<' y '>'

    jugadores = []
    jugadores2 = []
    for _ in range(num_jugadores):
        # Lectura de cada jugador
        datos = input().strip()[1:-1].split()  # Eliminar los signos '<' y '>'
        nombre = datos[0]
        habilidades = list(map(int, datos[1:]))  # Convertir las habilidades en
enteros
        jugadores.append((nombre, habilidades))
        jugadores2.append((nombre, habilidades))

    # Alineación vacía para los jugadores seleccionados en cada posición
    alineacion = {
        'POINT GUARD': None,
        'SHOOTING GUARD': None,
        'SMALL FORWARD': None,
        'POWER FORWARD': None,
        'CENTER': None
    }

    # Buscar el mejor jugador para cada posición
    for posicion in alineacion:
        mejor_calificacion = -1
        mejor_jugador = None
        for jugador in jugadores:
            nombre, habilidades = jugador
            #print("calculamos habilidades de ", nombre)
            calificacion = calcular_calificacion(habilidades,
coeficientes[posicion])
            if calificacion > mejor_calificacion:
                mejor_calificacion = calificacion
                mejor_jugador = nombre
        alineacion[posicion] = mejor_jugador
        # Eliminar el jugador seleccionado de la lista de jugadores para no
repetirlo
        jugadores = [j for j in jugadores if j[0] != mejor_jugador]

    # Calcular la calificación global del equipo
    calificacion_equipo = 0
    for posicion in alineacion:
        nombre = alineacion[posicion]
        for jugador in jugadores2:
            #print (nombre)
            #print(jugador[0])
            #print("--")
            if jugador[0] == nombre:
                habilidades = jugador[1]
                calificacion_equipo += calcular_calificacion(habilidades,
coeficientes[posicion])
                #print(calificacion_equipo)
                break

    # Calificación global redondeada a un valor entero entre 0 y 5
    calificacion_equipo = round(calificacion_equipo / 5)

    # Imprimir la alineación y la calificación
    #alineacion_str = ", ".join(alineacion[posicion] for posicion in alineacion)
    tmp_array = [alineacion[posicion] for posicion in alineacion]
    tmp_array.sort()  # Ordenar alfabéticamente
    # Crear la cadena final
    alineacion_str = ", ".join(tmp_array)

    print(f"Alignment of {team_name}: {alineacion_str}. Qualification
{calificacion_equipo}.")


if __name__ == "__main__":
    main()
```

27 LaGrange's Four-Square

13 points

Introduction

This theorem states: *any positive integer can be decomposed as the sum of four squares of integers.* For example:

$$23 = 3^2 + 3^2 + 2^2 + 1^2$$
$$1 = 1^2 + 0^2 + 0^2 + 0^2$$

In some cases, there may be multiple valid alternatives. Consider the case of number 59:

$$59 = 7^2 + 3^2 + 1^2 + 0^2$$
$$59 = 5^2 + 5^2 + 3^2 + 0^2$$
$$59 = 5^2 + 4^2 + 3^2 + 3^2$$

Your task is to write a program that returns the decomposition into 4 squares of a number up to 999.999, choosing the decomposition where the highest number is the largest first square. For example, in the previous case, the program would return the first option ($59 = 7^2 + 3^2 + 1^2 + 0^2$). If a number has several alternatives with the same highest square, the option with the second highest square should be chosen. Consider the case of number 43:

$$43 = 4^2 + 3^2 + 3^2 + 3^2$$
$$43 = 5^2 + 3^2 + 3^2 + 0^2$$
$$43 = 5^2 + 4^2 + 1^2 + 1^2$$

The third option ($43 = 5^2 + 4^2 + 1^2 + 1^2$) should be chosen (because 5 is the first squared integer for both solutions, but then 4 is the second squared integer only for the third solution and is greater than 3, which is the second squared integer for the second solution).

Input

A single number from 1 to 999999.

Output

The decomposition of the input number in squared integer integers following the Lagrange Four Square Theorem.

Example 1

Input

```
1
```

Output

```
1 = 1^2 + 0^2 + 0^2 + 0^2
```

Example 2

Input

```
59
```

Output

```
59 = 7^2 + 3^2 + 1^2 + 0^2
```

Example 3

Input

```
23
```

Output

```
23 = 3^2 + 3^2 + 2^2 + 1^2
```

Python

```python
import math

def factoring_lagrange(n):
    for a in range(int(math.sqrt(n)), -1, -1):
        for b in range(a, -1, -1):
            for c in range(b, -1, -1):
                remaining = n - a*a - b*b - c*c
                if remaining >= 0:
                    d = int(math.sqrt(remaining))
                    if a*a + b*b + c*c + d*d == n:
                        print(f"{n} = {a}^2 + {b}^2 + {c}^2 + {d}^2")
                        return
    print("No decomposition found.")

def main():
    number = int(input())
    if 1 <= number <= 999999:
        factoring_lagrange(number)
    else:
        print("Number must be between 1 and 999999.")

if __name__ == "__main__":
    main()
```

28 Ready Player One

17 points

Introduction

Directed by Steven Spielberg in 2018, this movie is based on Ernest Cline's best-selling book. Set in 2045, it features a virtual reality world called the OASIS, where players hunt for hidden Easter Eggs by solving pop culture puzzles. With over 120 character cameos, it offers endless rewatch value.

For example, did you notice the Ready Player One logo is a maze starting at the "R" and ending at the center of the "O" where the Golden Easter Egg lies?

Would you like to solve a maze? Consider a two-dimensional maze defined as a matrix of numbers like this:

```
1 1 1 1 1 1 1 1 0 1
1 3 1 0 1 0 0 0 0 1
1 0 1 0 0 0 1 1 0 1
1 0 1 1 1 1 1 0 0 1
1 0 1 0 0 0 0 0 0 1
1 0 1 0 1 0 1 0 0 1
1 0 1 0 1 0 0 0 0 0
1 0 1 0 1 0 1 1 0 1
1 0 0 0 1 0 0 0 0 1
1 1 1 0 1 1 1 1 2 1
```

where `0` is a safe place to walk, 1 is a maze wall, 2 represents the start point and 3 is the finish point.

There are four possible directions (`N` = North, `E` = East, `W` = West and `S` = South) to move inside the maze. You will receive a list of directions to follow. Like in a videogame, if you reach the finish point before exhausting all the received moves, you "Win". But if you collide with any wall or go beyond the maze border, you "Lose". And if you remain in the maze after using all the received moves, you are "Lost". Can you code a program that, given a maze and a list of directions, checks the outcome of following the list of direction moves?

Input

The input is a matrix of *n* rows and *m* columns, with each row on a separate line. The matrix ends with a line containing a single character '#'. The input ends with a line with containing the list of direction moves.

Output

The output prints the results of following the direction moves inside the maze: win, lose or lost.

Example 1

Input

```
1 1 1 1 1 1 1 1 0 1
1 3 1 0 1 0 0 0 0 1
1 0 1 0 0 0 1 1 0 1
1 0 1 1 1 1 1 0 0 1
1 0 1 0 0 0 0 0 0 1
1 0 1 0 1 0 1 0 0 1
1 0 1 0 1 0 0 0 0 0
1 0 1 0 1 0 1 1 0 1
1 0 0 0 1 0 0 0 0 1
1 1 1 0 1 1 1 1 2 1
#
N E E
```

Output

```
Lose
```

Example 2

Input

```
1 1 1 1 1 1 1 1 0 1
1 3 1 0 1 0 0 0 0 1
1 0 1 0 0 0 1 1 0 1
1 0 1 1 1 1 1 0 0 1
1 0 1 0 0 0 0 0 0 1
1 0 1 0 1 0 1 0 0 1
1 0 1 0 1 0 0 0 0 0
1 0 1 0 1 0 1 1 0 1
1 0 0 0 1 0 0 0 0 1
1 1 1 0 1 1 1 1 2 1
#
N N N E
```

Output

```
Lost
```

Example 3

Input

```
1 1 1 1 1 1 1 1 0 1
1 3 1 0 1 0 0 0 0 1
1 0 1 0 0 0 1 1 0 1
1 0 1 1 1 1 1 0 0 1
1 0 1 0 0 0 0 0 0 1
1 0 1 0 1 0 1 0 0 1
1 0 1 0 1 0 0 0 0 0
1 0 1 0 1 0 1 1 0 1
1 0 0 0 1 0 0 0 0 1
1 1 1 0 1 1 1 1 2 1
#
N W W W N N N N W W S S S S W W N N N N N N N
```

Output

```
Win
```

Python

```python
def read_maze():
    maze = []
    while True:
        row = input().strip()
        if row == '#':
            break
        maze.append(list(map(int, row.split())))
    return maze

def read_direction_moves():
    return list(map(str, input().split()))

def look_for_start_point(maze):
    for i in range(len(maze)):
        for j in range(len(maze[i])):
            if maze[i][j] == 2:
                return (i, j)

# Main program
result = ""

# Read the maze
maze = read_maze()

# Read the direction moves
direction_moves = read_direction_moves()

# Get start point in the maze
pos = look_for_start_point(maze)

# Follow the direction steps from the start point
for move in direction_moves:
    # Apply the move
    if move == 'N':
        pos = (pos[0] - 1, pos[1])
    elif move == 'S':
        pos = (pos[0] + 1, pos[1])
    elif move == 'E':
        pos = (pos[0], pos[1] + 1)
    elif move == 'W':
        pos = (pos[0], pos[1] - 1)

    # Check if the move arrived to the finish point
    if maze[pos[0]][pos[1]] == 3:
        result = "Win"
        break

    # Check if the move arrived to a wall
    if maze[pos[0]][pos[1]] == 1:
        result = "Lose"
        break

# If the result is empty, the player is lost in the maze
if result == "":
    result = "Lost"

print(result)
```

29 MineSweeper

19 points

Introduction

Have you ever played Minesweeper? It is a classic Windows game that tests the player's logical thinking skills.

The objective of the game is to uncover all the squares that do not contain mines without triggering any explosions by clicking on a square with a mine underneath. The locations of the mines are unknown and must be discovered through logical deduction. Clicking on the game board reveals the content beneath the chosen square or squares (a large number of blank squares can be uncovered at once if they are adjacent to each other).

Some squares are blank, while others contain numbers (ranging from 1 to 8). Each number represents the number of mines adjacent to that square. To avoid triggering a mine, you can mark the location of a suspected mine by right-clicking on the square. The game is won when all the empty squares have been uncovered without hitting a mine. Any remaining mines not flagged by the player are automatically marked by the computer. However, if the player makes a mistake and clicks on a mine, the game is lost. In this case, if the player mistakenly marks a safe square, it will appear either with a red "X" covering the mine (indicating that it is safe), or simply with the red "X" (also showing that it is safe).

Very well, all this is the theory. Now, you are tasked with implementing it in a way that takes standard input to define an 8x8 board. The board will contain 10 mines, and three positions will be specified where the player clicks. Your program should output the situation of the board after those three clicks or display "BUMMM" if a mine was clicked.

Input

The input consists of 9 lines. The first 8 are the board, the ninth are the 3 clicked positions. The possible characters of the board are "_" for a square without bomb and "O" for a square with bomb. Then the ninth line will be 3 positions of the board X,Y separated by semicolon which is where the user will click.

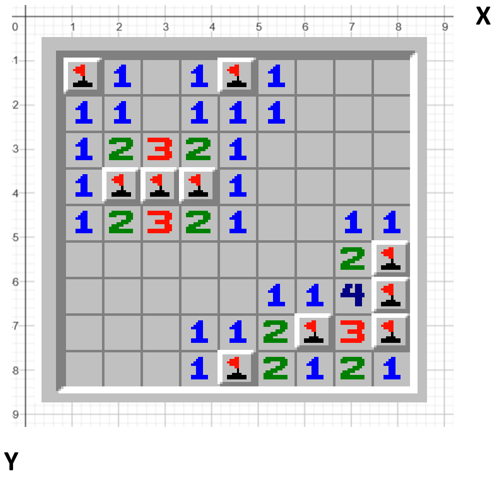
Output

The program should calculate where the user's attempts fall. There are several options:

- If any of the 3 rolls has landed on a bomb, the output will be BUMMM.
- If a roll lands on a square adjacent (horizontally, vertically or diagonally) to a bomb, the number of bombs around it will be displayed.
- If a roll falls on a square with no adjacent bombs, it will open the hole by clearing the squares around it that have no adjacent bombs.

And show the situation of the board after those three rolls.

The following image shows the Minesweeper board referenced over an X,Y axis:



Example 1

Input

```
_ _ _ _ _ _ 0 _
_ 0 _ _ _ _ _ _
_ _ _ _ 0 _ _ _
_ _ _ 0 _ _ _ _
0 _ _ _ _ _ _ 0
_ 0 _ 0 _ _ _ _
_ _ 0 _ _ 0 _ _
_ _ _ _ _ _ _ _
4,1; 7,3; 8,8
```

Output

```
_ _ 1     1 0 _
_ 0 1 1 1 2 1 1
_ _ _ _ 0 1
_ _ _ 0 _ 1 1 1
0 _ _ _ _ _ _ 0
_ 0 _ 0 _ _ 2 1
_ _ 0 _ _ 0 1
_ _ _ _ _ _ 1
```

Example 2

Input

```
_ _ _ _ _ _ 0 _
_ 0 _ _ _ _ _ _
_ _ _ _ 0 _ _ _
_ _ _ 0 _ _ _ _
0 _ _ _ _ _ _ 0
_ 0 _ 0 _ _ _ _
_ _ 0 _ _ 0 _ _
_ _ _ _ _ _ _ _
3,3; 5,5; 1,1
```

Output

```
1 _ _ _ _ _ 0 _
_ 0 _ _ _ _ _ _
_ _ 2 _ 0 _ _ _
_ _ _ 0 _ _ _ _
0 _ _ _ 2 _ _ 0
_ 0 _ 0 _ _ _ _
_ _ 0 _ _ 0 _ _
_ _ _ _ _ _ _ _
```

Example 3

Input

```
_ _ _ _ _ _ 0 _
_ 0 _ _ _ _ _ _
_ _ _ _ 0 _ _ _
_ _ _ 0 _ _ _ _
0 _ _ _ _ _ _ 0
_ 0 _ 0 _ _ _ _
_ _ 0 _ _ 0 _ _
_ _ _ _ _ _ _ _
5,5; 4,1; 1,4
```

Output

```
_ _ 1     1 0 _
_ 0 1 1 1 2 _ _
_ _ _ _ 0 _ _ _
1 _ _ 0 _ _ _ _
0 _ _ _ 2 _ _ 0
_ 0 _ 0 _ _ _ _
_ _ 0 _ _ 0 _ _
_ _ _ _ _ _ _ _
```

Example 4

Input

```
_ _ _ _ _ _ 0 _
_ 0 _ _ _ _ _ _
_ _ _ _ 0 _ _ _
_ _ _ 0 _ _ _ _
0 _ _ _ _ _ _ 0
_ 0 _ 0 _ _ _ _
_ _ 0 _ _ 0 _ _
_ _ _ _ _ _ _ _
7,1; 3,3; 1,1
```

Output

```
BUMMM
```

Python

```python
def imprimir_tablero(tablero):
    """Función para imprimir el tablero de forma legible."""
    for fila in tablero:
        print(" ".join(fila))

def contar_minas_adyacentes(tablero, x, y):
    """Cuenta cuántas minas están alrededor de la casilla (x, y)."""
    direcciones = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0),
(1, 1)]
    mina_count = 0
    for dx, dy in direcciones:
        nx, ny = x + dx, y + dy
        if 0 <= nx < 8 and 0 <= ny < 8 and tablero[nx][ny] == 'O':
            mina_count += 1
    return mina_count

def revelar_casilla(tablero, x, y, revelado):
    """Revela la casilla (x, y) y todas las casillas adyacentes vacías sin minas
adyacentes."""
    if revelado[x][y] != '_':
        return

    mina_count = contar_minas_adyacentes(tablero, x, y)

    if mina_count > 0:
        tablero[x][y] = str(mina_count)
    else:
        tablero[x][y] = ' '
        direcciones = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1,
0), (1, 1)]
        for dx, dy in direcciones:
            nx, ny = x + dx, y + dy
            if 0 <= nx < 8 and 0 <= ny < 8 and tablero[nx][ny] == '_' and
revelado[nx][ny] == '_':
                revelar_casilla(tablero, nx, ny, revelado)

def main():
    # Leer el tablero de 8x8
    tablero = [input().split() for _ in range(8)]

    # Leer las posiciones de los clics
    clics = input().split(";")

    # Convertir las posiciones a tuplas de enteros
    clics = [tuple(map(int, clic.split(","))) for clic in clics]
    # Crear una copia del tablero para marcar las casillas reveladas
    revelado = [['_'] * 8 for _ in range(8)]

    # Procesar los clics
    for y, x in clics:
        x -= 1  # Ajustar índices para que empiecen en 0
        y -= 1  # Ajustar índices para que empiecen en 0

        if tablero[x][y] == 'O':
            print("BUMMM")
            return

        # Si es una casilla vacía, revelamos la casilla
        revelar_casilla(tablero, x, y, revelado)

    # Imprimir el estado final del tablero
    imprimir_tablero(tablero)

if __name__ == "__main__":
    main()
```

30 Time Corridors

30 points

Introduction

Walking through the treacherous corridors of an ancient Egyptian pyramid you realize that these corridors are not just any ordinary paths; they span across three dimensions of space and one dimension of time. Your mission is to write a program to help you find the shortest path through this complex labyrinth, keeping in mind that time only moves forward.

In this adventure, you can change your speed while moving through the pyramid. This means that within a specific time slice, you can move as much as you want through the 3D space or even stay still before jumping to the next time slice. However, diagonal movement is not allowed, neither in space nor in time. You need to find a solution that requires the fewest movements in space, there may be multiple correct solutions (the system will check if the solution is valid).

Input

The first line will be the size of the space-time of the corridors inside our pyramid: Nx, Ny, Nz, Nt, all of them are positive integers and start at 0. For example, if Nx = 2, possible x positions are 0, and 1.

The second line is the starting point: Sx, Sy, Sz, St. Notice that St will always be 0, the start of time.

The third line is the ending point: Ex, Ey, Ez, Et where Et = Nt – 1, the end of time.

After these lines, there are Nx · Ny · Nz lines. Each line starts with 3 numbers that are the x, y, z coordinates of one point in space, followed by Nt values representing the evolution in time of this point in the time corridors. There are two possible values: a letter **X** for a wall you can't cross or a greater-than sign (**>**) meaning that is possible to move forward.

Output

Output will be composed by several lines with steps followed by the path through space-time in x, y, z, t coordinates and a final line with the total number of 3D (x,y,z) spaces crossed.

Example

Input

```
3 3 3 4
0 0 0 0
2 2 2 3
0 0 0 >XX>
0 0 1 >X>>
0 0 2 >X>>
0 1 0 >XX>
0 1 1 >X>>
0 1 2 >>>>
0 2 0 >>X>
0 2 1 >X>>
0 2 2 >>>>
1 0 0 >X>>
1 0 1 >X>>
1 0 2 >>>>
1 1 0 >X>>
1 1 1 >X>>
1 1 2 >>>>
1 2 0 >>>>
1 2 1 >X>>
1 2 2 >>>>
2 0 0 >>>>
2 0 1 >X>>
2 0 2 >>>>
2 1 0 >X>>
2 1 1 >X>>
2 1 2 >>>>
2 2 0 >>>>
2 2 1 >X>>
2 2 2 >>>>
```

Output 1

```
0 0 0 0
1 0 0 0
2 0 0 0
2 0 0 1
2 0 0 2
2 0 0 3
2 1 0 3
2 2 0 3
2 2 1 3
2 2 2 3
7
```

Output 2 (Alternative solution)

```
0 0 0 0
1 0 0 0
2 0 0 0
2 1 0 0
2 2 0 0
2 2 1 0
2 2 2 0
2 2 2 1
2 2 2 2
2 2 2 3
7
```

Example 2

Input

```
2 3 2 5
1 2 0 0
1 0 1 4
0 0 0 X>XXX
0 0 1 X>>X>
0 1 0 XXXXX
0 1 1 XX>>X
0 2 0 X>XXX
0 2 1 X>>X>
1 0 0 XXX>X
1 0 1 XX>>>
1 1 0 X>XXX
1 1 1 XX>XX
1 2 0 >>X>X
1 2 1 >>XX>
```

Output

```
1 2 0 0
1 2 0 1
0 2 0 1
0 2 1 1
0 2 1 2
0 1 1 2
1 1 1 2
1 0 1 2
1 0 1 3
1 0 1 4
6
```

Python

```python
import sys
from collections import deque

class Pos:
    def __init__(self, x, y, z, t, cube, parent=None):
        self.x = x
        self.y = y
        self.z = z
        self.t = t
        self.cube = cube
        self.parent = parent

    def clone(self):
        return Pos(self.x, self.y, self.z, self.t, self.cube, self)

def valid_position(n, X, Y, Z):
    if n.x < 0 or n.x >= X or n.y < 0 or n.y >= Y or n.z < 0 or n.z >= Z:
        return False
    elif n.cube[n.x][n.y][n.z] == '>':
        return True
    else:
        return False

def print_solution(c):
    path = []
    p = c
    while p is not None:
        path.append(p)
        p = p.parent
    path.reverse()

    stepsCount = 1
    i = 0
    for p in path:
        print(f"{p.x} {p.y} {p.z} {p.t}")
        if i > 0:
            if p.x != path[ i- 1].x or p.y != path[i - 1].y or p.z != path[i -
1].z:
                stepsCount = stepsCount + 1
        i = i + 1

    print(stepsCount)

def move(c, X, Y, Z, T, Ex, Ey, Ez, Et, timeline, moves):
    c.cube[c.x][c.y][c.z] = 'v'
    if c.x == Ex and c.y == Ey and c.z == Ez and c.t == Et:
        print_solution(c)
        sys.exit(0)

    if c.t + 1 < T:
        n = c.clone()
        n.t += 1
        n.cube = timeline[n.t]
        if valid_position(n, X, Y, Z):
            n.cube[n.x][n.y][n.z] = 'a'
            moves.append(n)

    for dx, dy, dz in [(1, 0, 0), (-1, 0, 0), (0, 1, 0), (0, -1, 0), (0, 0, 1),
(0, 0, -1)]:
        n = c.clone()
        n.x += dx
        n.y += dy
        n.z += dz
        if valid_position(n, X, Y, Z):
            n.cube[n.x][n.y][n.z] = 'a'
            moves.append(n)

def main():
    # Read input
    input_data = sys.stdin.read().split()
    idx = 0

    X = int(input_data[idx])
    idx += 1
    Y = int(input_data[idx])
    idx += 1
    Z = int(input_data[idx])
    idx += 1
    T = int(input_data[idx])
    idx += 1
    Sx = int(input_data[idx])
    idx += 1
    Sy = int(input_data[idx])
    idx += 1
    Sz = int(input_data[idx])
    idx += 1
    St = int(input_data[idx])
    idx += 1
    Ex = int(input_data[idx])
    idx += 1
    Ey = int(input_data[idx])
    idx += 1
    Ez = int(input_data[idx])
    idx += 1
    Et = int(input_data[idx])
    idx += 1

    # Read timeline
    timeline = []
    for _ in range(T):
        timeline.append([[['>'] * Z for _ in range(Y)] for _ in range(X)])

    for _ in range(X * Y * Z):
        x = int(input_data[idx])
        idx += 1
        y = int(input_data[idx])
        idx += 1
        z = int(input_data[idx])
        idx += 1
        seq = input_data[idx]
        idx += 1
        for t in range(T):
            timeline[t][x][y][z] = seq[t]

    # Start search from the initial position
    s = Pos(Sx, Sy, Sz, St, timeline[St])

    # The use of a deque and the popleft method indicates that the search is
    # performed in a BFS manner, exploring all possible moves at the current
    # depth before moving to the next depth level.
    moves = deque([s])

    # Breadth First Search (as soon as we find the Exit point for the first
time,
    # we know it's the shortest path in number of steps)
    while moves:
        move(moves.popleft(), X, Y, Z, T, Ex, Ey, Ez, Et, timeline, moves)

if __name__ == "__main__":
    main()
```