1 Welcome to Codewars I

1 points

Introduction

Now is the time to prove that you have what it takes to be a true warrior. Now is the time to start coding and having fun! As a rite of passage, you must create your very first program in this journey. However, we are feeling generous today, so we will not ask you to do something too difficult. For now you only have to write a program that takes no input and returns the string "Welcome to Codewars 2025!".

Exercise

Write a program that outputs the string "Welcome to Codewars 2025!".

Input

None.

Output

The string "Welcome to Codewars 2025!".

Example 1

Input

No input for this problem.

Output

```
Welcome to Codewars 2025!
```

Solutions

Python

```
def main():
    print("Welcome to Codewars 2025!")
if __name__ == "__main__":
    main()
```

C++

```
#include <iostream>
int main(void) {
    std::cout << "Welcome to Codewars 2025!" << std::endl;
    return 0;
}
```

Java

```
class welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Codewars 2025!");
    }
}
```

2 Welcome to Codewars II

2 points

Introduction

Handling input will also be important for the challenges you will face today. Now, you have to extend your first program a bit, so it takes a team name (like the name of your team, but it can be any arbitrary string) and outputs the string *Welcome to Codewars 2025, TeamName!*, where *TeamName* is the name of the team you input.

Exercise

Write a program that outputs the string *Welcome to Codewars 2025, TeamName!* where *TeamName* is the name of the team you input.

Input

A string representing a name of a team.

Output

The string *Welcome to Codewars 2025, TeamName!* where *TeamName* is the name of the team you input.

Example 1

Input

```
Trambolikos
```

Output

```
Welcome to Codewars 2025, Trambolikos!
```

Example 2

Input

```
Los top del Mundo
```

Output

```
Welcome to Codewars 2025, Los top del Mundo!
```

Solutions

Python

```
def main():
    team = input()
    print("Welcome to Codewars 2025, " + team + "!")
if __name__ == "__main__":
    main()
```

C++

```
#include <iostream>
#include <string>
int main(void) {
    std::string team;
    std::getline(std::cin, team);
    std::cout << "Welcome to Codewars 2025, " << team << "!" << std::endl;
    return 0;
}
```

Java

```
import java.util.*;
class Welcome2 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String teamName = scanner.nextLine().trim();
        System.out.println("Welcome to Codewars 2025, " + teamName+"!");
    }
}
```

3 Back to the future

3 points

Introduction

In a quiet workshop in Hill Valley, California, an eccentric scientist named Dr. Emmett Quantum invented the Year Shifter—a compact, wristwatch-like device capable of sending its wearer to any year in history or the future. The device was powered by a futuristic microfusion core and a precise algorithm to manipulate temporal dimensions.

One day, Dr. Quantum's assistant, Alex, accidentally activated the Year Shifter and got transported to an unknown time period. Alex's only clue to return home was an old note left by Dr. Quantum:

> *"To navigate through time, input the current year and the number of years you plan to travel. The device will take you there, but beware: The timeline will never forgive miscalculations."*

Unfortunately, the device's interface wasn't intuitive—it only displayed numbers, and Alex had to figure out where they had landed and how to return home. To avoid getting stuck, Alex built a simple calculation program on a nearby console to help understand the shifts in time. This program became The Year Shifter Calculator.

Alex used the program to make sense of the journey, determining:

- When they had landed (past, present, or future relative to the original timeline).
- How far they needed to travel to get back to safety.

But Alex soon discovered a complication: each shift not only moved them through time but also subtly altered historical events. For every jump, Alex had to decide whether to fix history or leave it unchanged.

Exercise

Create a program to calculate where Alex is going to travel

Input

Input consist in two lines with two different integer numbers:

- The current year.
- The number of years to travel (positive for future, negative for the past).

Output

The program will output one single line of text with the following structure:

"You will arrive in the year 3XXX, which is in the YYYY relative to ZZZZ."

Where:

- XXXX is the destination year.
- YYY is whether the year is in the past or the future relative to the present (texts "past" or "future").
- ZZZZ is the current year.

Example 1

Input

```
2024
-16
```

Output

```
You will arrive in the year 2008, which is in the past relative to 2024.
```

Example 2

Input

```
2004
50
```

Output

```
You will arrive in the year 2074, which is in the future relative to 2024.
```

Solutions

Python

```
# User inputs
# currentYear = int(input("Enter the current year: "))
currentYear = int(input())
# travelYears = int(input("Enter the number of years to travel (negative for
the past, positive for the future): "))
travelYears = int(input())
# Calculate the destination year
destinationYear = currentYear + travelYears
# Determine the time period relative to the current year
if travelYears > 0:
    timePeriod = "future"
elif travelYears < 0:
    timePeriod = "past"
else:
    timePeriod = "present"
# Output the result
print(f"You will arrive in the year {destinationYear}, which is in the
{timePeriod} relative to {currentYear}.")
```

C++

```
#include <iostream>
#include <string>
```

```
int main() {
    // User inputs
    int currentYear, travelYears, destinationYear;

    // Prompt for the current year
    //std::cout << "Enter the current year: ";
    std::cin >> currentYear;

    // Prompt for years to travel (positive for forward, negative for
backward)
    //std::cout << "Enter the number of years to travel (negative for the
past, positive for the future): ";
    std::cin >> travelYears;

    // Calculate the destination year
    destinationYear = currentYear + travelYears;

    // Determine the time period relative to the current year
    std::string timePeriod;
    if (travelYears > 0) {
        timePeriod = "future";
    } else if (travelYears < 0) {
        timePeriod = "past";
    } else {
        timePeriod = "present";
    }

    // Output the result
    std::cout << "You will arrive in the year " << destinationYear << ",
which is in the "
              << timePeriod << " relative to " << currentYear << ".\n";

    return 0;
}
```

Java

```java
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        // Create a Scanner object for user input
        Scanner scanner = new Scanner(System.in);
        // User inputs
        int currentYear, travelYears, destinationYear;
        // Prompt for the current year
        // System.out.print("Enter the current year: ");
        currentYear = scanner.nextInt();
        // Prompt for years to travel (positive for forward, negative for
backward)
        // System.out.print("Enter the number of years to travel (negative
for the past, positive for the future): ");
        travelYears = scanner.nextInt();
        // Calculate the destination year
        destinationYear = currentYear + travelYears;
        // Determine the time period relative to the current year
        String timePeriod;
        if (travelYears > 0) {
            timePeriod = "future";
        } else if (travelYears < 0) {
            timePeriod = "past";
        } else {
            timePeriod = "present";
        }
        // Output the result
        System.out.println("You will arrive in the year " + destinationYear +
", which is in the "
                           + timePeriod + " relative to " + currentYear +
".");
    }
}
```

4 The Enchanted Quest (Part I)

3 points

Introduction

You are an apprentice wizard who has just embarked on a grand quest to retrieve the five legendary magical artifacts scattered throughout the Enchanted Forest. These artifacts are the key to defeating an ancient curse threatening the kingdom. To find them, you must solve five mystical challenges set by the Guardians of the Forest. Each challenge tests your intellect and mastery of the magical arts. The Guardians have told you that the puzzles are rooted in ancient algorithms, and only the cleverest wizards will succeed.

Challenge 1: The Sorting Spell of the Shifting Stones

The first artifact is hidden among the Shifting Stones, a labyrinth of enchanted rocks that constantly move. You must sort the stones by their magical energy to reveal the path forward. The stones' energies are represented by numbers, and only by arranging them in ascending order will the way to the artifact open.

Exercise

You must write a program that sorts an array of numbers, which represent the energy levels of the Shifting Stones. Sorting them will reveal the path forward.

Input

The program will receive a list of integer numbers, comma-separated inside brackets. The program will write them inside brackets as well. For instance:

[5, 1, 9, 3, 7]

The amount of numbers is indeterminate (but greater than 0).

Output

A single line with the list of numbers, comma-separated, inside brackets but ordered. For instance:

[1, 3, 5, 7, 9]

Example 1

Input

```
[10, 2, 5, 7, 1, 4]
```

Output

```
[1, 2, 4, 5, 7, 10]
```

Example 2

Input

```
[100, 99, 88, 76, 50]
```

Output

```
[50, 76, 88, 99, 100]
```

Solutions

Python

```
# Read the input list as a string
input_list = input("").strip()
# Remove the brackets and split the string into individual integers
input_list = input_list[1:-1].split(',')
# Convert the string elements to integers
int_list = [int(x.strip()) for x in input_list]
# Sort the list of integers
sorted_list = sorted(int_list)
# Convert the sorted list back to a string with brackets
sorted_list_str = '[' + ', '.join(map(str, sorted_list)) + ']'
print(sorted_list_str)
```

C++

```
#include <iostream>
#include <vector>
#include <sstream>
#include <algorithm>
int main() {
    std::string input_list;
    std::getline(std::cin, input_list);
    // Remove the brackets
    input_list = input_list.substr(1, input_list.size() - 2);
    // Split the string into individual integers
    std::vector<int> int_list;
    std::stringstream ss(input_list);
    std::string item;
    while (std::getline(ss, item, ',')) {
        int_list.push_back(std::stoi(item));
    }
    // Sort the list of integers
    std::sort(int_list.begin(), int_list.end());
    // Convert the sorted list back to a string with brackets
    std::cout << "[";
    for (size_t i = 0; i < int_list.size(); ++i) {
        std::cout << int_list[i];
        if (i < int_list.size() - 1) {
            std::cout << ", ";
        }
    }
    std::cout << "]" << std::endl;
    return 0;
}
```

Java

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String inputList = scanner.nextLine();
        // Remove the brackets
        inputList = inputList.substring(1, inputList.length() - 1);
        // Split the string into individual integers
        List<Integer> intList = new ArrayList<>();
        String[] items = inputList.split(",");
        for (String item : items) {
            intList.add(Integer.parseInt(item.trim()));
        }
        // Sort the list of integers
        Collections.sort(intList);
        // Convert the sorted list back to a string with brackets
        System.out.print("[");
        for (int i = 0; i < intList.size(); i++) {
            System.out.print(intList.get(i));
            if (i < intList.size() - 1) {
                System.out.print(", ");
            }
        }
        System.out.println("]");
    }
}
```

5 Pyramid

4 points

Introduction

In the depths of an ancient temple, Indiana Jones discovers an ancient mechanism: a numerical pyramid.

To unlock the treasure, Indiana must input a number between 1 and 9. The pyramid mechanism will create a structure where each row contains:

- Numbers counting up to the row number.
- A central symbol (∗) representing the artifact.
- Numbers counting down from the row number.
- Indiana must ensure the input is valid and carefully interpret the output pyramid to unlock the next step. If the input is invalid, the mechanism will reject it, and he’ll have to try again.

Exercise

Create an application that receives a number and creates a pyramid of the indicated number of steps.

Input

```
3
```

Output

```
  1*1
 12*21
123*321
```

Example 1

Input

```
5
```

Output

```
    1*1
```

```
   12*21
  123*321
 1234*4321
12345*54321
```

Example 2

Input

```
11
```

Output

```
Invalid input! Please enter a number between 1 and 9.
```

Solutions

Python

```
import collections
levels = int(input())
if 1 <= levels <= 9:
    # Construir la pirámide
    for i in range(1, levels + 1):
        # Imprimir espacios para centrar la pirámide
        print(" " * (levels - i), end="")
        # Imprimir números ascendentes hasta la mitad
        for num in range(1, i + 1):
            print(num, end="")
        # Imprimir el carácter especial en el centro
        print("*", end="")
        # Imprimir números descendentes desde la mitad
        for num in range(i, 0, -1):
            print(num, end="")
        # Pasar a la siguiente línea
        print()
else:
    print("Invalid input! Please enter a number between 1 and 9.")
```

C++

```
#include <iostream>
using namespace std;
int main() {
    int levels;
    // Solicitar al usuario un número válido
    cin >> levels;
    if (levels >= 1 && levels <= 9) {
        // Construir la pirámide
        for (int i = 1; i <= levels; i++) {
            // Imprimir espacios para centrar la pirámide
            for (int j = 1; j <= levels - i; j++) {
                cout << " ";
            }
            // Imprimir números ascendentes hasta la mitad
            for (int num = 1; num <= i; num++) {
                cout << num;
            }
            // Imprimir el carácter especial en el centro
            cout << "*";
            // Imprimir números descendentes desde la mitad
            for (int num = i; num >= 1; num--) {
                cout << num;
            }
            // Pasar a la siguiente línea
            cout << endl;
        }
    } else {
        cout << "Invalid input! Please enter a number between 1 and 9." <<
endl;
    }
    return 0;
}
```

Java

```
import java.util.Scanner;
class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int levels = scanner.nextInt();
        if (levels >= 1 && levels <= 9) {
            // Build the pyramid
            for (int i = 1; i <= levels; i++) {
                // Print spaces to center the pyramid
                for (int j = 0; j < levels - i; j++) {
                    System.out.print(" ");
                }
                // Print ascending numbers up to the middle
                for (int num = 1; num <= i; num++) {
                    System.out.print(num);
                }
                // Print the special character in the center
                System.out.print("*");
                // Print descending numbers from the middle
                for (int num = i; num > 0; num--) {
                    System.out.print(num);
                }
                // Move to the next line
                System.out.println();
            }
        } else {
            System.out.println("Invalid input! Please enter a number between
1 and 9.");
        }
        scanner.close();
    }
}
```

6 Palindroms

5 points

Introduction

A palindrome is a word or phrase whose letters are arranged in such a way that they read the same from left to right as from right to left.

Write a program that determines whether a text string is a palindrome. The program must meet the following conditions:

- The input must contain only alphabetic letters and spaces (a-z, A-Z). In other cases, the input will be considered invalid.
- Upper and lower case letters must not affect the evaluation.
- Blanks must not be considered in the evaluation.

The output must be one of the following messages:

- The string is a palindrome
- The string is not a palindrome
- The input is not valid

Exercise

Write a program that receives a text as input and checks it as a palindrome.

Input

The input will consist in a text sentence.

Output

The Output can be:

"The string is a palindrome"

or

"The string is not a palindrome"

Example 1

Input

```
dabale arroz a la zorra el abad
```

Output

```
The string is a palindrome
```

Example 2

Input

```
aaabbbcccddd
```

Output

```
The string is not a palindrome
```

Solutions

Python

```
def is_palindrome(text):
    # Check if the input contains only alphabetic characters and spaces
    if not all(char.isalpha() or char.isspace() for char in text):
        return "The input is not valid"

    # Remove non-alphabetic characters and convert to lowercase
    cleaned_text = ''.join(filter(str.isalpha, text)).lower()

    # Check if the cleaned text is empty
    if not cleaned_text:
        return "The input is not valid"

    # Check if the cleaned text is a palindrome
    if cleaned_text == cleaned_text[::-1]:
        return "The string is a palindrome"
    else:
        return "The string is not a palindrome"
def main():
    # Get input from the user
    input_text = input("")
    # Check the response
    result = is_palindrome(input_text)
    # Print the result
    print(result)
if __name__ == "__main__":
    main()
```

C++

```
#include <iostream>
#include <algorithm>
#include <cctype>
#include <string>
using namespace std;
string isPalindrome(const string& text) {
    // Check if the input contains only alphabetic characters and spaces
    for (char c : text) {
        if (!isalpha(c) && !isspace(c)) {
            return "The input is not valid";
        }
    }
    // Remove non-alphabetic characters and convert to lowercase
    string cleanedText;
    for (char c : text) {
        if (isalpha(c)) {
            cleanedText += tolower(c);
        }
    }
    // Check if the cleaned text is empty
    if (cleanedText.empty()) {
        return "The input is not valid";
    }
    // Check if the cleaned text is a palindrome
    string reversedText = cleanedText;
    reverse(reversedText.begin(), reversedText.end());
    if (cleanedText == reversedText) {
        return "The string is a palindrome";
    } else {
        return "The string is not a palindrome";
    }
}
int main() {
    // Get input from the user
    string inputText;
    getline(cin, inputText);
    // Check if the input text is a palindrome and print the result
    string result = isPalindrome(inputText);
    cout << result << endl;
    return 0;
}
```

Java

```
import java.util.Scanner;
class Main {

    public static String isPalindrome(String text) {
        // Check if the input contains only alphabetic characters and spaces
        if (!text.matches("[a-zA-Z ]+")) {
            return "The input is not valid";
        }
        // Remove non-alphabetic characters and convert to lowercase
        String cleanedText = text.replaceAll("[^a-zA-Z]", "").toLowerCase();
        // Check if the cleaned text is empty
        if (cleanedText.isEmpty()) {
            return "The input is not valid";
        }
        // Check if the cleaned text is a palindrome
        String reversedText = new
StringBuilder(cleanedText).reverse().toString();
        if (cleanedText.equals(reversedText)) {
            return "The string is a palindrome";
        } else {
            return "The string is not a palindrome";
        }
    }

    public static void main(String[] args) {
        // Get input from the user
        Scanner scanner = new Scanner(System.in);
        String inputText = scanner.nextLine();
        scanner.close();

        // Check if the input text is a palindrome and print the result
        String result = isPalindrome(inputText);
        System.out.println(result);
    }
}
```

7 The Jedi Formula

5 points

Introduction

Thanks to the Force, the Jedi Knights have discovered the formula for identifying the probability of defeat against an Empire ship. It turns out to be a simple mathematical formula that depends on the reflexes of each 7-Wing pilot. The Jedi Knights are able to analyze these reflexes and by applying them to a computer understand the chance of victory against an Empire ship.

Exercise

You are asked to write a program that calculates this defeat probability by reading the following data as input lines (all will be positive integers between 0 and 100) and provides as output a single positive integer as follows:

- First line, the probability in percent that an anti-aircraft tower or enemy ship shot will destroy an X-Wing and its pilot (say 10. That is, 10%, only 1 out of 10 enemy shots hit an X-Wing).
- Second line, the probability that an X-Wing pilot, if still alive after hitting the target, will hit a strategic point of the Empire's ship with his shot (i.e. 40 that means 40%).
- In the third line, the average number of shots the pilots receive when trying to reach the target (positive number >= 0)
- The output of the program will be the probability of success of a single pilot aboard his dodgy X-Wing.

Let's go there with a small guide to solve this:

The probability that a pilot with his X-Wing arrives alive and operational until he can make his shot is the product of the success of each anti-aircraft shot.

Since we have N shots on average (third line of entry), we will have N-1 products to perform to calculate the probability of being alive before reaching the imperial ship. Note that the probability provided is the probability of success **of the enemy** ;).

For instance if the success probability of the anti-aircraft is 10%, and 5 shots are shot by the cannons, the success probability would be $0.9 * 0.9 * 0.9 * 0.9 * 0.9$.

If the X-Wing reaches its destination it will have to fire and hit (probability obtained from the second line of the program). Again the probability of destroying the death star is the product of the probabilities of hitting the shot and being alive after the path of turrets and enemy ships. With this we have calculated the probability of a single X-Wing destroying the precious and unloved planet-destroying space station.

Let's take a look at some examples so you can check your results:

Anti-Aircraft Probability: 10%

X-Wing probability: 40%

Number of shots received: 5

Final success probability: 23.62%

Because $0.9 * 0.9 \ * 0.9 * 0.9 \ * 0.9 * 0.4 = 0.2362$

Another example:

Anti-Aircraft Probability: 10%

X-Wing probability: 40%

Number of shots received: 20

Final success probability: 4.86%

Because $0.9 *$ .... 20 times... $* 0.9 * 0.4 = 0.0486$

Input

The program will receive three input lines with the Anti-Aircraft Probability (in percentage), the X-Wing probability (in percetage) and the Number of shots received (integer).

Output

A numeric value with 2 decimals from 0.00 to 99.99.

Example 1

Input

```
10
40
5
```

Output

```
23.62
```

Example 2

Input

```
10
40
20
```

Output

```
4.86
```

Example 3

Input

```
50
40
10
```

Output

```
0.04
```

Solutions

Python

```
def calculate_success_probability(anti_aircraft_prob, xwing_hit_prob,
num_shots):
    # Convert percentages to probabilities
    anti_aircraft_prob /= 100
    xwing_hit_prob /= 100
    # Calculate the probability of surviving all shots
    survival_prob = (1 - anti_aircraft_prob) ** num_shots
    # Calculate the overall success probability
    success_prob = survival_prob * xwing_hit_prob
    # Convert the probability back to a percentage
    success_prob_percentage = success_prob * 100
    return success_prob_percentage
```

```
def main():
    # Read input values
    anti_aircraft_prob = int(input())
    xwing_hit_prob = int(input())
    num_shots = int(input())
    # Calculate the success probability
    success_prob_percentage =
calculate_success_probability(anti_aircraft_prob, xwing_hit_prob, num_shots)
    # Print the result
    print(f"{success_prob_percentage:.2f}")
if __name__ == "__main__":
    main()
```

C++

```
#include <iostream>
#include <iomanip>
#include <cmath>
double calculate_success_probability(double anti_aircraft_prob, double
xwing_hit_prob, int num_shots) {
    // Convert percentages to probabilities
    anti_aircraft_prob /= 100;
    xwing_hit_prob /= 100;
    // Calculate the probability of surviving all shots
    double survival_prob = std::pow(1 - anti_aircraft_prob, num_shots);
    // Calculate the overall success probability
    double success_prob = survival_prob * xwing_hit_prob;
    // Convert the probability back to a percentage
    double success_prob_percentage = success_prob * 100;
    return success_prob_percentage;
}
int main() {
    // Read input values
    int anti_aircraft_prob, xwing_hit_prob, num_shots;
    std::cin >> anti_aircraft_prob >> xwing_hit_prob >> num_shots;
    // Calculate the success probability
    double success_prob_percentage =
calculate_success_probability(anti_aircraft_prob, xwing_hit_prob, num_shots);
    // Print the result
    std::cout << std::fixed << std::setprecision(2) <<
success_prob_percentage << std::endl;
    return 0;
}
```

Java

```
import java.util.Scanner;
import java.util.Locale;
public class Main {
    public static double calculateSuccessProbability(double antiAircraftProb,
double xwingHitProb, int numShots) {
        // Convert percentages to probabilities
        antiAircraftProb /= 100;
        xwingHitProb /= 100;
        // Calculate the probability of surviving all shots
        double survivalProb = Math.pow(1 - antiAircraftProb, numShots);
        // Calculate the overall success probability
        double successProb = survivalProb * xwingHitProb;
        // Convert the probability back to a percentage
        double successProbPercentage = successProb * 100;
        return successProbPercentage;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Locale.setDefault(Locale.US);

        // Read input values
        int antiAircraftProb = scanner.nextInt();
        int xwingHitProb = scanner.nextInt();
        int numShots = scanner.nextInt();
        // Calculate the success probability
        double successProbPercentage =
calculateSuccessProbability(antiAircraftProb, xwingHitProb, numShots);
        // Print the result
        System.out.printf("%.2f%n", successProbPercentage);
    }
}
```

8 The cipher of life

6 points

Introduction

The Cipher of life

In the darkest days of World War II, when the Axis powers sought to dominate the world, the Allied forces knew they had to stay one step ahead. The art of encryption became their shield, protecting their plans and strategies from the prying eyes of the enemy. Among their secret operations, one unique encryption system stood out—a code inspired by the building blocks of life itself: DNA.

The Birth of the BioCipher

Dr. Eleanor Hayes, a brilliant British biochemist and cryptographer, had spent years studying genetics. As the war escalated, she was recruited into the Allied intelligence division, Ultra, at Bletchley Park. Tasked with developing unbreakable codes, Eleanor had an epiphany: "Why not use the language of nature itself?"

Eleanor designed an encoding scheme where every letter of the English alphabet and common symbols were translated into binary. Then, using the rules of molecular biology, she assigned A, T, C, and G to binary pairs:

- 00 = A
- 01 = T
- 10 = C
- 11 = G

With this method, any message could be transformed into a sequence of what appeared to be random DNA bases. It was elegant, deceptive, and completely alien to traditional cryptography. The Nazis, masters of breaking mechanical ciphers like the Enigma, would have no idea they were reading simulated genetic codes.

Operation Genesis

The Allies implemented Eleanor's BioCipher in Operation Genesis, a mission to coordinate troop movements across occupied Europe. Messages were encoded into DNA sequences and disguised as scientific research papers. These documents would be hand-delivered by couriers posing as geneticists or medical researchers, a perfect cover in an era when biology was rapidly advancing.

For example, a message like:

```
Troops advance at dawn
```

would be encoded as:

```
01010100 01110010 01101111 01101111 01110000 01110011 00100000 01100001
01100100 01110110 01100001 01101110 01100011 01100101 00100000 01100001
01110100 00100000 01100100 01100001 01110111 01101110
```

and translated into:

```
TACTGCAATTGCCGATC...
```

It looked like nothing more than gibberish to anyone without the key.

The Nazi Interception

One chilling night in 1944, a courier carrying a BioCipher-encoded message was captured in Nazi-occupied France. The Nazis, suspecting he was a spy, sent the document to their cryptographers in Berlin. For weeks, the Germans poured over the DNA-like strings, convinced they had uncovered a critical Allied cipher. They tried substitutions, permutations, and mechanical decodings, but the sequences resisted every attempt.

Frustrated, they turned to their scientists. The German biologist Dr. Klaus Reinhardt examined the sequences and declared them a plausible genetic experiment. Believing the message to be unrelated to the war, the Nazis abandoned their efforts and returned to their existing codes.

Unbeknownst to them, the message had already reached the Allied command, leading to the liberation of a critical stronghold in northern France.

Legacy of the BioCipher

Eleanor Hayes's BioCipher remained one of the war’s best-kept secrets, only declassified decades later. Her ingenious use of biology to confound the Nazis became a cornerstone of modern cryptographic history. While never a large-scale cipher due to its complexity, it was a testament to the brilliance and creativity of Allied intelligence.

Today, Eleanor's work is celebrated as a reminder of how innovation and science can serve as powerful tools, even in the most desperate of times. The BioCipher may have been rooted in war, but it also symbolized humanity's resilience and ingenuity, drawing inspiration from the fundamental code of life.

Exercise

Write a program that receives as input the string to be encoded and returns the encoded script as a ATCG DNA string.

Input

```
Enemies on the door
```

Output

```
TATTTCGCTCTTTCGTTCCTTCTTTGAGACAATCGGTCGCACAATGTATCCATCTTACAATCTATCGGTCGGTGAC
```

Example 1

Input

```
Attack from the north, retreat to east
```

Output

```
TAATTGTATGTATCATTCAGTCCGACAATCTCTGACTCGGTCGTACAATGTATCCATCTTACAATCGCTCGGTGACT
GTATCCAACGAACAATGACTCTTTGTATGACTCTTTCATTGTAACAATGTATCGGACAATCTTTCATTGAGTGTA
```

Solutions

Python

```
# Mapa para codificar binarios a bases de ADN
binary_to_dna = {
    "00": 'A',
    "01": 'T',
    "10": 'C',
    "11": 'G'
}
# Función para convertir un carácter ASCII en binario (8 bits)
def char_to_binary(c):
    return format(ord(c), '08b')
# Función para convertir una cadena binaria a ADN
def binary_to_dna_sequence(binary):
    dna_sequence = ""
    # Recorrer el binario en grupos de 2 bits
    for i in range(0, len(binary), 2):
        bits = binary[i:i+2]
        if bits in binary_to_dna:
            dna_sequence += binary_to_dna[bits]
        else:
            print(f"Error: Secuencia de bits no válida: {bits}")
            return ""
    return dna_sequence
def main():
    input_text = input("")
    # Convertir cada carácter ASCII a binario
    binary_sequence = "".join(char_to_binary(c) for c in input_text)
    # Convertir el binario a la secuencia de ADN
    dna_sequence = binary_to_dna_sequence(binary_sequence)
    if dna_sequence:
        print(dna_sequence)
if __name__ == "__main__":
    main()
```

C++

```
#include <iostream>
#include <bitset>
#include <string>
#include <unordered_map>
using namespace std;
// Mapa para codificar binarios a bases de ADN
```

```
unordered_map<string, char> binaryToDNA = {
    {"00", 'A'},
    {"01", 'T'},
    {"10", 'C'},
    {"11", 'G'}
};
// Función para convertir un carácter ASCII en binario (8 bits)
string charToBinary(char c) {
    return bitset<8>(c).to_string();
}
// Función para convertir una cadena binaria a ADN
string binaryToDNASequence(const string& binary) {
    string dnaSequence = "";
    // Recorrer el binario en grupos de 2 bits
    for (size_t i = 0; i < binary.size(); i += 2) {
        string bits = binary.substr(i, 2); // Extraer 2 bits
        if (binaryToDNA.find(bits) != binaryToDNA.end()) {
            dnaSequence += binaryToDNA[bits];
        } else {
            cerr << "Error: Secuencia de bits no válida: " << bits << endl;
            return "";
        }
    }
    return dnaSequence;
}
int main() {
    string input;
    getline(cin, input);
    // Convertir cada carácter ASCII a binario
    string binarySequence = "";
    for (char c : input) {
        binarySequence += charToBinary(c);
    }
    // Convertir el binario a la secuencia de ADN
    string dnaSequence = binaryToDNASequence(binarySequence);
    if (!dnaSequence.empty()) {
        cout << dnaSequence << endl;
    }
    return 0;
}
```

Java

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
public class cipher {
    // Mapa para codificar binarios a bases de ADN
    private static final Map<String, Character> binaryToDNA = new
HashMap<>();
    static {
        binaryToDNA.put("00", 'A');
        binaryToDNA.put("01", 'T');
        binaryToDNA.put("10", 'C');
        binaryToDNA.put("11", 'G');
    }
    // Función para convertir un carácter ASCII en binario (8 bits)
    public static String charToBinary(char c) {
        return String.format("%8s", Integer.toBinaryString(c)).replace(' ',
'0');
    }
    // Función para convertir una cadena binaria a ADN
    public static String binaryToDNASequence(String binary) {
        StringBuilder dnaSequence = new StringBuilder();
        // Recorrer el binario en grupos de 2 bits
        for (int i = 0; i < binary.length(); i += 2) {
            String bits = binary.substring(i, Math.min(i + 2,
binary.length()));
            if (binaryToDNA.containsKey(bits)) {
                dnaSequence.append(binaryToDNA.get(bits));
            } else {
                System.err.println("Error: Secuencia de bits no válida: " +
bits);
                return "";
            }
        }
        return dnaSequence.toString();
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();
        // Convertir cada carácter ASCII a binario
        StringBuilder binarySequence = new StringBuilder();
        for (char c : input.toCharArray()) {
            binarySequence.append(charToBinary(c));
        }
        // Convertir el binario a la secuencia de ADN
        String dnaSequence = binaryToDNASequence(binarySequence.toString());
        if (!dnaSequence.isEmpty()) {
            System.out.println(dnaSequence);
        }
        scanner.close();
    }
}
```

9 The Magic Cloud

6 points

Introduction

After the cataclysmic war where the noble empires of Elvenkind and Dwarvenkind collided in a tempest of steel and sorcery, the battlefield, now a desolate wasteland known only as "Knowhere," became the cradle of legend. The echoes of clashing blades and the cries of ancient warriors have long faded, but their story endures.

To ensure that this epic tale of valor and betrayal is never lost to the sands of time, the enigmatic and cunning Snake Scripture was invoked—a mystical artifact capable of weaving a cloud of enchanted words. Through this ethereal tapestry, all who seek the truth may behold the saga unfold before their very eyes, as though gazing into the currents of time itself.

You, a mage of esteemed renown, have been summoned to aid in crafting this cloud of inherited wisdom. Your task is to harness the essence of the tale, distilling its most pivotal moments and shaping them into a constellation of words. Each word, infused with magic, will rise in prominence and size if it holds the weight of the story's heart.

Step forth into this sacred duty, conjurer of knowledge, and ensure that the legacy of the Last War shines bright in the minds of generations to come.

Exercise

Write a program that with a input text, counts the number that each word is repeated and shows an ordered list of words that are repeated 3 or more times. Note that before counting word, all symbols should be removed , except the apostrophes

Input

```
dragon spell adventure treasure warrior mage castle stars sword knight legend
kingdom battle sorcerer adventure dragon magic creature spell travel journey
epic myth guardian mystic treasure alchemy spell dragon wisdom legends fog
magic elixir runes treasure castle prophecy
```

Output

```
dragon dragon dragon
spell spell spell
treasure treasure treasure
```

Example 1

Input

```
dragon spell adventure treasure warrior mage castle stars sword knight legend
kingdom battle sorcerer adventure dragon magic creature spell travel journey
epic myth guardian mystic treasure alchemy spell dragon wisdom legends fog
magic elixir runes treasure castle prophecy
```

Output

```
dragon dragon dragon
spell spell spell
treasure treasure treasure
```

Example 2

Input

```
Everyone my age remembers where they were and what they were doing when they
first heard about the contest. I was sitting in my hideout watching cartoons
when the news bulletin broke in on my video feed, announcing that James
Halliday had died during the night. I'd heard of Halliday, of course.
Everyone had. He was the videogame designer responsible for creating the
OASIS, a massively multiplayer online game that had gradually evolved into
the globally networked virtual reality most of humanity now used on a daily
basis. The unprecedented success of the OASIS had made Halliday one of the
wealthiest people in the world. At first, I couldn't understand why the media
was making such a big deal of the billionaire's death. After all, the people
of Planet Earth had other concerns. The ongoing energy crisis. Catastrophic
climate change. Widespread famine, poverty, and disease. Half a dozen wars.
You know: "dogs and cats living together … mass hysteria!" Normally, the
newsfeeds didn't interrupt everyone's interactive sitcoms and soap operas
unless something really major had happened. Like the outbreak of some new
killer virus, or another major city vanishing in a mushroom cloud. Big stuff
like that. As famous as he was, Halliday's death should have warranted only a
brief segment on the evening news, so the unwashed masses could shake their
heads in envy when the newscasters announced the obscenely large amount of
money that would be doled out to the rich man's heirs.
```

Output

```
a a a a a a
and and and and
had had had had had had
halliday halliday halliday
in in in in in
my my my
of of of of of of of of of
on on on
that that that that
the the the the the the the the the the the the the the the the the the the
the the
they they they
was was was was
when when when
```

Solutions

Python

```
import re
from collections import Counter
def main():
    text = input("")
    # Dividir el texto en palabras usando expresiones regulares
    words = re.findall(r'\b[áéíóú1-9A-z’\']+\b', text.lower())
    # Contar las palabras
    word_count = Counter(words)
    # Filtrar palabras que aparecen 3 o más veces
    filtered_words = {word: count for word, count in word_count.items() if
count >= 3}
    # Ordenar alfabéticamente
    sorted_words = sorted(filtered_words.keys())
    # Imprimir las palabras en el formato solicitado
    for word in sorted_words:
        print(" ".join([word] * filtered_words[word])+" ")
if __name__ == "__main__":
    main()
```

C++

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#define MAX_WORDS 1000
#define MAX_LENGTH 100
typedef struct {
    char word[MAX_LENGTH];
    int count;
} Word;
void toLowerCase(char *str) {
    for (int i = 0; str[i]; i++) {
        str[i] = tolower((unsigned char)str[i]);
    }
}
int findWord(Word words[], int size, char *word) {
    for (int i = 0; i < size; i++) {
        if (strcmp(words[i].word, word) == 0) {
            return i;
        }
    }
    return -1;
}
int compareWords(const void *a, const void *b) {
    Word *wordA = (Word *)a;
    Word *wordB = (Word *)b;
    return strcmp(wordA->word, wordB->word);
}
void countWords(char *text) {
    Word words[MAX_WORDS];
    int wordCount = 0;
    char *token = strtok(text, " .,;:!?()\n");
    while (token != NULL) {
        toLowerCase(token);
        int index = findWord(words, wordCount, token);
        if (index == -1) {
            // Nueva palabra
            strcpy(words[wordCount].word, token);
            words[wordCount].count = 1;
            wordCount++;
        } else {
            // Palabra ya existente
            words[index].count++;
        }
        token = strtok(NULL, " .,;:!?()\n");
    }
    // Ordenar palabras alfabéticamente
    qsort(words, wordCount, sizeof(Word), compareWords);
    for (int i = 0; i < wordCount; i++) {
        if (words[i].count >= 3) {
            for (int j = 0; j < words[i].count; j++) {
                printf("%s ", words[i].word);
            }
            printf("\n");
        }
    }
}
int main() {
    char text[10000];
    fgets(text, sizeof(text), stdin);
    countWords(text);
    return 0;
}
```

Java

```
import java.util.*;
public class magicCloud {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String text = scanner.nextLine();
        // Dividir el texto en palabras usando delimitadores comunes
        String[] words = text.toLowerCase().split("[\\s.,;:!?()]+");

        // Mapa para contar las palabras
        Map<String, Integer> wordCount = new HashMap<>();
        for (String word : words) {
            if (!word.isEmpty()) {
                wordCount.put(word, wordCount.getOrDefault(word, 0) + 1);
            }
        }
        // Filtrar palabras que aparecen 3 o más veces y ordenarlas
alfabéticamente
        List<String> filteredWords = new ArrayList<>();
        for (Map.Entry<String, Integer> entry : wordCount.entrySet()) {
            if (entry.getValue() >= 3) {
                filteredWords.add(entry.getKey());
            }
        }
        Collections.sort(filteredWords);
        // Imprimir las palabras en el formato solicitado
        for (String word : filteredWords) {
            for (int i = 0; i < wordCount.get(word); i++) {
                System.out.print(word + " ");
            }
            System.out.println();
        }
        scanner.close();
    }
    }
```

10 Building a calculator I

7 points

Introduction

In your school they have asked you to write a program that simulate a simple calculator.

The program will simple receive a text with two numbers, a operation symbol and a equals symbol at the end, and your program will return the result of the operation.

Exercise

Write a program that reads a line like:

13 + 20 =

And return the solution:

33

The posible operations are +, -, ∗, / and % (module)

Note: The calculator will not consider divisions by zero (They won't be tested).

Input

A single line with two numbers and a operation symbold, finished by an equals symbol.

Note: There will always be a space between the numbers and the operations, and the line will end with a "="

Output

A single number with the solution.

Note: Divisions will only show (if necessary) 2 decimal digits. If the result is 3.20, it will show 3.2.

Note: The decimal separator is the dot (.)

Example 1

Input

```
99 % 12 =
```

Output

```
3
```

Example 2

Input

```
15 / 5 =
```

Output

```
3
```

Example 3

Input

```
16 / 3 =
```

Output

```
5.33
```

Solutions

Python

```
def calculate(a, operator, b):
    if operator == '+':
        return a + b
    elif operator == '-':
        return a - b
    elif operator == '*':
        return a * b
    elif operator == '/':
        if b != 0:
            if ((a / b) %1==0):
                return int(a/b)
            else:
                return round(a / b, 2)
        else:
            raise ValueError("Division by zero")
    elif operator == '%':
        if b != 0:
            return a % b
        else:
            raise ValueError("Division by zero")
    else:
        raise ValueError("Invalid operator")
def main():
    input_line = input("").strip()
    # Remove the '=' sign and split the input into parts
    input_line = input_line.replace("=", "").strip()
    parts = input_line.split()
    if len(parts) != 3:
        print("Invalid input format")
        return
    try:
        a = int(parts[0])
        operator = parts[1]
        b = int(parts[2])
        result = calculate(a, operator, b)
        print(result)
    except ValueError as e:
        print(e)
if __name__ == "__main__":
    main()
```

C++

```
#include <iostream>
#include <sstream>
#include <vector>
#include <iomanip>
#include <stdexcept>
#include <algorithm>  // For std::remove
#include <cmath>
double calculate(int a, const std::string& operator_, int b) {
    if (operator_ == "+") {
        return a + b;
    } else if (operator_ == "-") {
        return a - b;
    } else if (operator_ == "*") {
        return a * b;
    } else if (operator_ == "/") {
        if (b != 0) {
            return static_cast<double>(a) / b;
        } else {
            throw std::invalid_argument("Division by zero");
        }
    } else if (operator_ == "%") {
        if (b != 0) {
            return a % b;
        } else {
            throw std::invalid_argument("Division by zero");
        }
    } else {
        throw std::invalid_argument("Invalid operator");
    }
}
std::vector<std::string> split(const std::string& str, char delimiter) {
    std::vector<std::string> tokens;
    std::string token;
    std::istringstream tokenStream(str);
    while (std::getline(tokenStream, token, delimiter)) {
        tokens.push_back(token);
    }
    return tokens;
}
int main() {
    std::string input;
    std::getline(std::cin, input);
    // Remove the '=' sign using std::remove and erase
    input.erase(std::remove(input.begin(), input.end(), '='), input.end());
    // Trim leading and trailing spaces
    input = input.substr(0, input.find_last_not_of(" \t\n\r\f\v") + 1);
    std::vector<std::string> parts = split(input, ' ');
    if (parts.size() != 3) {
        std::cout << "Invalid input format" << std::endl;
        return 1;
    }
    try {
        int a = std::stoi(parts[0]);
        std::string operator_ = parts[1];
        int b = std::stoi(parts[2]);
        double result = calculate(a, operator_, b);
        if (operator_ == "/") {
            if (std::floor(result) == result) {
                std::cout << static_cast<int>(result) << std::endl;  // Print
as integer
            } else if (std::floor(result*10) == (result*10)) {
                std::cout << std::fixed << std::setprecision(1) << result <<
std::endl;  // Print with 1 decimal place
            }
            else {
                std::cout << std::fixed << std::setprecision(2) << result <<
std::endl;  // Print with 2 decimal place
            }
        } else {
            std::cout << static_cast<int>(result) << std::endl;  // Print as
integer for other operations
        }
    } catch (const std::invalid_argument& e) {
        std::cout << e.what() << std::endl;
    }
    return 0;
}
```

Java

```java
import java.util.*;
public class calculatorI {
    public static double calculate(int a, String operator_, int b) {
        switch (operator_) {
            case "+":
                return a + b;
            case "-":
                return a - b;
            case "*":
                return a * b;
            case "/":
                if (b != 0) {
                    return (double) a / b;
                } else {
                    throw new IllegalArgumentException("Division by zero");
                }
            case "%":
                if (b != 0) {
                    return a % b;
                } else {
                    throw new IllegalArgumentException("Division by zero");
                }
            default:
                throw new IllegalArgumentException("Invalid operator");
        }
    }
    public static List<String> split(String str, char delimiter) {
        List<String> tokens = new ArrayList<>();
        String[] parts = str.split(String.valueOf(delimiter));
        Collections.addAll(tokens, parts);
        return tokens;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();
        // Remove the '=' sign and trim trailing whitespace
        input = input.replace("=", "").trim();
        // Split the input into parts
        List<String> parts = split(input, ' ');
        if (parts.size() != 3) {
            System.out.println("Invalid input format");
            return;
        }
        try {
            int a = Integer.parseInt(parts.get(0));
            String operator_ = parts.get(1);
            int b = Integer.parseInt(parts.get(2));
            double result = calculate(a, operator_, b);
            if ("/".equals(operator_)) {
                if (result % 1 == 0) {
                    System.out.println((int) result);
                } else {
                    // Format result with up to 2 decimals
                    String formatted = String.format(Locale.US,"%.2f",
result)
                            .replaceAll("\\.0+$|(?<=\\.\\d)0+$", ""); //
Remove trailing zeros after decimal
                    System.out.println(formatted);
                }
            } else {
                System.out.println((int) result);
            }
        } catch (NumberFormatException e) {
            System.out.println("Invalid number format");
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }
    }
    }
```

11 Mario and the mushroom jump

7 points

Introduction

Mario is on an adventure in the Mushroom Kingdom! His level has its own rules:

- The level is made up of a row of blocks, each with a number on it.
- This number tells Mario how far he can jump forward from that block.
- Mario starts at the first block (block 0).
- His goal is to reach the last block in as few jumps as possible.
- Mario can only jump forward, and he cannot jump past the last block.
- If Mario lands on a block with the number 0, he can't jump forward anymore and might get stuck.

Exercise

Mario has asked you to help him solve this problem. Can you do it?

Write a program that calculates the smallest number of jumps Mario needs to reach the last block.

Input

The program will take one input:

- A comma-separated list of numbers. Each number represents a block, starting from block 0.

Output

The program should print:

- The minimum number of jumps Mario needs to reach the last block, or
- *"It is impossible to reach the last block"* if Mario cannot finish the level.

Example 1

Input

```
1,0,5,2,0,9,1,2,3
```

Output

```
It is impossible to reach the last block
```

Explanation:

- Mario starts on block 0, which has 1. He jumps to block 1 (value 0).
- Since block 1 is 0, Mario cannot jump further.

Example 2

Input

```
2,3,1,1,4
```

Output

2

Explanation:

- Mario starts on block 0 (value 2). He jumps to block 1 (value 3).
- From block 1, he jumps to block 4 (last block).
- Mario could have jumped 2 blocks instead of 1 at the beginning, but it would take more jumps to reach the last block.

Solutions

Python

```
def jump(nums):
    if len(nums) <= 1:
        return 0  # No jumps needed if the array has 1 or fewer elements

    jumps = 0  # Number of jumps made
    current_end = 0  # The farthest we can reach with the current jump
    farthest = 0  # The farthest we can reach in the next jump
    for i in range(len(nums) - 1):  # We don't need to check the last element
        farthest = max(farthest, i + nums[i])
        # If we have reached the end of the current jump range, make another
jump
        if i == current_end:
            jumps += 1
            current_end = farthest  # Update to the farthest we can reach
with this jump
            # If the farthest we can reach is at or beyond the last index,
we're done
            if current_end >= len(nums) - 1:
                return jumps
            # If at any point we are at a position with a value of 0 and
can't progress
            if nums[i] == 0:
                print("It is impossible to reach the last block")
                return  # Exit the function immediately
    return jumps
# Read input from standard input as a comma-separated list of numbers
if __name__ == "__main__":
    input_str = input("")
    nums = list(map(int, input_str.split(',')))

    result = jump(nums)
    if isinstance(result, int):  # If a valid number of jumps was returned
        print(result)
```

C++

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int jump(vector<int>& nums) {
    if (nums.size() <= 1) {
        return 0;  // No jumps needed if the array has 1 or fewer elements
    }
    int jumps = 0;  // Number of jumps made
    int current_end = 0;  // The farthest we can reach with the current jump
    int farthest = 0;  // The farthest we can reach in the next jump
    for (int i = 0; i < nums.size() - 1; ++i) {  // We don't need to check
the last element
        farthest = max(farthest, i + nums[i]);
        // If we have reached the end of the current jump range, make another
jump
        if (i == current_end) {
            jumps++;
            current_end = farthest;  // Update to the farthest we can reach
with this jump
            // If the farthest we can reach is at or beyond the last index,
we're done
            if (current_end >= nums.size() - 1) {
                return jumps;
            }
            // If at any point we are at a position with a value of 0 and
can't progress
            if (nums[i] == 0) {
                cout << "It is impossible to reach the last block" << endl;
                return -1;  // Return a special value to indicate failure
            }
        }
    }
    return jumps;
}
int main() {
    string input_str;
    getline(cin, input_str);  // Read the entire line from input
    // Convert the input string to a vector of integers
    vector<int> nums;
    size_t start = 0;
    size_t end = input_str.find(',');
    while (end != string::npos) {
        nums.push_back(stoi(input_str.substr(start, end - start)));  //
Convert to integer
        start = end + 1;
        end = input_str.find(',', start);
    }
    nums.push_back(stoi(input_str.substr(start)));  // Add the last number
    int result = jump(nums);
    if (result != -1) {  // If the result is valid, print the number of jumps
        cout << result << endl;
    }
    return 0;
}
```

Java

```
import java.util.*;
public class JumpGame {
    public static int jump(int[] nums) {
        if (nums.length <= 1) {
            return 0;  // No jumps needed if the array has 1 or fewer
elements
        }
        int jumps = 0;  // Number of jumps made
        int currentEnd = 0;  // The farthest we can reach with the current
jump
        int farthest = 0;  // The farthest we can reach in the next jump
        for (int i = 0; i < nums.length - 1; ++i) {  // We don't need to
check the last element
            farthest = Math.max(farthest, i + nums[i]);
            // If we have reached the end of the current jump range, make
another jump
            if (i == currentEnd) {
                jumps++;
                currentEnd = farthest;  // Update to the farthest we can
reach with this jump
```

```
                // If the farthest we can reach is at or beyond the last
index, we're done
                if (currentEnd >= nums.length - 1) {
                    return jumps;
                }
                // If at any point we are at a position with a value of 0 and
can't progress
                if (nums[i] == 0) {
                    System.out.println("It is impossible to reach the last
block");
                    return -1;  // Return a special value to indicate failure
                }
            }
        }
        return jumps;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Read input
        String inputStr = scanner.nextLine();
        // Convert the input string to an array of integers
        String[] inputArr = inputStr.split(",");
        int[] nums = new int[inputArr.length];
        for (int i = 0; i < inputArr.length; i++) {
            nums[i] = Integer.parseInt(inputArr[i].trim());
        }
        // Call the jump function
        int result = jump(nums);
        if (result != -1) {  // If the result is valid, print the number of
jumps
            System.out.println(result);
        }
        scanner.close();
    }
}
```

12 The enchanted quest (Part II)

7 points

Introduction

Challenge 2: The Fibonacci Talisman of Time

The second artifact, the Talisman of Time, is hidden within a series of mystical time loops. The key to unlocking the Talisman is a sequence of numbers known as the Fibonacci sequence. Only by calculating the first N numbers in the sequence can you unlock the portal to the artifact's resting place.

Exercise

You must write a program that computes the first N numbers of the Fibonacci sequence. The numbers in the sequence are calculated by adding the two previous numbers, starting from 0 and 1.

Exercise

Input

The input will be a single integer number, greater than 0, than will represent the number of positions of the fibonacci series that will be shown.

For instance:

```
5
```

Output

The program will shown the indicated number of elements of the fibonacci series starting from the first element (0). For instance, with input value 5, the output would be:

```
[0, 1, 1, 2, 3]
```

Example 1

Input

```
8
```

Output

```
[0, 1, 1, 2, 3, 5, 8, 13]
```

Example 2

Input

```
10
```

Output

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Example 3

Input

```
15
```

Output

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
```

Solutions

Python

```
def fibonacci(n):
    if n <= 0:
        return "Input must be greater than 0"

    fibonacci_series = []
    for i in range(n):
        if i == 0:
            fibonacci_series.append(0)
        elif i == 1:
            fibonacci_series.append(1)
        else:
            next_number = fibonacci_series[-1] + fibonacci_series[-2]
            fibonacci_series.append(next_number)

    return fibonacci_series
if __name__ == "__main__":
    n = int(input())
    result = fibonacci(n)
    print(result)
```

C++

```
#include <iostream>
#include <vector>
std::vector<int> fibonacci(int n) {
    std::vector<int> fibonacci_series;
    if (n <= 0) {
        std::cerr << "Input must be greater than 0" << std::endl;
        return fibonacci_series;
    }
    for (int i = 0; i < n; ++i) {
        if (i == 0) {
            fibonacci_series.push_back(0);
        } else if (i == 1) {
            fibonacci_series.push_back(1);
        } else {
            int next_number = fibonacci_series[i - 1] + fibonacci_series[i -
2];
            fibonacci_series.push_back(next_number);
        }
    }
    return fibonacci_series;
}
int main() {
    int n;
    std::cin >> n;
    std::vector<int> result = fibonacci(n);
    std::cout << "[";
    for (size_t i = 0; i < result.size(); ++i) {
        std::cout << result[i];
        if (i < result.size() - 1) {
            std::cout << ", ";
        }
    }
    std::cout << "]" << std::endl;
    return 0;
}
```

Java

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
public class Fibonacci {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        scanner.close();
        if (n <= 0) {
            System.out.println("Input must be greater than 0");
            return;
        }
        List<Integer> fibonacciSeries = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            if (i == 0) {
                fibonacciSeries.add(0);
            } else if (i == 1) {
                fibonacciSeries.add(1);
            } else {
                int nextNumber = fibonacciSeries.get(i - 1) +
fibonacciSeries.get(i - 2);
                fibonacciSeries.add(nextNumber);
            }
        }
        System.out.println(fibonacciSeries);
    }
}
```

13 The enchanted quest (Part III)

7 points

Introduction

Challenge 3: The Matrix of Magical Connections

The third artifact is hidden in a magical matrix, a mysterious grid of numbers. To find the artifact, you must compute the product of two magical matrices. These matrices hold the key to unlocking the artifact's true power. Only by multiplying the matrices will you uncover the artifact's location.

Exercise

Write a program to multiply two 2x2 matrices. The matrices will be represented as lists of lists.

Remember that the multiplication of two matrices with numbers ([13Y],[ZA]) with another like ([TR],[GH]) is:

- In left top position: X∗T + Y∗G
- In right top position: X∗R + Y∗H
- In left bottom position: Z∗T + A∗G
- In right bottom position: Z∗R + A∗H

Input

The input will be both matrices A and B, semicolon separated, and each row, would be comma-separated inside a brackets. That is something like this:

A = [[1, 2], [3, 4]]; B = [[5, 6], [7, 8]]

would mean that matrix A is:

12

34

...and matrix B is:

56

78

Output

The output would the product of both matrices in the same format:

[[19, 22], [43, 50]]

This is because:

$1 * 5 + 2 * 7 = 19$

$1 * 6 + 2 * 8 = 22$

$3 * 5 + 4 * 7 = 43$

$3 * 6 + 4 * 8 = 50$

Example 1

Input

```
A = [[2, 0], [1, 3]]; B = [[4, 1], [2, 3]]
```

Output

```
[[8, 2], [10, 10]]
```

Example 2

Input

```
A = [[1, 2], [3, 4]]; B = [[5, 6], [7, 8]]
```

Output

```
[[19, 22], [43, 50]]
```

Solutions

Python

```
def parse_matrix(matrix_str):
    matrix_str = matrix_str.strip()[matrix_str.index('['):]  # Remove
everything before the first '['
    rows = matrix_str.split("], [")
    matrix = []
    for row in rows:
        matrix.append(list(map(int, row.replace("[", "").replace("]",
"").split(", "))))
    return matrix
def multiply_matrices(A, B):
    m = len(A)
    n = len(A[0])
    p = len(B[0])

    # Initialize the result matrix with zeros
    C = [[0 for _ in range(p)] for _ in range(m)]

    # Perform matrix multiplication
    for i in range(m):
        for j in range(p):
            for k in range(n):
                C[i][j] += A[i][k] * B[k][j]

    return C
def format_matrix(matrix):
    return str(matrix).replace(' ','')
if __name__ == "__main__":
    input_str = input()
    A_str, B_str = input_str.split(";")

    A = parse_matrix(A_str)
    B = parse_matrix(B_str)

    result = multiply_matrices(A, B)
    print(format_matrix(result))
```

C++

```
#include <iostream>
#include <sstream>
#include <vector>
#include <string>
```

```
#include <algorithm> // For std::remove
using namespace std;
// Function to parse a matrix string into a 2D vector
vector<vector<int>> parseMatrix(const string& matrixStr) {
    // Find the start of the matrix data (after "A = ")
    size_t startIdx = matrixStr.find("[[");
    size_t endIdx = matrixStr.rfind("]]");
    // Extract the part of the string containing the matrix
    string matrixContent = matrixStr.substr(startIdx + 1, endIdx - startIdx -
1); // Exclude the outer brackets
    // Use stringstream to process the matrix content
    stringstream ss(matrixContent);
    string rowStr;
    vector<vector<int>> matrix;
    // Parse each row of the matrix
    while (getline(ss, rowStr, ']')) {
        if (rowStr.empty()) continue;
        // Remove any '[' left at the beginning of the row
        size_t startPos = rowStr.find('[');
        if (startPos != string::npos) {
            rowStr = rowStr.substr(startPos + 1); // Remove '['
        }
        // Now split the row by comma to get individual elements
        stringstream rowStream(rowStr);
        string element;
        vector<int> row;
        while (getline(rowStream, element, ',')) {
            if (!element.empty()) {
                row.push_back(stoi(element)); // Convert string to integer
            }
        }
        // Add the row to the matrix
        matrix.push_back(row);
    }
    return matrix;
```

```
}
// Function to multiply two matrices A and B
vector<vector<int>> multiplyMatrices(const vector<vector<int>>& A, const
vector<vector<int>>& B) {
    int m = A.size();     // Number of rows in A
    int n = A[0].size();  // Number of columns in A
    int p = B[0].size();  // Number of columns in B
    // Initialize result matrix C with zeros
    vector<vector<int>> C(m, vector<int>(p, 0));
    // Matrix multiplication logic
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < p; j++) {
            for (int k = 0; k < n; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    return C;
}
// Function to format the matrix as a string (removes spaces)
string formatMatrix(const vector<vector<int>>& matrix) {
    stringstream ss;
    ss << "[";
    for (size_t i = 0; i < matrix.size(); i++) {
        ss << "[";
        for (size_t j = 0; j < matrix[i].size(); j++) {
            ss << matrix[i][j];
            if (j < matrix[i].size() - 1) ss << ", ";
        }
        ss << "]";
        if (i < matrix.size() - 1) ss << ", ";
    }
    ss << "]";
    string formatted = ss.str();
    // Remove spaces from the formatted string
    formatted.erase(remove(formatted.begin(), formatted.end(), ' '),
formatted.end());
    return formatted;
}
int main() {
    string inputStr;
    getline(cin, inputStr);
    // Split input string into matrices
    size_t semicolonIdx = inputStr.find(';');
    string matrixA = inputStr.substr(0, semicolonIdx);
    string matrixB = inputStr.substr(semicolonIdx + 1);
    // Parse matrices
    vector<vector<int>> A = parseMatrix(matrixA);
    vector<vector<int>> B = parseMatrix(matrixB);
    // Multiply matrices
    vector<vector<int>> result = multiplyMatrices(A, B);
    // Format and output the result matrix
    cout << formatMatrix(result) << endl;
    return 0;
}
```

Java

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
public class MatrixMultiplication {
    public static List<List<Integer>> parseMatrix(String matrixStr) {
        matrixStr = matrixStr.trim().substring(matrixStr.indexOf('['));
        String[] rows = matrixStr.split("], \\[");
        List<List<Integer>> matrix = new ArrayList<>();
        for (String row : rows) {
            row = row.replace("[", "").replace("]", "");
            String[] elements = row.split(", ");
            List<Integer> intRow = new ArrayList<>();
            for (String element : elements) {
                intRow.add(Integer.parseInt(element));
            }
            matrix.add(intRow);
        }
        return matrix;
    }
    public static List<List<Integer>> multiplyMatrices(List<List<Integer>> A, 
List<List<Integer>> B) {
        int m = A.size();
        int n = A.get(0).size();
        int p = B.get(0).size();

        List<List<Integer>> C = new ArrayList<>();
        for (int i = 0; i < m; i++) {
            List<Integer> row = new ArrayList<>();
            for (int j = 0; j < p; j++) {
                row.add(0);
            }
            C.add(row);
        }

        for (int i = 0; i < m; i++) {
            for (int j = 0; j < p; j++) {
                for (int k = 0; k < n; k++) {
                    C.get(i).set(j, C.get(i).get(j) + A.get(i).get(k) * 
B.get(k).get(j));
                }
            }
        }

        return C;
    }
    public static String formatMatrix(List<List<Integer>> matrix) {
        return matrix.toString().replace(" ", "");
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String inputStr = scanner.nextLine();
        scanner.close();

        String[] matrices = inputStr.split(";");
        List<List<Integer>> A = parseMatrix(matrices[0]);
        List<List<Integer>> B = parseMatrix(matrices[1]);

        List<List<Integer>> result = multiplyMatrices(A, B);
        //System.out.println(formatMatrix(result));
        System.out.println(formatMatrix(result).replace(" ", ""));
    }
}
```

14 The Enchanted Quest(Part IV)

7 points

Introduction

Challenge 4: The Prime Stone of Perfection

The fourth artifact, the Prime Stone of Perfection, requires you to identify and sum the prime numbers in a range. The stone reveals itself only to those who can discover all the prime numbers up to a certain threshold.

Exercise

Write a program that computes the sum of all prime numbers up to N. A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself.

Input

The input will be a single positive integer number (N). For instance:

```
10
```

Output

The output will be a single positive integer number (N). For instance:

```
17
```

This is because the first primes until 10 are 2,3,5,7, which sums up 17.

Note: 0 and 1 are not considered primes.

Example 1

Input

```
20
```

Output

```
77
```

Example 2

Input

```
50
```

Output

```
328
```

Solutions

Python

```python
def is_prime(num):
    if num <= 1:
        return False
    if num == 2:
        return True
    if num % 2 == 0:
        return False
    for i in range(3, int(num**0.5) + 1, 2):
        if num % i == 0:
            return False
    return True
def sum_of_primes(n):
    sum_primes = 0
    for i in range(2, n + 1):
        if is_prime(i):
            sum_primes += i
    return sum_primes
if __name__ == "__main__":
    n = int(input())
    result = sum_of_primes(n)
    print(result)
```

C++

```cpp
#include <iostream>
#include <cmath>
bool isPrime(int num) {
    if (num <= 1) {
        return false;
    }
    if (num == 2) {
        return true;
    }
    if (num % 2 == 0) {
        return false;
    }
    for (int i = 3; i <= std::sqrt(num); i += 2) {
        if (num % i == 0) {
            return false;
        }
    }
    return true;
}
int sumOfPrimes(int n) {
    int sumPrimes = 0;
    for (int i = 2; i <= n; ++i) {
        if (isPrime(i)) {
            sumPrimes += i;
        }
    }
    return sumPrimes;
}
int main() {
    int n;
    std::cin >> n;

    int result = sumOfPrimes(n);
    std::cout << result << std::endl;

    return 0;
}
```

Java

```
import java.util.Scanner;
public class SumOfPrimes {
    public static boolean isPrime(int num) {
        if (num <= 1) {
            return false;
        }
        if (num == 2) {
            return true;
        }
        if (num % 2 == 0) {
            return false;
        }
        for (int i = 3; i <= Math.sqrt(num); i += 2) {
            if (num % i == 0) {
                return false;
            }
        }
        return true;
    }
    public static int sumOfPrimes(int n) {
        int sumPrimes = 0;
        for (int i = 2; i <= n; i++) {
            if (isPrime(i)) {
                sumPrimes += i;
            }
        }
        return sumPrimes;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        scanner.close();
        int result = sumOfPrimes(n);
        System.out.println(result);
    }
}
```

15 The Enchanted quest (Part V)

7 points

Introduction

Challenge 5: The Portal's Fibonacci Key

The fifth and final artifact is hidden within the Fibonacci Key, which is a magical number derived from the Fibonacci sequence. To retrieve the artifact, you must compute the N-th Fibonacci number using an efficient algorithm. This key is the final piece of the puzzle.

With your mastery of the ancient algorithms, you have now unlocked all five powerful artifacts! Each challenge you faced was a test of your intellect, from sorting the Shifting Stones to unlocking the Portal's Fibonacci Key. With the power of the artifacts, you are now ready to confront the ancient curse and restore peace to the kingdom!

Exercise

You must write a program that calculate the N-th element of the Fibonacci series

Input

The program will read a single integer positive number indicating N (the fibonacci series element we are requesting). For instance:

```
5
```

Output

The program will simply write a positive integer showing the Nth element of the fibonacci series. For instance:

```
5
```

This is because the Fibonacci series is 1 , 1 , 2 , 3 , 5 , 8 , 13...

Example 1

Input

```
5
```

Output

```
5
```

Example 2

Input

```
8
```

Output

```
21
```

Solutions

Python

```python
def fibonacci(n):
    if n <= 0:
        return "Input must be a positive integer"
    elif n == 1 or n == 2:
        return 1
    else:
        a, b = 1, 1
        for _ in range(3, n + 1):
            a, b = b, a + b
        return b
if __name__ == "__main__":
    n = int(input())
    result = fibonacci(n)
    print(result)
```

C++

```cpp
#include <iostream>
#include <stdexcept>
int fibonacci(int n) {
    if (n <= 0) {
        throw std::invalid_argument("Input must be a positive integer");
    } else if (n == 1 || n == 2) {
        return 1;
    } else {
        int a = 1, b = 1;
        for (int i = 3; i <= n; ++i) {
            int temp = a + b;
            a = b;
            b = temp;
        }
        return b;
    }
}
int main() {
    int n;
    std::cin >> n;

    try {
        int result = fibonacci(n);
        std::cout << result << std::endl;
    } catch (const std::invalid_argument& e) {
        std::cerr << e.what() << std::endl;
    }

    return 0;
}
```

Java

```
import java.util.Scanner;
public class Fibonacci {
    public static int fibonacci(int n) {
        if (n <= 0) {
            throw new IllegalArgumentException("Input must be a positive
integer");
        } else if (n == 1 || n == 2) {
            return 1;
        } else {
            int a = 1, b = 1;
            for (int i = 3; i <= n; i++) {
                int temp = a + b;
                a = b;
                b = temp;
            }
            return b;
        }
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        scanner.close();

        try {
            int result = fibonacci(n);
            System.out.println(result);
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

16 The labyrinth

7 points

Introduction

In the heart of a forgotten kingdom, hidden deep within the labyrinthine ruins of an ancient castle, lies a deadly maze. The whispers of old legends tell of a creature, the Minotaur, who roams its endless corridors, guarding the secrets of the castle's dark past. Many have ventured inside, but none have returned.

To survive, you must navigate the treacherous maze, making at least ten deliberate moves to reach a safe place within its depths. But be warned: the Minotaur watches your every move. If you stray too far or take fewer than ten steps, the creature will sense your presence and strike, ending your quest in an instant.

Choose your path wisely, and may you outsmart the labyrinth… or risk becoming just another victim of the Minotaur's wrath.

Exercise

In this exercise, you will navigate a 5x5 grid of rooms, starting at the bottom-left corner (position 1,1). Your task is to make at least 10 moves to explore the maze using the following commands: (u)p, (l)eft, (r)ight, and (d)own.

However, there's a catch: the Minotaur is lurking, and if you fail to make at least 10 moves or if you step outside the boundaries of the maze, the Minotaur will strike, and you'll be out of the game. So, plan your path wisely, stay within the maze, and make it to the end without being caught!

Input

The input will be a single text line with a set of 10 comma-separated instructions indicating the directions to where you will move inside the labyrinth. Only 4 letters are allowed: u,r,l,d, indicating Up, Right, Left and Down. An example would be:

```
u,u,u,u,r,r,r,r,d,d
```

Output

The output will be a single line with the structure:

Final position: (16, Y)

....where X and Y is the position where you will end your journey inside the labyrinth. An example of output would be:

```
Final position: (5, 3)
```

If you find the exit, the output would be:

```
Exited the maze at step 7
```

Or if any of the letters is not valid, the output will be:

```
Invalid move 'x' at step 3
```

Or if you do not enter 10 movements, the output will be:

```
The input must contain at least 10 moves.
```

Example 1

Input

```
r,r,u,u,u,l,d,d,d,l
```

Output

```
Final position: (1, 1)
```

Example 2

Input

```
u,u,r,r,d,d,d,r,r,l
```

Output

```
Exited the maze at step 7
```

Example 3

Input

```
u,u,x,x,s,s,r,r,u,u
```

Output

```
Invalid move 'x' at step 3
```

Example 4

Input

```
u,u,r,r
```

Output

```
The input must contain at least 10 moves.
```

Solutions

Python

```
def move_in_maze(moves):
    # Inicializar la posición inicial
    x, y = 1, 1
    # Definir los límites de la matriz
    min_x, max_x = 1, 5
    min_y, max_y = 1, 5
    # Recorrer los movimientos
    for step, move in enumerate(moves, start=1):
        if move == "u":
            y += 1
        elif move == "d":
            y -= 1
        elif move == "l":
            x -= 1
        elif move == "r":
            x += 1
        else:
            print(f"Invalid move '{move}' at step {step}")
            return
        # Verificar si se ha salido del laberinto
        if x < min_x or x > max_x or y < min_y or y > max_y:
            print(f"Exited the maze at step {step}")
            return
    # Imprimir la posición final
    print(f"Final position: ({x}, {y})")
# Obtener la entrada del usuario
```

```
moves = input().split(',')
# Verificar que la entrada tenga al menos 10 movimientos
if len(moves) < 10:
    print("The input must contain at least 10 moves.")
else:
    move_in_maze(moves)
```

C++

```
#include <iostream>
#include <vector>
#include <sstream>
void move_in_maze(const std::vector<std::string>& moves) {
    // Initialize the starting position
    int x = 1, y = 1;
    // Define the matrix limits
    int min_x = 1, max_x = 5;
    int min_y = 1, max_y = 5;
    // Iterate through the moves
    for (size_t step = 0; step < moves.size(); ++step) {
        const std::string& move = moves[step];
        if (move == "u") {
            y += 1;
        } else if (move == "d") {
            y -= 1;
        } else if (move == "l") {
            x -= 1;
        } else if (move == "r") {
            x += 1;
        } else {
            std::cout << "Invalid move '" << move << "' at step " << step + 1
<< std::endl;
            return;
        }
        // Check if exited the maze
        if (x < min_x || x > max_x || y < min_y || y > max_y) {
                std::cout << "Exited the maze at step " << step + 1 << std::endl;
                return;
            }
        }
    // Print the final position
    std::cout << "Final position: (" << x << ", " << y << ")" << std::endl;
}
int main() {
    std::string input;
    std::getline(std::cin, input);
    std::istringstream iss(input);
    std::vector<std::string> moves;
    std::string move;
    while (std::getline(iss, move, ',')) {
        moves.push_back(move);
    }
    // Check that the input contains at least 10 moves
    if (moves.size() < 10) {
        std::cout << "The input must contain at least 10 moves." <<
std::endl;
    } else {
        move_in_maze(moves);
    }
    return 0;
}
```

Java

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
class Main {
    public static void moveInMaze(List<String> moves) {
        // Initialize the starting position
        int x = 1, y = 1;
        // Define the matrix limits
        int minX = 1, maxX = 5;
        int minY = 1, maxY = 5;
        // Iterate through the moves
        for (int step = 0; step < moves.size(); step++) {
            String move = moves.get(step);
            switch (move) {
                case "u":
                    y += 1;
                    break;
                case "d":
                    y -= 1;
                    break;
                case "l":
                    x -= 1;
                    break;
                case "r":
                    x += 1;
                    break;
                default:
                    System.out.println("Invalid move '" + move + "' at step "
+ (step + 1));
                    return;
            }
            // Check if exited the maze
            if (x < minX || x > maxX || y < minY || y > maxY) {
                System.out.println("Exited the maze at step " + (step + 1));
                return;
            }
        }
        // Print the final position
        System.out.println("Final position: (" + x + ", " + y + ")");
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();
        String[] moveArray = input.split(",");
        List<String> moves = new ArrayList<>();
        for (String move : moveArray) {
            moves.add(move);
        }
        // Check that the input contains at least 10 moves
        if (moves.size() < 10) {
            System.out.println("The input must contain at least 10 moves.");
        } else {
            moveInMaze(moves);
        }
        scanner.close();
    }
}
```

17 The Perfect Power

7 points

Introduction

A new encryption algorithm has been discovered that is based in a big number that complies the requisite of being a perfect power. That is, for this algorithm it's necessary to find the lowest perfect power of different 'powers'. For instance if the poers are 2 and 3, we need to find the lowest number which is a the same a power of and a power 3. In this example, that number would be 64 because it's the lowest number which is a power of 2 (8 x 8 = 64) and a power of 3 (4 x 4 x 4 = 64).

Exercise

We need your help to write a program to find the lowest perfect power given a series of 'powers' that will be received from the keyboard.

Input

The program will receive a line with up to 3 non-repetitive numbers from 1 to 9, separated by a semicolon (;) and will have to find the lowest perfect power of them.

For example, it could receive something like:

```
2;3
```

and you will have to find which is the lowest perfect power (64) and write it:

```
64
```

because it's 8∗8 (2 numbers) is 64 and 4∗4∗4 (3 numbers) is 64.

Another Example. With the input:

```
2;3;5
```

The solution would be

```
1073741824
```

(Because it's *32768∗32768*, *1024∗1024∗1024* and *64∗64∗64∗64∗64*)

Output

The output will be a single integer number which is the result of the previously explained operation.

Example 1

Input

```
2;3
```

Output

```
64
```

Example 2

Input

```
2;3;5
```

Output

```
1073741824
```

Solutions

Python

```
from math import prod
print(2**prod(int(i) for i in input().split(';')))
```

C++

```
#include <iostream>
#include <sstream>
#include <vector>
#include <cmath>
int main() {
    std::string input;
    std::getline(std::cin, input);  // Read the entire input line
    std::stringstream ss(input);  // Create a stringstream from the input
    std::string token;
    std::vector<int> numbers;
    // Split the input string by semicolons and store the numbers in a vector
    while (std::getline(ss, token, ';')) {
        numbers.push_back(std::stoi(token));  // Convert string to int and
add to the vector
    }
    // Compute the product of the numbers
    int product = 1;
    for (int num : numbers) {
        product *= num;
    }
    // Calculate 2 raised to the power of the product and print it
    std::cout << static_cast<int>(pow(2, product)) << std::endl;
    return 0;
}
```

Java

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();  // Read the entire input line
        // Split the input string by semicolons and parse the integers
        String[] tokens = input.split(";");
        int product = 1;
        // Compute the product of the numbers
        for (String token : tokens) {
            product *= Integer.parseInt(token);  // Convert string to integer
and multiply
        }
        // Calculate 2 raised to the power of the product and print it
        System.out.println((int) Math.pow(2, product));  // Use Math.pow and
cast to int
        scanner.close();
    }
    }
```

18 The Riddle of the Golden Numbers

7 points

Introduction

In a distant kingdom called Numeria, the 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 a special value, 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 lay still and mysterious in the center of the Temple of Eternal Sequences.

The Challenge of the Young Programmer

In a small village in Numeria, lived a young apprentice named *Auron*. He was passionate about solving problems with modern tools. While the wise mathematicians debated and wrote endless lists of numbers, Auron had a revolutionary idea: to use the recently invented calculation machine to decipher the riddle.

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

Exercise

Act as Auron and create an application that asks for a number and returns the exact sum of all golden numbers lower that specified number.

Input

A single line with a positive integer number. For instance 10.

Output

A single line with the golden number:

17

Example 1

Input

```
10
```

Output

```
17
```

Example 2

Input

```
20
```

Output

```
77
```

Solutions

Python

```
# Function to check if a number is prime
def is_prime(num):
    if num < 2:
        return False
    for i in range(2, int(num**0.5) + 1):  # Check divisors up to the square
root
        if num % i == 0:
            return False
    return True
# Main function
def main():
    try:
        N = int(input())
    except ValueError:
        print("Invalid input.")
        return

    if N < 0:
        print("Invalid input.")
        return
    if N < 2:
        print("0")
        return
    # Calculate the sum of primes
    total_sum = 0
    for i in range(2, N + 1):
        if is_prime(i):
            total_sum += i
    print(total_sum)
if __name__ == "__main__":
    main()
```

C++

```
#include <iostream>
using namespace std;
bool isPrime(int num) {
    if (num < 2) return false;
    for (int i = 2; i * i <= num; i++) { // Check divisors up to the square
root
        if (num % i == 0) return false;
    }
    return true;
}
int main() {
    int N;
    cin >> N;
    if (cin.fail() || N < 0) {
        cout << "Invalid input." << endl;
        return 0;
    }

    if (N < 2) {
        printf("0\n", N);
        return 0;
    }
    // Calculate the sum of primes
    int sum = 0;
    for (int i = 2; i <= N; i++) {
        if (isPrime(i)) {
            sum += i;
        }
    }
    // Output the result
    printf("%d\n", sum);
    return 0;
}
```

Java

```
import java.util.Scanner;
public class goldenNumbers {
    // Function to check if a number is prime
    public static boolean isPrime(int num) {
        if (num < 2) {
            return false;
        }
        for (int i = 2; i * i <= num; i++) {  // Check divisors up to the
square root
            if (num % i == 0) {
                return false;
            }
        }
        return true;
    }
    // Main function
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int N;

        if (!scanner.hasNextInt()) {
            System.out.println("Invalid input.");
            return;
        }

        N = scanner.nextInt();

        if (N < 0) {
            System.out.println("Invalid input.");
            return;
        }
        if (N < 2) {
            System.out.println("0");
            return;
        }
        // Calculate the sum of primes
        int totalSum = 0;
        for (int i = 2; i <= N; i++) {
            if (isPrime(i)) {
                totalSum += i;
            }
        }
        System.out.println(totalSum);
    }
}
```

19 Building a calculator II

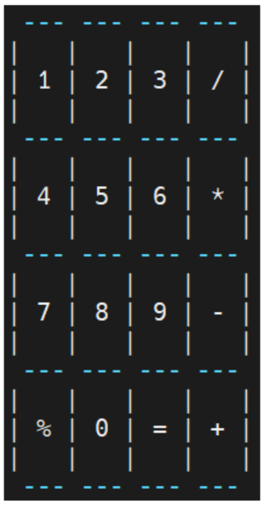
8 points

Introduction

In your school they have asked you to write a program that simulate a simple calculator.

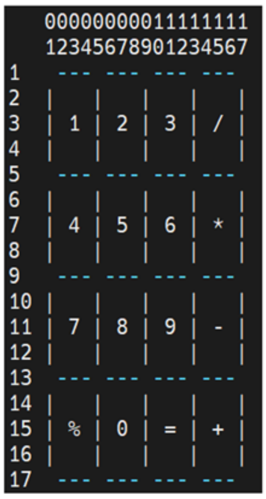
In the first evaluation you got a great grade with that code you prepared for the previous exercise. Now you are expected to simulate a graphical interface of a calculator.

The following will be the graphical interface of your calculator:



```
 ---  ---  ---  ---
|    |    |    |    |
|  1 |  2 |  3 |  / |
|    |    |    |    |
 ---  ---  ---  ---
|    |    |    |    |
|  4 |  5 |  6 |  * |
|    |    |    |    |
 ---  ---  ---  ---
|    |    |    |    |
|  7 |  8 |  9 |  - |
|    |    |    |    |
 ---  ---  ---  ---
|    |    |    |    |
|  % |  0 |  = |  + |
|    |    |    |    |
 ---  ---  ---  ---
```

We want to simulate the mouse as well, so what you are going to receive are screen position where the mouse button was clicked. The coordinates of this screen are the following:



```
   00000000011111111
   12345678901234567
1   --- --- --- ---
2  |   |   |   |   |
3  | 1 | 2 | 3 | / |
4  |   |   |   |   |
5   --- --- --- ---
6  |   |   |   |   |
7  | 4 | 5 | 6 | * |
8  |   |   |   |   |
9   --- --- --- ---
10 |   |   |   |   |
11 | 7 | 8 | 9 | - |
12 |   |   |   |   |
13  --- --- --- ---
14 |   |   |   |   |
15 | % | 0 | = | + |
16 |   |   |   |   |
17  --- --- --- ---
```

Therefore if you receive the coordinates 12,4 (the first is the 19 and the second is the Y ordinate), you would be clicking button "3".

And if you are receiving coordinates 10,14, then the user is pressing the "equals" button.

The mouse has a small program (called driver) that will detect if you are clicking very close to the button separation lines and will adjust the screen position of the click. This means that no position of a separation line will be considered as an input value: Therefore lines 1,5,9,13 and 17 will never be pressed and columns 1,5,9,13 and 17 will never be pressed either.

For instance, your program will receive something like:

04,04;14,14;10,10;12,14

You will have to decipher the buttons pressed. 04,04 is the button 1. 14,14 is the button '+', 10,10 is the button 9 and 16,14 is the button '='.

Once deciphered you will have obtain the operation: "1 + 9 ="

Following the previous exercise, the user will only be able to press buttons to write a simple operation with two numbers, an operation symbol and a equals symbol at the end, and your program will return the result of the operation.

In this case the output will be:

10

Exercise

Write a program that reads the mouse click positions of the calculator and return the solution of the operation clicked in the calculator.

Note: The calculator will not consider divisions by zero (They won't be tested)

Note: The operations will be simple and if you have obtained the previous exercise you can transfer the code (with minor changes) to this exercise.

Input

A single line of semicolon-separated pairs of comma-separated integers. That is something like:

04,04;14,14;10,10;12,14

Note: The last clicked button will always be the "=" button.

Output

A single number with the solution.

Note: Divisions will only show (if necessary) 2 decimal digits.

Example 1

Input

```
10,11;11,12;3,15;3,3;8,2;12,16
```

Output

```
3
```

Example 2

Input

```
2,2;6,6;16,2;6,8;10,16
```

Output

```
3
```

Example 3

Input

```
4,4;11,6;14,4;11,3;10,15
```

Output

```
5.33
```

Solutions

Python

```
def get_button(x, y):
    if 2 <= x <= 4:
        if 2 <= y <= 4:
            return '1'
        elif 6 <= y <= 8:
            return '4'
        elif 10 <= y <= 12:
            return '7'
        elif 14 <= y <= 16:
            return '%'
    elif 6 <= x <= 8:
        if 2 <= y <= 4:
            return '2'
        elif 6 <= y <= 8:
            return '5'
        elif 10 <= y <= 12:
            return '8'
        elif 14 <= y <= 16:
            return '0'
    elif 10 <= x <= 12:
        if 2 <= y <= 4:
            return '3'
        elif 6 <= y <= 8:
            return '6'
        elif 10 <= y <= 12:
            return '9'
        elif 14 <= y <= 16:
            return '='
    elif 14 <= x <= 16:
        if 2 <= y <= 4:
            return '/'
        elif 6 <= y <= 8:
            return '*'
        elif 10 <= y <= 12:
            return '-'
        elif 14 <= y <= 16:
            return '+'
    return None
def get_text(input_text):
    pairs = input_text.split(';')
    result = []
    for pair in pairs:
        x, y = map(int, pair.split(','))
        button = get_button(x, y)
        if button in ['+', '-', '*', '/', '%', '=']:
            result.append(' ')
            result.append(button)
            result.append(' ')
        else:
            result.append(button)
    return ''.join(result)
def calculate(a, operator, b):
    if operator == '+':
        return a + b
    elif operator == '-':
        return a - b
    elif operator == '*':
        return a * b
    elif operator == '/':
        if b != 0:
            if ((a / b) %1==0):
                return int(a/b)
            else:
                return round(a / b, 2)
            return round(a / b, 2)
        else:
            raise ValueError("Division by zero")
    elif operator == '%':
        if b != 0:
            return a % b
        else:
            raise ValueError("Division by zero")
    else:
        raise ValueError("Invalid operator")
def process_input(input_line):
    # Remove the '=' sign and split the input into parts
    input_line = input_line.replace("=", "").strip()
    parts = input_line.split()
    if len(parts) != 3:
        return "Invalid input format"
    try:
        a = int(parts[0])
        operator = parts[1]
        b = int(parts[2])
        result = calculate(a, operator, b)
        return result
    except ValueError as e:
        return str(e)
# Example usage
#input_text = "04,04;14,14;10,10;12,14"
input_text = input("").strip()
output = get_text(input_text)
#print(output)
output2= process_input(output)
print(output2)
```

C++

```cpp
#include <iostream>
#include <sstream>
#include <vector>
#include <string>
#include <stdexcept>
#include <iomanip>
#include <cmath>
#include <algorithm>
char get_button(int x, int y) {
    if (2 <= x && x <= 4) {
        if (2 <= y && y <= 4) {
            return '1';
        } else if (6 <= y && y <= 8) {
            return '4';
        } else if (10 <= y && y <= 12) {
            return '7';
        } else if (14 <= y && y <= 16) {
            return '%';
        }
    } else if (6 <= x && x <= 8) {
        if (2 <= y && y <= 4) {
            return '2';
        } else if (6 <= y && y <= 8) {
            return '5';
        } else if (10 <= y && y <= 12) {
            return '8';
        } else if (14 <= y && y <= 16) {
            return '0';
        }
    } else if (10 <= x && x <= 12) {
        if (2 <= y && y <= 4) {
            return '3';
        } else if (6 <= y && y <= 8) {
            return '6';
        } else if (10 <= y && y <= 12) {
            return '9';
        } else if (14 <= y && y <= 16) {
            return '=';
        }
    } else if (14 <= x && x <= 16) {
        if (2 <= y && y <= 4) {
            return '/';
        } else if (6 <= y && y <= 8) {
            return '*';
        } else if (10 <= y && y <= 12) {
            return '-';
        } else if (14 <= y && y <= 16) {
            return '+';
        }
    }
    return '?';  // In case of invalid coordinates
}
std::string get_text(const std::string& input_text) {
    std::istringstream iss(input_text);
    std::string token;
    std::vector<std::string> pairs;
    std::string result;
    while (std::getline(iss, token, ';')) {
        pairs.push_back(token);
    }
    for (const auto& pair : pairs) {
        std::istringstream pair_stream(pair);
        std::string x_str, y_str;
        std::getline(pair_stream, x_str, ',');
        std::getline(pair_stream, y_str, ',');
        int x = std::stoi(x_str);
        int y = std::stoi(y_str);
        char button = get_button(x, y);
        if (button == '+' || button == '-' || button == '*' || button == '/'
|| button == '%' || button == '=') {
            result += ' ';
            result += button;
            result += ' ';
        } else {
            result += button;
        }
    }
    return result;
}
double calculate(int a, const std::string& operator_, int b) {
    if (operator_ == "+") {
        return a + b;
    } else if (operator_ == "-") {
        return a - b;
    } else if (operator_ == "*") {
        return a * b;
    } else if (operator_ == "/") {
        if (b != 0) {
            double result = static_cast<double>(a) / b;
            if (result == static_cast<int>(result)) {
                return static_cast<int>(result);
            } else {
                return std::floor(result * 100 + 0.5) / 100;
            }
        } else {
            throw std::invalid_argument("Division by zero");
        }
    } else if (operator_ == "%") {
        if (b != 0) {
            return a % b;
        } else {
            throw std::invalid_argument("Division by zero");
        }
    } else {
        throw std::invalid_argument("Invalid operator");
    }
}
std::string process_input(const std::string& input_line) {
    std::string line = input_line;
    line.erase(std::remove(line.begin(), line.end(), '='), line.end());
    std::istringstream iss(line);
    std::vector<std::string> parts;
    std::string part;
    while (iss >> part) {
        parts.push_back(part);
    }
    if (parts.size() != 3) {
        return "Invalid input format";
    }
    try {
        int a = std::stoi(parts[0]);
        std::string operator_ = parts[1];
        int b = std::stoi(parts[2]);
        double result = calculate(a, operator_, b);
        std::ostringstream oss;
        if (operator_ == "/") {
             if (std::floor(result) == result) {
                //oss <<  std::fixed << std::setprecision(2) << result;
                oss << static_cast<int>(result);
             } else if (std::floor(result*10) == result*10) {
                oss << std::fixed << std::setprecision(1) << result;
             } else {
                oss << std::fixed << std::setprecision(2) << result;
             }
        } else {
            oss << static_cast<int>(result);
        }
        return oss.str();
    } catch (const std::invalid_argument& e) {
        return e.what();
    }
}
int main() {
    std::string input_text;
    std::getline(std::cin, input_text);
    std::string output = get_text(input_text);
    std::string output2 = process_input(output);
    std::cout << output2 << std::endl;
    return 0;
}
```

Java

```
import java.util.*;
public class calculatorII {
    public static char getButton(int x, int y) {
        if (2 <= x && x <= 4) {
            if (2 <= y && y <= 4) {
                return '1';
            } else if (6 <= y && y <= 8) {
                return '4';
            } else if (10 <= y && y <= 12) {
                return '7';
            } else if (14 <= y && y <= 16) {
                return '%';
            }
        } else if (6 <= x && x <= 8) {
            if (2 <= y && y <= 4) {
                return '2';
            } else if (6 <= y && y <= 8) {
                return '5';
            } else if (10 <= y && y <= 12) {
                return '8';
            } else if (14 <= y && y <= 16) {
                return '0';
            }
        } else if (10 <= x && x <= 12) {
            if (2 <= y && y <= 4) {
                return '3';
            } else if (6 <= y && y <= 8) {
                return '6';
            } else if (10 <= y && y <= 12) {
                return '9';
            } else if (14 <= y && y <= 16) {
                return '=';
            }
        } else if (14 <= x && x <= 16) {
            if (2 <= y && y <= 4) {
                return '/';
            } else if (6 <= y && y <= 8) {
                return '*';
            } else if (10 <= y && y <= 12) {
                return '-';
            } else if (14 <= y && y <= 16) {
                return '+';
            }
        }
        return '?';  // In case of invalid coordinates
    }
    public static String getText(String inputText) {
        String[] pairs = inputText.split(";");
        StringBuilder result = new StringBuilder();
        for (String pair : pairs) {
            String[] coordinates = pair.split(",");
            int x = Integer.parseInt(coordinates[0]);
            int y = Integer.parseInt(coordinates[1]);
            char button = getButton(x, y);
            if (button == '+' || button == '-' || button == '*' || button ==
'/' || button == '%' || button == '=') {
```

```
                result.append(' ').append(button).append(' ');
            } else {
                result.append(button);
            }
        }
        return result.toString();
    }
    public static double calculate(int a, String operator, int b) {
        switch (operator) {
            case "+":
                return a + b;
            case "-":
                return a - b;
            case "*":
                return a * b;
            case "/":
                if (b != 0) {
                    double result = (double) a / b;
                    if (result == (int) result) {
                        return (int) result;
                    } else {
                        return Math.round(result * 100.0) / 100.0;
                    }
                } else {
                    throw new ArithmeticException("Division by zero");
                }
            case "%":
                if (b != 0) {
                    return a % b;
                } else {
                    throw new ArithmeticException("Division by zero");
                }
            default:
                throw new IllegalArgumentException("Invalid operator");
        }
    }
    public static String processInput(String inputLine) {
        inputLine = inputLine.replace("=", "").trim();
        String[] parts = inputLine.split("\\s+");
        if (parts.length != 3) {
            return "Invalid input format";
        }
        try {
            int a = Integer.parseInt(parts[0]);
            String operator = parts[1];
            int b = Integer.parseInt(parts[2]);
            double result = calculate(a, operator, b);
            if (operator.equals("/")) {
                if (result % 1 == 0) {
                    return String.valueOf(((int) result));
                } else {
                    // Format result with up to 2 decimals
                    return (String.format(Locale.US, "%.2f", result)
                            .replaceAll("\\.0+$|(?<=\\.\\d)0+$", ""));
                }
            } else {
                return String.format(Locale.US,"%.0f", result);
            }
        } catch (NumberFormatException e) {
            return "Invalid number format";
        } catch (IllegalArgumentException | ArithmeticException e) {
            return e.getMessage();
        }
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String inputText = scanner.nextLine().trim();
        String output = getText(inputText);
        String output2 = processInput(output);
        System.out.println(output2);
        }
    }
```

20 The mysterious world of Zyntar

8 points

Introduction

You venture into a remote and unknown universe, a place that does not appear on conventional maps. This is the Kingdom of Zyntar, a world filled with magic, mysteries, and numerical secrets. In Zyntar, numbers are not mere figures; they are mystical creatures that communicate through an ancient and forgotten language, known only as "Zynka."

Centuries ago, the inhabitants of Zyntar mastered the language of numbers, a mystical art that transformed any number into a powerful word. However, over time, the Zynka language has faded, and now only a select few have the ability to translate Zyntar's Mystical Numbers back into their original form as human words.

Upon arriving in Zyntar, humans have discovered that the Zynka language closely resembles the way numbers are written in **Spanish**.

Exercise

You are one of the chosen ones to go to the Great Tower of Zyntar and translate the lost language of numbers. Your mission is to decipher the Mystical Numbers and restore them to their word form in Spanish. When you encounter a number, you must translate it into its Zynka equivalent and return it in the human language.

For example, if you receive the number 2904, your task will be to return **"dos mil novecientos cuatro"**. If the number is 608, you must return **"seiscientos ocho"**. No matter how big or small the number is, the Zynka language has a special way of expressing it… and you will be the translator!

Challenge Rules:

- The inhabitants of the world of Zyntar are numbers with up to 4 digits. There are only 9,999 inhabitants.
- You must convert the number into the Zynka language, meaning you must translate it into words in Spanish.

Remember to use the correct connectors, such as **"y"** between tens and units when necessary.

These are Mystical Numbers. They have their own magical essence. If you do not translate them correctly, numerical chaos will be unleashed in the Kingdom of Zyntar, and the numerical creatures will scatter across the universe. It is your duty to restore order and return power to the Mystical Numbers.

Input

A single line with a positive integer number between 0 to 9999.

For instance:

2904

Output

Its translation into Zyntar (or... spanish)

Dos mil novecientos cuatro

Note: remember the first letter in capitals...

Example 1

Input

```
9999
```

Output

```
Nueve mil novecientos noventa y nueve
```

Example 2

Input

```
10
```

Output

```
Diez
```

Solutions

Python

```
def convertir_a_palabras(num):
    # Diccionarios para las palabras en español
    un_digito = ["", "uno", "dos", "tres", "cuatro", "cinco", "seis",
"siete", "ocho", "nueve"]
    dos_digitos = ["", "", "veinte", "treinta", "cuarenta", "cincuenta",
"sesenta", "setenta", "ochenta", "noventa"]
    decenas_especiales = ["", "diez", "once", "doce", "trece", "catorce",
"quince", "dieciséis", "diecisiete", "dieciocho", "diecinueve"]
    veintenas = ["", "veintiuno", "veintidos", "veintitres"]
    centenas = ["", "cien", "doscientos", "trescientos", "cuatrocientos",
"quinientos", "seiscientos", "setecientos", "ochocientos", "novecientos"]
    # Casos base de 1-9, 10-19, 20-99 y 100-999
    def convertir_3_digitos(n):
        if n == 100:
            return "cien"
        elif n > 100:
            c = n // 100
            d = n % 100
            if d == 0:
                return centenas[c]
            else:
                return centenas[c] + " " + convertir_2_digitos(d)
        else:
            return convertir_2_digitos(n)
    def convertir_2_digitos(n):
        if n==10:
            return "diez"
        if n < 10:
            return un_digito[n]
        elif 10 <= n <= 19:
            return decenas_especiales[n - 9]
        elif 21 <= n <= 23:
            return veintenas[n - 20]
        else:
            d = n // 10
            u = n % 10
            if u == 0:
                return dos_digitos[d]
            else:
                return dos_digitos[d] + " y " + un_digito[u]
    # Función principal para números hasta 9999
    if num == 0:
        return "cero"
    elif num < 10:
        return un_digito[num]  # Para números entre 0 y 9
    elif num < 100:
        return convertir_2_digitos(num)  # Para números entre 10 y 99
    elif num < 1000:
        return convertir_3_digitos(num)  # Para números entre 100 y 999
    else:
        miles = num // 1000
        resto = num % 1000
        if miles == 1:
            # Si es "mil" (no "uno mil")
            if resto == 0:
                return "mil"
            else:
                return "mil " + convertir_3_digitos(resto)
        else:
            if resto == 0:
                return un_digito[miles] + " mil"
            else:
                return un_digito[miles] + " mil " +
convertir_3_digitos(resto)
# Función principal para interactuar con el usuario
def main():
    # Leer el número desde la entrada
    numero = int(input())

    # Convertir el número a palabras
    resultado = convertir_a_palabras(numero)
    # Mostrar el resultado, asegurándose de que la primera letra sea
mayúscula
    print(resultado.capitalize())
if __name__ == "__main__":
    main()
```

C++

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
vector<string> un_digito = {"", "uno", "dos", "tres", "cuatro", "cinco",
"seis", "siete", "ocho", "nueve"};
vector<string> dos_digitos = {"", "", "veinte", "treinta", "cuarenta",
"cincuenta", "sesenta", "setenta", "ochenta", "noventa"};
vector<string> decenas_especiales = {"", "diez", "once", "doce", "trece",
"catorce", "quince", "dieciséis", "diecisiete", "dieciocho", "diecinueve"};
vector<string> veintenas = {"", "veintiuno", "veintidos", "veintitres"};
vector<string> centenas = {"", "cien", "doscientos", "trescientos",
"cuatrocientos", "quinientos", "seiscientos", "setecientos", "ochocientos",
"novecientos"};
// Función para convertir 2 dígitos
string convertir_2_digitos(int n) {
    if (n == 10) {
        return "diez";
    }
    if (n < 10) {
        return un_digito[n];
    } else if (n >= 10 && n <= 19) {
        return decenas_especiales[n - 10];
    } else if (n >= 21 && n <= 23) {
        return veintenas[n - 20];
    } else {
        int d = n / 10;
        int u = n % 10;
        if (u == 0) {
            return dos_digitos[d];
        } else {
            return dos_digitos[d] + " y " + un_digito[u];
        }
    }
}
// Función para convertir 3 dígitos
string convertir_3_digitos(int n) {
    if (n == 100) {
        return "cien";
    } else if (n > 100) {
        int c = n / 100;
        int d = n % 100;
        if (d == 0) {
            return centenas[c];
        } else {
            return centenas[c] + " " + convertir_2_digitos(d);
        }
    } else {
        return convertir_2_digitos(n);
    }
}
// Función para convertir el número a palabras
string convertir_a_palabras(int num) {
    if (num == 0) {
        return "cero";
    } else if (num < 10) {
        return un_digito[num];  // Para números entre 0 y 9
    } else if (num < 100) {
        return convertir_2_digitos(num);  // Para números entre 10 y 99
    } else if (num < 1000) {
        return convertir_3_digitos(num);  // Para números entre 100 y 999
    } else {
        int miles = num / 1000;
        int resto = num % 1000;
        if (miles == 1) {
            // Si es "mil" (no "uno mil")
            if (resto == 0) {
                return "mil";
            } else {
                return "mil " + convertir_3_digitos(resto);
            }
        } else {
            if (resto == 0) {
                return un_digito[miles] + " mil";
            } else {
                return un_digito[miles] + " mil " +
convertir_3_digitos(resto);
            }
        }
    }
}
int main() {
    int numero;
    cin >> numero;
    // Convertir el número a palabras
    string resultado = convertir_a_palabras(numero);
    // Mostrar el resultado, asegurándose de que la primera letra sea
mayúscula
    resultado[0] = toupper(resultado[0]);
    cout << resultado << endl;
    return 0;
}
```

Java

```
import java.util.Scanner;
public class Zyntar {
    // Diccionarios para las palabras en español
    private static final String[] unDigito = {"", "uno", "dos", "tres",
"cuatro", "cinco", "seis", "siete", "ocho", "nueve"};
    private static final String[] dosDigitos = {"", "", "veinte", "treinta",
"cuarenta", "cincuenta", "sesenta", "setenta", "ochenta", "noventa"};
    private static final String[] decenasEspeciales = {"", "diez", "once",
"doce", "trece", "catorce", "quince", "dieciséis", "diecisiete", "dieciocho",
"diecinueve"};
    private static final String[] centenas = {"", "cien", "doscientos",
"trescientos", "cuatrocientos", "quinientos", "seiscientos", "setecientos",
"ochocientos", "novecientos"};
    public static void main(String[] args) {
        // Crear un objeto Scanner para leer la entrada del usuario
        Scanner scanner = new Scanner(System.in);
        // Leer el número desde la entrada
        int numero = scanner.nextInt();
        // Convertir el número a palabras
        String resultado = convertirAPalabras(numero);
        // Mostrar el resultado, asegurándose de que la primera letra sea
mayúscula
        System.out.println(resultado.substring(0, 1).toUpperCase() +
resultado.substring(1));
    }
    // Función para convertir el número a palabras
    public static String convertirAPalabras(int num) {
        // Función principal para números hasta 9999
        if (num == 0) {
            return "cero";
        } else if (num < 10) {
            return unDigito[num];  // Para números entre 0 y 9
        } else if (num < 100) {
            return convertir2Digitos(num);  // Para números entre 10 y 99
        } else if (num < 1000) {
            return convertir3Digitos(num);  // Para números entre 100 y 999
        } else {
            int miles = num / 1000;
            int resto = num % 1000;
            if (miles == 1) {
                // Si es "mil" (no "uno mil")
                if (resto == 0) {
                    return "mil";
                } else {
                    return "mil " + convertir3Digitos(resto);
                }
            } else {
                if (resto == 0) {
                    return unDigito[miles] + " mil";
                } else {
                    return unDigito[miles] + " mil " +
convertir3Digitos(resto);
                }
            }
        }
    }
    // Función para convertir 3 dígitos
    private static String convertir3Digitos(int n) {
        if (n == 100) {
            return "cien";
        } else if (n > 100) {
            int c = n / 100;
            int d = n % 100;
            if (d == 0) {
                return centenas[c];
            } else {
                return centenas[c] + " " + convertir2Digitos(d);
            }
        } else {
            return convertir2Digitos(n);
        }
    }
    // Función para convertir 2 dígitos
```

```
        private static String convertir2Digitos(int n) {
            if (n == 10) {
                return "diez";
            }
            if (n < 10) {
                return unDigito[n];
            } else if (n >= 10 && n <= 19) {
                return decenasEspeciales[n - 10];
            } else {
                int d = n / 10;
                int u = n % 10;
                if (u == 0) {
                    return dosDigitos[d];
                } else {
                    return dosDigitos[d] + " y " + unDigito[u];
                }
            }
        }
    }
```

21 The Oracle of Aetherium

8 points

Introduction

Long time ago, in the ancient city of Aetherium, an oracle devised a mysterious cipher to interpret divine messages. Legend says the oracle would take a question (as a string of text) and respond with an enigmatic numeric answer. The oracle's answers were immutable; the same question always returned the same response.

Over centuries, scholars decoded the oracle's method as a combination of text manipulation, prime numbers, and modular arithmetic. The exact rules were reconstructed to preserve the oracle's reliability and enigma.

Exercise

You are tasked to implement the Oracle of Aetherium's cipher.

Given an input string S, the oracle must compute a single numeric answer A. The oracle guarantees that:

- The same string S will always produce the same answer A
- A is always a non-negative integer.
- A depends only on the content and length of S.

Input

A single line containing a string S ($1 \leq \text{length}(S) \leq 100$, consisting of printable ASCII characters).

Output

A single integer A, the oracle's response.

The cipher's logic

The oracle computes A in the following steps:

- Normalize the string: Remove all spaces and convert all characters to lowercase.
- Compute a "hash" number: For each character in the normalized string, calculate

    hash_value=ord(c)×(position+1)

where...

- position is the 0-based index of the character.
- Sum the hash values: Compute the sum of all hash_value
- Map to a smaller number: Take the result modulo the 10,007th prime number (104,729).

Example 1

Input

```
hello world
```

Output

```
5992
```

Example 2

Input

```
this is a test
```

Output

```
7256
```

Solutions

Python

```
def oracle_of_aetherium():
    # Step 1: Read input using input()
    input_string = input().strip()

    # Step 2: Normalize the string
    normalized = input_string.replace(" ", "").lower()

    # Step 3: Compute the "hash" values
    hash_sum = sum(ord(c) * (i + 1) for i, c in enumerate(normalized))

    # Step 4: Map to a smaller number using the 10,007th prime
    result = hash_sum % 104729  # 10,007th prime number
    # Step 5: Output the result
    print(result)
# Uncomment to use:
if __name__ == "__main__":
 oracle_of_aetherium()
```

C++

```
#include <iostream>
#include <string>
#include <cctype>
using namespace std;
int main() {
    // Step 1: Read input from standard input
    string inputString;
    getline(cin, inputString);
    // Step 2: Normalize the string
    string normalized = "";
    for (char c : inputString) {
        if (!isspace(c)) {
            normalized += tolower(c);
        }
    }
    // Step 3: Compute the "hash" values
    long long hashSum = 0;
    for (size_t i = 0; i < normalized.length(); i++) {
        hashSum += static_cast<long long>(normalized[i]) * (i + 1);
    }
    // Step 4: Map to a smaller number using the 10,007th prime
    const long long PRIME = 104729; // 10,007th prime number
    long long result = hashSum % PRIME;
    // Step 5: Output the result
    cout << result << endl;
    return 0;
}
```

Java

```java
import java.util.Scanner;
class oracle {
    public static void main(String[] args) {
        // Step 1: Read input using Scanner
        Scanner scanner = new Scanner(System.in);
        String inputString = scanner.nextLine();
        scanner.close();
        // Step 2: Normalize the string
        String normalized = inputString.replace(" ", "").toLowerCase();
        // Step 3: Compute the "hash" values
        long hashSum = 0;
        for (int i = 0; i < normalized.length(); i++) {
            char c = normalized.charAt(i);
            hashSum += (long) c * (i + 1);
        }
        // Step 4: Map to a smaller number using the 10,007th prime
        long result = hashSum % 104729; // 10,007th prime number
        // Step 5: Output the result
        System.out.println(result);
    }
}
```

22 Account Security

9 points

Introduction

You have recently been hired to be part of the IT security team that manages the bank accounts of the clubs of a sports competition.

One of your first tasks is to implement a security system to ensure that only accredited users can access the information in these accounts, using a username and password system. These passwords will not be stored in plain text, but will be hashed.

A hash is the result of a function that maps data of arbitrary size to data of fixed size. The advantage of using it to store passwords is that the passwords are not stored in readable form, but rather their hash is stored. When users enter their password, their hash is calculated and checked to see if it is the same as the stored hash, thus deciding whether the password is correct or not.

The hash used to store passwords is the output of the Fowler-Noll-Vo function, in its FNV-1 variant. This hash is based on the following (pseudocode):

```
hash = FNV_offset_basis
for each byte_of_data_to_be_hashed
  hash = hash x FNV_prime
  hash = hash 22OR byte_of_data
  hash = hash % FNV_offset_basis
return hash
```

As you can see it consists of initializing the hash to a fixed number, and then, for each byte of data, this hash is multiplied by a prime number and XORing (exclusive OR) with the byte in question. The version of FNV-1 to be used is the 32-bit version, whose key values are:

- FNV_offset_basis = 2166136261
- FNV_prime = 16777619

Exercise

You are asked to develop a program that takes as standard input user and password values separated on different lines, and produces as output values of the type "*user:hash_hexadecimal*". This is because it is common to store hashes as hexadecimal sequences instead of numbers.

Hint: Use a function included in the programming language to convert a number to a hexadecimal string, do not implement the conversion yourself.

Input

The input will consist in two lines. The first line is the username and the second is the password in plain text.

For instance:

```
luis
hola
```

Output

A single line with the text:

<username>:<hash>

for instance:

```
luis:572877e9
```

Example 1

Input

```
manolito
EstaEsMiPassword
```

Output

```
manolito:123d6092
```

Example 2

Input

```
lolita
```

```
EnUnLugarDeLaMancha
```

Output

```
lolita:3240cef7
```

Solutions

Python

```
# Function to compute the FNV-1 hash
def fnv1_hash(data):
    # Initialize the hash with FNV_offset_basis
    hash = 2166136261
    offset_basis = 2166136261
    FNV_prime = 16777619

    # Process each byte of the data
    for byte in data.encode('utf-8'):  # Convert the string to bytes
        hash = hash * FNV_prime
        hash = hash ^ byte  # XOR with the byte
        hash = hash % offset_basis
#         print (int(byte), " -- ", hash)

    return hash
# Read input
username = input().strip()  # Read the username
password = input().strip()  # Read the password
# Calculate the hash
hashed_value = fnv1_hash(password)
#print(hashed_value,"\n")
# Convert the hash to a hexadecimal string
hash_hex = hex(hashed_value)[2:]  # hex() gives a string prefixed with '0x',
so we slice off the '0x'
# Print the result in the required format
print(f"{username}:{hash_hex}")
```

C++

```cpp
#include <iostream>
#include <string>
#include <sstream>
#include <iomanip>
#include <array>
using namespace std;
// Function to compute the FNV-1 hash (128-bit)
string fnv1_hash(const string& data) {
    // FNV-1 initial hash value (128-bit): 2 x 64-bit values
    //hash = 2166136261
    //FNV_prime = 16777619
    unsigned long long hash_high = 2166136261;
    const unsigned long long offset_basis = 2166136261;
    const unsigned long FNV_prime = 16777619;
    // Process each byte of the data
    for (char byte : data) {
        unsigned char byte_value = static_cast<unsigned char>(byte);
        // Apply the FNV-1 hash formula (High and Low)
        hash_high = hash_high * FNV_prime;
        hash_high = hash_high ^ byte_value;
        hash_high = hash_high % offset_basis;
//          cout << byte << " -- " << hash_high << std::endl;
    }
    // Combine the high and low 64-bit parts into a final hash
    stringstream ss;
    ss << std::hex << hash_high;
    return ss.str();
}
int main() {
    string username, password;
    // Read input
    getline(cin, username);  // Read the username
    getline(cin, password);  // Read the password
    // Calculate the hash
    string hashed_value = fnv1_hash(password);
 //   cout << hashed_value;
    // Print the result in the required format
    cout << username << ":" << hashed_value << endl;
    return 0;
}
```

Java

```
import java.util.Scanner;
public class FNV1Hash {
    // Función para calcular el hash FNV-1
    public static long fnv1Hash(String data) {
        // Inicializar el hash con FNV_offset_basis
        long hash = 2166136261L;
        long offset_basis = 2166136261L;
        int FNV_prime = 16777619;
        // Procesar cada byte de los datos
        for (int i = 0; i < data.length(); i++) {
            char byteChar = data.charAt(i);
            hash = hash * FNV_prime;
            hash = hash ^ byteChar;  // XOR con el byte
            hash = hash % offset_basis;
        }
        return hash;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Leer nombre de usuario y contraseña
        String username = scanner.nextLine().trim();  // Leer nombre de
usuario
        String password = scanner.nextLine().trim();  // Leer contraseña
        // Calcular el hash
        long hashedValue = fnv1Hash(password);
        // Convertir el hash a hexadecimal
        String hashHex = Long.toHexString(hashedValue);
        // Imprimir el resultado en el formato solicitado
        System.out.println(username + ":" + hashHex);
        scanner.close();
    }
}
```

23 The gap number

9 points

Introduction

In the bustling world of high-frequency trading, milliseconds can mean the difference between profit and loss. Traders rely on algorithms to analyze market data and make split-second decisions. One critical aspect of these algorithms is identifying gaps in stock prices to predict potential market movements. By analyzing the largest gaps between consecutive price points, traders can identify significant opportunities for buying or selling stocks. This exercise simulates a simplified version of such an analysis, focusing on finding the largest gap in a set of numbers.

Exercise

Create an array of 8 unique integers between 1 and 1000. Your task is to find the largest gap between consecutive numbers in the array. If you have 2 or more gaps with the same size, return the one with the lowest number.

Input

An array of 8 unique positive integers between 1 and 1000, separated by commas.

Output

The largest gap between consecutive numbers in the sorted array, followed by the lower and upper numbers that form this gap, in the format:

23 , Y->Z

...where X is the gap, Y is the lower number, and Z is the upper number.

In case of an array with less or more than 8 elements, return: "*Error: set exactly 8 numbers*".

In case of an array with non-unique elements, return: "*Error: no duplicate numbers allowed*".

In case of an array with numbers outside the 1-1000 range, return: "*Error: all numbers must be between 1 and 1000*".

In case of an array with non-integer elements, return: "*Error: invalid input. Make sure to enter integers separated by commas*".

Example 1

Input

```
45,123,678,234,890,567,345,789
```

Output

```
222,345->567
```

Example 2

Input

```
3,6,9,12,15,18,21,24
```

Output

```
3,3->6
```

Solutions

Python

```
def find_largest_gap(numbers):
    # Ordenar el array
    sorted_numbers = sorted(numbers)

    # Encontrar el mayor hueco entre números contiguos
    largest_gap = 0
    lower_number = 0
    upper_number = 0
    for i in range(len(sorted_numbers) - 1):
        gap = sorted_numbers[i + 1] - sorted_numbers[i]
        if gap > largest_gap:
            largest_gap = gap
            lower_number = sorted_numbers[i]
            upper_number = sorted_numbers[i + 1]

    return largest_gap, lower_number, upper_number
def main():
    # Solicitar al usuario que ingrese 8 números enteros entre 1 y 1000
    try:
        input_numbers = input()
        numbers = list(map(int, input_numbers.split(',')))

        if len(numbers) != 8:
            print("Error: set exactly 8 numbers.")
            return

        if len(set(numbers)) != 8:
            print("Error: no duplicate numbers allowed.")
            return

        if any(n < 1 or n > 1000 for n in numbers):
            print("Error: all numbers must be between 1 and 1000.")
            return

    except ValueError:
        print("Error: invalid input. Make sure to enter integers separated by 
commas.")
        return

    # Encontrar el mayor hueco
    largest_gap, lower_number, upper_number = find_largest_gap(numbers)
    print(f"{largest_gap},{lower_number}->{upper_number}")
if __name__ == "__main__":
    main()
```

C++

```
#include <iostream>
#include <vector>
#include <set>
#include <algorithm>
#include <sstream>
using namespace std;
vector<int> findLargestGap(const vector<int>& numbers) {
    vector<int> sorted_numbers = numbers;
    sort(sorted_numbers.begin(), sorted_numbers.end());

    int largest_gap = 0;
    int lower_number = 0;
    int upper_number = 0;

    for (size_t i = 0; i < sorted_numbers.size() - 1; ++i) {
        int gap = sorted_numbers[i + 1] - sorted_numbers[i];
        if (gap > largest_gap) {
            largest_gap = gap;
            lower_number = sorted_numbers[i];
            upper_number = sorted_numbers[i + 1];
        }
    }

    return {largest_gap, lower_number, upper_number};
}
int main() {
    string input;
    getline(cin, input);

    try {
        stringstream ss(input);
        string token;
        vector<int> numbers;
        set<int> number_set;

        while (getline(ss, token, ',')) {
            int number = stoi(token);
            if (number < 1 || number > 1000) {
                cout << "Error: all numbers must be between 1 and 1000." <<
endl;
                return 0;
            }
            if (!number_set.insert(number).second) {
                cout << "Error: no duplicate numbers allowed." << endl;
                return 0;
            }
            numbers.push_back(number);
        }

        if (numbers.size() != 8) {
            cout << "Error: set exactly 8 numbers." << endl;
            return 0;
        }

        vector<int> result = findLargestGap(numbers);
        cout << result[0] << "," << result[1] << "->" << result[2] << endl;

    } catch (const invalid_argument& e) {
        cout << "Error: invalid input. Make sure to enter integers separated
by commas." << endl;
        return 0;
    }

    return 0;
}
```

Java

```
import java.util.Arrays;
import java.util.HashSet;
import java.util.Scanner;
import java.util.Set;
class Main {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();
        scanner.close();
        try {
            String[] inputArray = input.split(",");
            if (inputArray.length != 8) {
                System.out.println("Error: set exactly 8 numbers.");
                return;
            }

            int[] numbers = new int[8];
            Set<Integer> numberSet = new HashSet<>();
            for (int i = 0; i < 8; i++) {
                numbers[i] = Integer.parseInt(inputArray[i].trim());
                if (numbers[i] < 1 || numbers[i] > 1000) {
                    System.out.println("Error: all numbers must be between 1
and 1000.");
                    return;
                }
                if (!numberSet.add(numbers[i])) {
                    System.out.println("Error: no duplicate numbers
allowed.");
                    return;
                }
            }

            // Encontrar el mayor hueco
            int[] result = findLargestGap(numbers);
            System.out.println(result[0] + "," + result[1] + "->" +
result[2]);

        } catch (NumberFormatException e) {
            System.out.println("Error: invalid input. Make sure to enter
integers separated by commas.");
        }
    }

    public static int[] findLargestGap(int[] numbers) {
        Arrays.sort(numbers);

        int largestGap = 0;
        int lowerNumber = 0;
        int upperNumber = 0;

        for (int i = 0; i < numbers.length - 1; i++) {
            int gap = numbers[i + 1] - numbers[i];
            if (gap > largestGap) {
                largestGap = gap;
                lowerNumber = numbers[i];
                upperNumber = numbers[i + 1];
            }
        }

        return new int[]{largestGap, lowerNumber, upperNumber};
    }
}
```

24 Maze solver

10 points

Introduction

You are an apprentice wizard, seeking to uncover the secrets of the ancient labyrinth hidden deep within the enchanted forest. The maze is not just a mere puzzle of walls and paths—it is imbued with magic. Each twist and turn might lead you closer to hidden treasures or deeper into the depths of the unknown. Your task is to navigate the maze and find the magical key that will unlock the gates to an ancient realm of wisdom.

The maze shifts with time, and its pathways are alive with an arcane energy. Only those with the intellect to decipher the paths can survive. Your journey requires you to program a spell (your maze solver) that can navigate through these mystical corridors. You need to hurry to avoid this maze change again.

Exercise

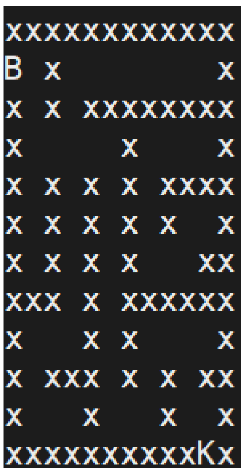
Your goal is to create a maze solver program that can find the path from the entrance (marked with letter 'K') to the exit (marked with letter 'K'), while avoiding mystical barriers.

To make this exercise easier, the maze will be only 12x12 characters, the input will always be located in the same position (1st column and 2nd row) and marked with letter 'B'. The end of the maze will always be in the same position (11th column and 12th row) and will be marked with letter 'K'.

There will be only one route to the exit.

Input

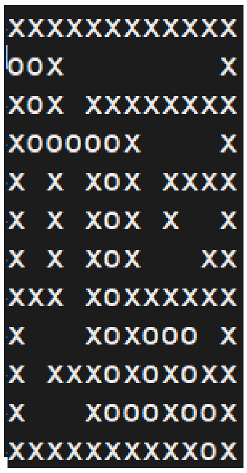
The input will be 12 lines of 12 characters. Only letters 'x' (the wall), ' ' (the corridor), 'B' (the beginning) and 'K' the exit(key of the forest) are allowed. An example of such maze would be:



```
xxxxxxxxxxxx
B x        x
x x xxxxxxxx
x     x    x
x x x x xxxx
x x x x x  x
x x x x   xx
xxx x xxxxxx
x   x x    x
x xxx x x xx
x   x   x  x
xxxxxxxxxxKx
```

Output

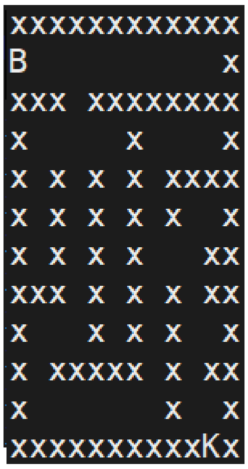
The output of the program will be the same maze with the route marked with letters 'o' (lower case o). For example:



```
xxxxxxxxxxxx
oox        x
xox xxxxxxxx
xooooox    x
x x xox xxxx
x x xox x  x
x x xox   xx
xxx xoxxxxxx
x   xoxooo x
x xxxoxoxoxx
x   xoooxoox
xxxxxxxxxxox
```

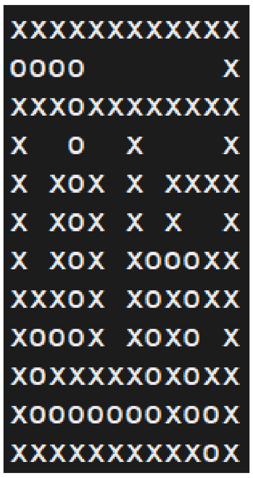
Example 1

Input



```
xxxxxxxxxxxx
B          x
xxx xxxxxxxx
x     x    x
x x x x xxxx
x x x x x  x
x x x x   xx
xxx x x x xx
x   x x x  x
x xxxxx x xx
x       x  x
xxxxxxxxxxKx
```

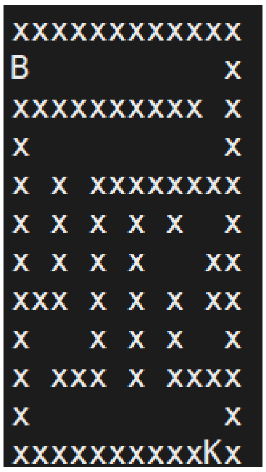
Output



```
xxxxxxxxxxxx
oooo       x
xxxoxxxxxxxx
x  o  x    x
x xox x xxxx
x xox x x  x
x xox xoooxx
xxxox xoxoxx
xooox xoxo x
xoxxxxxoxoxx
xoooooooxoox
xxxxxxxxxxox
```

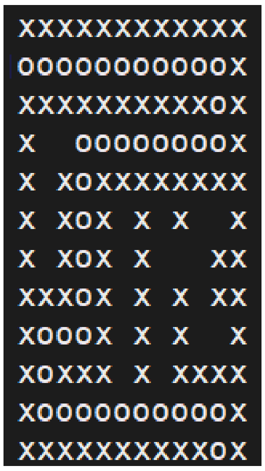
Example 2

Input



```
xxxxxxxxxxxx
B          x
xxxxxxxxxx x
x          x
x x xxxxxxxx
x x x x x  x
x x x x   xx
xxx x x x xx
x   x x x  x
x xxx x xxxx
x          x
xxxxxxxxxxKx
```

Output



```
xxxxxxxxxxxx
ooooooooooox
xxxxxxxxxxox
x  oooooooox
x xoxxxxxxxx
x xox x x  x
x xox x   xx
xxxox x x xx
xooox x x  x
xoxxx x xxxx
xoooooooooox
xxxxxxxxxxox
```

Solutions

Python

```
def read_maze():
    maze = []
    for _ in range(12):
        row = input().strip()
        maze.append(list(row))
    return maze
```

```
def print_maze(maze):
    for row in maze:
        print(''.join(row))
def is_valid_move(maze, x, y, visited):
    return 0 <= x < 12 and 0 <= y < 12 and maze[x][y] != 'x' and not
visited[x][y]
def find_path(maze, x, y, visited, path):
    if x == 11 and y == 10:  # End position (11th column, 12th row)
        path.append((x, y))
        return True
#     print ("Is valid move (",x,",",y,")? ", maze[x][y], " ---
",visited[x][y] )
    if is_valid_move(maze, x, y, visited):
#         print("Si")
        visited[x][y] = True
        path.append((x, y))
        # Move right
        if find_path(maze, x, y + 1, visited, path):
            return True
        # Move down
        if find_path(maze, x + 1, y, visited, path):
            return True
        # Move left
        if find_path(maze, x, y - 1, visited, path):
            return True
        # Move up
        if find_path(maze, x - 1, y, visited, path):
            return True
        path.pop()
        visited[x][y] = False
    return False
def main():
    maze = read_maze()
#     print_maze(maze)
    visited = [[False for _ in range(12)] for _ in range(12)]
```

```python
    path = []
    if find_path(maze, 1, 0, visited, path):
        for x, y in path:
            maze[x][y] = 'o'
        print_maze(maze)
    else:
        print("No path found")
if __name__ == "__main__":
    main()
```

C++

```cpp
#include <iostream>
#include <vector>
#include <string>
std::vector<std::vector<char>> read_maze() {
    std::vector<std::vector<char>> maze(12, std::vector<char>(12));
    for (int i = 0; i < 12; ++i) {
        std::string row;
        std::getline(std::cin, row);
        for (int j = 0; j < 12; ++j) {
            maze[i][j] = row[j];
        }
    }
    return maze;
}
void print_maze(const std::vector<std::vector<char>>& maze) {
    for (const auto& row : maze) {
        for (char cell : row) {
            std::cout << cell;
        }
        std::cout << std::endl;
    }
}
bool is_valid_move(const std::vector<std::vector<char>>& maze, int x, int y,
const std::vector<std::vector<bool>>& visited) {
    return 0 <= x && x < 12 && 0 <= y && y < 12 && maze[x][y] != 'x' &&
!visited[x][y];
}
bool find_path(std::vector<std::vector<char>>& maze, int x, int y,
std::vector<std::vector<bool>>& visited, std::vector<std::pair<int, int>>&
path) {
    if (x == 11 && y == 10) {  // End position (11th column, 12th row)
        path.emplace_back(x, y);
        return true;
    }
    if (is_valid_move(maze, x, y, visited)) {
        visited[x][y] = true;
        path.emplace_back(x, y);
        // Move right
        if (find_path(maze, x, y + 1, visited, path)) {
            return true;
        }
        // Move down
        if (find_path(maze, x + 1, y, visited, path)) {
            return true;
        }
        // Move left
        if (find_path(maze, x, y - 1, visited, path)) {
            return true;
        }
        // Move up
        if (find_path(maze, x - 1, y, visited, path)) {
            return true;
        }
        path.pop_back();
        visited[x][y] = false;
    }
    return false;
}
int main() {
    auto maze = read_maze();
    std::vector<std::vector<bool>> visited(12, std::vector<bool>(12, false));
    std::vector<std::pair<int, int>> path;
    if (find_path(maze, 1, 0, visited, path)) {
        for (const auto& [x, y] : path) {
            maze[x][y] = 'o';
        }
        print_maze(maze);
    } else {
        std::cout << "No path found" << std::endl;
    }
    return 0;
}
```

Java

```
import java.util.*;
public class MazeSolver {
    public static char[][] readMaze() {
        Scanner scanner = new Scanner(System.in);
        char[][] maze = new char[12][12];
        for (int i = 0; i < 12; i++) {
            String row = scanner.nextLine().trim();
            for (int j = 0; j < 12; j++) {
                maze[i][j] = row.charAt(j);
            }
        }
        return maze;
    }
    public static void printMaze(char[][] maze) {
        for (char[] row : maze) {
            for (char cell : row) {
                System.out.print(cell);
            }
            System.out.println();
        }
    }
    public static boolean isValidMove(char[][] maze, int x, int y,
boolean[][] visited) {
        return 0 <= x && x < 12 && 0 <= y && y < 12 && maze[x][y] != 'x' &&
!visited[x][y];
    }
    public static boolean findPath(char[][] maze, int x, int y, boolean[][]
visited, List<int[]> path) {
        if (x == 11 && y == 10) {  // End position (11th column, 12th row)
            path.add(new int[]{x, y});
            return true;
        }
        if (isValidMove(maze, x, y, visited)) {
            visited[x][y] = true;
            path.add(new int[]{x, y});
            // Move right
            if (findPath(maze, x, y + 1, visited, path)) {
                return true;
            }
            // Move down
            if (findPath(maze, x + 1, y, visited, path)) {
                return true;
            }
            // Move left
            if (findPath(maze, x, y - 1, visited, path)) {
                return true;
            }
            // Move up
            if (findPath(maze, x - 1, y, visited, path)) {
                return true;
            }
            path.remove(path.size() - 1);
            visited[x][y] = false;
        }
        return false;
    }
    public static void main(String[] args) {
        char[][] maze = readMaze();
        boolean[][] visited = new boolean[12][12];
        List<int[]> path = new ArrayList<>();
        if (findPath(maze, 1, 0, visited, path)) {
            for (int[] pos : path) {
                maze[pos[0]][pos[1]] = 'o';
            }
            printMaze(maze);
        } else {
            System.out.println("No path found");
        }
    }
}
```

25 Our best defense

11 points

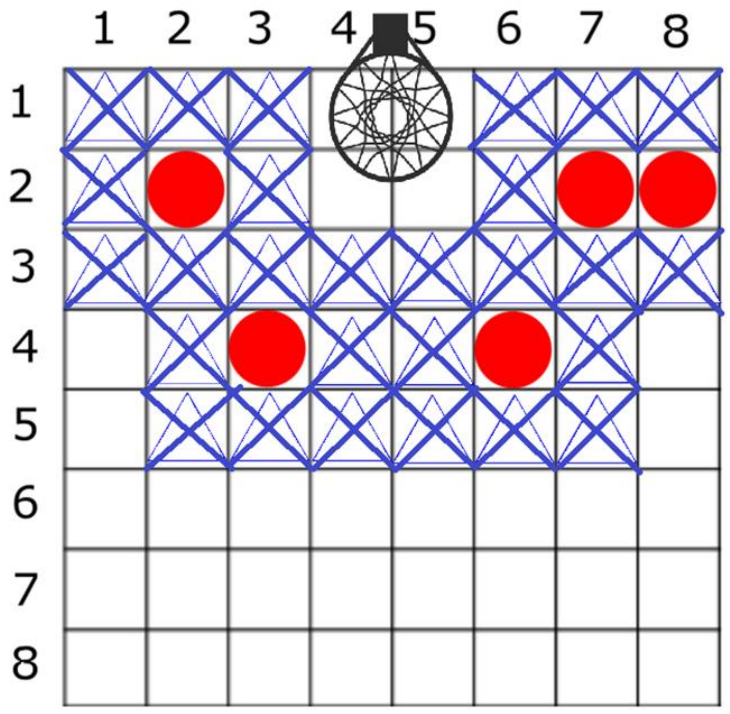
Introduction

The latest trends in basketball coaching have determined the defensive level of a team based on a simple algorithm. It has been calculated that the length of a player's arms corresponds to 1/8 of the width of the court. Therefore, a player is able to defend up to 3/8 of the width of the court (both arms and body).

In this way half a basketball court could be divided into a 8x8 grid, and a player can cover a maximum of 9 squares of that grid. At the edges the player will not be able to defend all 9 squares, or if there is a player next to him, he will be in the way.

For example if we place the defensive players in the red positions, the part of the court defended will be the

the defended part of the court will be the one grated in blue:



Exercise

You are asked to develop a program that allows you to evaluate the defensive level of your team according to the position of the 5 players on the court and the total percentage of the court occupied, according to this table:

```
Percentage Defensive Level
    > 70%   VERY HIGH
    > 50%   HIGH
    > 25%   LOW
   <= 25%   VERY LOW
```

In the example, there are a total of 32 defended squares (the player himself defends his position). That is 32 out of 64 (50%) that would be considered a LOW level of defense (note that HIGH is ABOVE 50%).

Note: The basket shown in the illustrative image is not actually occupying the cells, and these cells should be evaluated like any other.

Input

There will always be 5 players, so the input consists of 5 lines corresponding to the position of each player on the court previously represented. For example for the above drawing it will be:

2,2

2,7

2,8

4,3

4,6

From this entry you have to calculate all the positions that are covered, i.e. where the player is (red dot) and those that are covered (blue crosses).

To calculate the % of the field covered, you have to add up all the positions with red dots and blue crosses (in this case 32) and divide them by the total number of squares of the field (56). 32/64 is 0.5: 50%.

Output

The output will be a single line with the team's defensive level:

[LEVEL NUMBER]%: [VERY LOW|LOW|HIGH|VERY HIGH]

For example:

50%: LOW

Example 1

Input

```
2,2
2,7
2,8
4,3
4,6
```

Output

```
50%: LOW
```

Example 2

Input

```
2,2
2,7
5,2
5,5
7,7
```

Output

```
69%: HIGH
```

Solutions

Python

```
def get_covered_positions(player_positions):
    covered_positions = set()
    directions = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 0), (0, 1), (1, -
1), (1, 0), (1, 1)]
```

```
    for x, y in player_positions:
        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if 1 <= nx <= 8 and 1 <= ny <= 8:
                covered_positions.add((nx, ny))
    return covered_positions
def calculate_defensive_level(covered_positions):
    total_positions = 64
    covered_count = len(covered_positions)
    coverage_percentage = (covered_count / total_positions) * 100
    if coverage_percentage > 70:
        level = "VERY HIGH"
    elif coverage_percentage > 50:
        level = "HIGH"
    elif coverage_percentage > 25:
        level = "LOW"
    else:
        level = "VERY LOW"
    return coverage_percentage, level
def main():
    player_positions = []
    for _ in range(5):
        x, y = map(int, input().strip().split(','))
        player_positions.append((x, y))
    covered_positions = get_covered_positions(player_positions)
    coverage_percentage, level = calculate_defensive_level(covered_positions)
    print(f"{coverage_percentage:.0f}%: {level}")
if __name__ == "__main__":
    main()
```

C++

```
#include <iostream>
#include <vector>
#include <set>
#include <string>
```

```
#include <sstream>
#include <iomanip>
std::set<std::pair<int, int>> get_covered_positions(const
std::vector<std::pair<int, int>>& player_positions) {
    std::set<std::pair<int, int>> covered_positions;
    std::vector<std::pair<int, int>> directions = {{-1, -1}, {-1, 0}, {-1,
1}, {0, -1}, {0, 0}, {0, 1}, {1, -1}, {1, 0}, {1, 1}};
    for (const auto& pos : player_positions) {
        int x = pos.first;
        int y = pos.second;
        for (const auto& dir : directions) {
            int nx = x + dir.first;
            int ny = y + dir.second;
            if (1 <= nx && nx <= 8 && 1 <= ny && ny <= 8) {
                covered_positions.emplace(nx, ny);
            }
        }
    }
    return covered_positions;
}
std::pair<double, std::string> calculate_defensive_level(const
std::set<std::pair<int, int>>& covered_positions) {
    int total_positions = 64;
    int covered_count = covered_positions.size();
    double coverage_percentage = (static_cast<double>(covered_count) /
total_positions) * 100;
    std::string level;
    if (coverage_percentage > 70) {
        level = "VERY HIGH";
    } else if (coverage_percentage > 50) {
        level = "HIGH";
    } else if (coverage_percentage > 25) {
        level = "LOW";
    } else {
        level = "VERY LOW";
    }
    return {coverage_percentage, level};
}
int main() {
    std::vector<std::pair<int, int>> player_positions;
    for (int i = 0; i < 5; ++i) {
        std::string input;
        std::getline(std::cin, input);
        std::istringstream iss(input);
        int x, y;
        char comma;
        iss >> x >> comma >> y;
        player_positions.emplace_back(x, y);
    }
    auto covered_positions = get_covered_positions(player_positions);
    auto [coverage_percentage, level] =
calculate_defensive_level(covered_positions);
    std::cout << std::fixed << std::setprecision(0) << coverage_percentage <<
"%: " << level << std::endl;
    return 0;
}
```

Java

```
import java.util.*;
public class OurBestDefense {
    public static Set<Map.Entry<Integer, Integer>>
getCoveredPositions(List<Map.Entry<Integer, Integer>> playerPositions) {
        Set<Map.Entry<Integer, Integer>> coveredPositions = new HashSet<>();
        List<Map.Entry<Integer, Integer>> directions = Arrays.asList(
            new AbstractMap.SimpleEntry<>(-1, -1), new
AbstractMap.SimpleEntry<>(-1, 0), new AbstractMap.SimpleEntry<>(-1, 1),
            new AbstractMap.SimpleEntry<>(0, -1), new
AbstractMap.SimpleEntry<>(0, 0), new AbstractMap.SimpleEntry<>(0, 1),
            new AbstractMap.SimpleEntry<>(1, -1), new
AbstractMap.SimpleEntry<>(1, 0), new AbstractMap.SimpleEntry<>(1, 1)
```

```
        );
        for (Map.Entry<Integer, Integer> pos : playerPositions) {
            int x = pos.getKey();
            int y = pos.getValue();
            for (Map.Entry<Integer, Integer> dir : directions) {
                int nx = x + dir.getKey();
                int ny = y + dir.getValue();
                if (1 <= nx && nx <= 8 && 1 <= ny && ny <= 8) {
                    coveredPositions.add(new AbstractMap.SimpleEntry<>(nx,
ny));
                }
            }
        }
        return coveredPositions;
    }
    public static Map.Entry<Double, String>
calculateDefensiveLevel(Set<Map.Entry<Integer, Integer>> coveredPositions) {
        int totalPositions = 64;
        int coveredCount = coveredPositions.size();
        double coveragePercentage = (double) coveredCount / totalPositions *
100;
        String level;
        if (coveragePercentage > 70) {
            level = "VERY HIGH";
        } else if (coveragePercentage > 50) {
            level = "HIGH";
        } else if (coveragePercentage > 25) {
            level = "LOW";
        } else {
            level = "VERY LOW";
        }
        return new AbstractMap.SimpleEntry<>(coveragePercentage, level);
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<Map.Entry<Integer, Integer>> playerPositions = new
ArrayList<>();
        for (int i = 0; i < 5; ++i) {
            String input = scanner.nextLine();
            String[] parts = input.split(",");
            int x = Integer.parseInt(parts[0].trim());
            int y = Integer.parseInt(parts[1].trim());
            playerPositions.add(new AbstractMap.SimpleEntry<>(x, y));
        }
        Set<Map.Entry<Integer, Integer>> coveredPositions =
getCoveredPositions(playerPositions);
        Map.Entry<Double, String> result =
calculateDefensiveLevel(coveredPositions);
        System.out.printf("%.0f%%: %s%n", result.getKey(),
result.getValue());
    }
}
```

26 The conversation

11 points

Introduction

Two school friends having a coffee:

- **Jose**: Do you remember that time when we were studying powers, and the math teacher told us: "I'll give you the whole afternoon off… if you can line up in such a way that the sum of each of your class numbers with the next one in the line equals a perfect square (except for the last one, of course, since they have no one behind them to add to)"?

- **Mario**: We didn't understand anything, did we?

- **Jose**: Yes... – you remember, he told us – "you have to line up all 15 of you in a specific order. For example, if the first is Pablo Garcia, who is number 9, the next could be Alba Fernandez, who is number 7, because 9 + 7 is 16, which is a perfect square. And after Alba, who is 7, could be Juan Alvarez, number 2, because 7 + 2 is 9, which is also a perfect square, and so on, without missing anyone. Can you do it?"

- **Mario**: Wow! The chaos that ensued! Everyone trying to organize the others. What a mess...

- **Jose**: Thank God that Cholo is a genius and started programming something on his laptop. It didn't take him 5 minutes.

- **Mario**: Oh, yeah, he explained to me later: he wrote a function that, given an integer N, returned an array of integers from 1 to N, ordered so that the sum of each pair of consecutive numbers was a perfect square. The solution was valid if and only if two conditions were met:

  - Each number in the range 1..N was used exactly once.

  - The sum of each pair of consecutive numbers was a perfect square.

- **Jose**: And I still remember the order in which he told us to line up according to that program. Since there were 15 of us, the order was: [9, 7, 2, 14, 11, 5, 4, 12, 13, 3, 6, 10, 15, 1, 8]

- **Mario**: What was coolest about his program is that it worked for any number of students (up to 64). You simply wrote the number of students in the class... and voila! If there was no solution, it simply returned "ERROR," and if there were multiple solutions, it only returned the one starting with the highest number.

- **Jose**: So,in the program, you'd to input a number (and nothing else), and it would return either "ERROR" or the solution?

- **Mario**: Exactly!!!

- **Jose**: These computer scientists are amazing...

Exercise

Could you mimic Cholo's code?

Input

The input will consist in a single integer positive number (N). For instance:

```
25
```

Output

The output will be a list of single-space-separated numbers from 1 to N in the appropiate order indicated by the problem. For instance:

```
23 2 14 22 3 13 12 4 21 15 10 6 19 17 8 1 24 25 11 5 20 16 9 7 18
```

Because 23+2 is 25, then 2+14 is 16, then 14 + 22 is 36...

If it's not possible to obtain the list, write the word:

```
ERROR
```

Example 1

Input

```
16
```

Output

```
16 9 7 2 14 11 5 4 12 13 3 6 10 15 1 8
```

Example 2

Input

```
15
```

Output

```
9 7 2 14 11 5 4 12 13 3 6 10 15 1 8
```

Example 3

Input

```
1
```

Output

```
1
```

Example 4

Input

```
4
```

Output

```
ERROR
```

Solutions

Python

```
import math


def es_cuadrado_perfecto(n):
    raiz = int(math.sqrt(n))
    return raiz * raiz == n

def generar_cuadrados_hasta(n):
    cuadrados = set()
    i = 1
    while i * i <= n:
        cuadrados.add(i * i)
        i += 1
    return cuadrados

def backtrack(solucion, visitados, N, cuadrados):
    if len(solucion) == N:
        return solucion
    ultimo = solucion[-1]
    for siguiente in range(1, N + 1):
        if siguiente not in visitados and (ultimo + siguiente) in cuadrados:
            visitados.add(siguiente)
            solucion.append(siguiente)
            resultado = backtrack(solucion, visitados, N, cuadrados)
            if resultado:
                return resultado
            solucion.pop()
            visitados.remove(siguiente)
    return None

def ordenar_amigos(N):
    if N == 1:
        return [1]
    cuadrados = generar_cuadrados_hasta(2 * N)
    for inicio in range(N, 0, -1):
        solucion = [inicio]
        visitados = {inicio}
        resultado = backtrack(solucion, visitados, N, cuadrados)
        if resultado:
            return resultado
    return "ERROR"

# Ejemplo de uso
N = int(input())
resultado = ordenar_amigos(N)
if resultado == "ERROR":
    print("ERROR")
else:
    print(" ".join(map(str, resultado)))
```

C++

```
#include <iostream>
#include <vector>
#include <unordered_set>
#include <cmath>
bool esCuadradoPerfecto(int n) {
    int raiz = static_cast<int>(std::sqrt(n));
    return raiz * raiz == n;
}
std::unordered_set<int> generarCuadradosHasta(int n) {
    std::unordered_set<int> cuadrados;
    int i = 1;
    while (i * i <= n) {
        cuadrados.insert(i * i);
        ++i;
    }
    return cuadrados;
}
bool backtrack(std::vector<int>& solucion, std::unordered_set<int>&
visitados, int N, const std::unordered_set<int>& cuadrados) {
    if (solucion.size() == N) {
        return true;
    }
    int ultimo = solucion.back();
    for (int siguiente = 1; siguiente <= N; ++siguiente) {
        if (visitados.find(siguiente) == visitados.end() &&
cuadrados.find(ultimo + siguiente) != cuadrados.end()) {
            visitados.insert(siguiente);
            solucion.push_back(siguiente);
            if (backtrack(solucion, visitados, N, cuadrados)) {
                return true;
            }
            solucion.pop_back();
            visitados.erase(siguiente);
        }
    }
    return false;
}
std::vector<int> ordenarAmigos(int N) {
    if (N == 1) {
        return {1};
    }
    std::unordered_set<int> cuadrados = generarCuadradosHasta(2 * N);
    for (int inicio = N; inicio > 0; --inicio) {
        std::vector<int> solucion = {inicio};
        std::unordered_set<int> visitados = {inicio};
        if (backtrack(solucion, visitados, N, cuadrados)) {
            return solucion;
        }
    }
    return {};
}
int main() {
    int N;
    std::cin >> N;
    std::vector<int> resultado = ordenarAmigos(N);
    if (resultado.empty()) {
        std::cout << "ERROR" << std::endl;
    } else {
        for (size_t i = 0; i < resultado.size(); ++i) {
            std::cout << resultado[i];
            if (i < resultado.size() - 1) {
                std::cout << " "; // Imprimir espacio solo si no es el último
número
            }
        }
        std::cout << std::endl;
    }
    return 0;
}
```

Java

```
import java.util.*;
public class OrdenarAmigos {
    public static boolean esCuadradoPerfecto(int n) {
        int raiz = (int) Math.sqrt(n);
        return raiz * raiz == n;
    }
    public static Set<Integer> generarCuadradosHasta(int n) {
        Set<Integer> cuadrados = new HashSet<>();
        int i = 1;
        while (i * i <= n) {
            cuadrados.add(i * i);
            i++;
        }
        return cuadrados;
    }
    public static boolean backtrack(List<Integer> solucion, Set<Integer>
visitados, int N, Set<Integer> cuadrados) {
        if (solucion.size() == N) {
            return true;
        }
        int ultimo = solucion.get(solucion.size() - 1);
        for (int siguiente = 1; siguiente <= N; siguiente++) {
            if (!visitados.contains(siguiente) && cuadrados.contains(ultimo +
siguiente)) {
                visitados.add(siguiente);
                solucion.add(siguiente);
                if (backtrack(solucion, visitados, N, cuadrados)) {
                    return true;
                }
                solucion.remove(solucion.size() - 1);
                visitados.remove(siguiente);
            }
        }
        return false;
    }
    public static List<Integer> ordenarAmigos(int N) {
        if (N == 1) {
            return Collections.singletonList(1);
        }
        Set<Integer> cuadrados = generarCuadradosHasta(2 * N);
        for (int inicio = N; inicio > 0; inicio--) {
            List<Integer> solucion = new ArrayList<>();
            solucion.add(inicio);
            Set<Integer> visitados = new HashSet<>();
            visitados.add(inicio);
            if (backtrack(solucion, visitados, N, cuadrados)) {
                return solucion;
            }
        }
        return Collections.emptyList();
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int N = scanner.nextInt();
        List<Integer> resultado = ordenarAmigos(N);
        if (resultado.isEmpty()) {
            System.out.println("ERROR");
        } else {
            int ii=1;
            for (int num : resultado) {
                //System.out.print(num + " ");
                if (ii ==resultado.size()) {
                   System.out.println(num);
                } else {
                   System.out.printf("%d ", num);
                   ii++;
                }
            }
//            System.out.printf("\n");
        }
        scanner.close();
    }
}
```

27 The villains league

11 points

Introduction

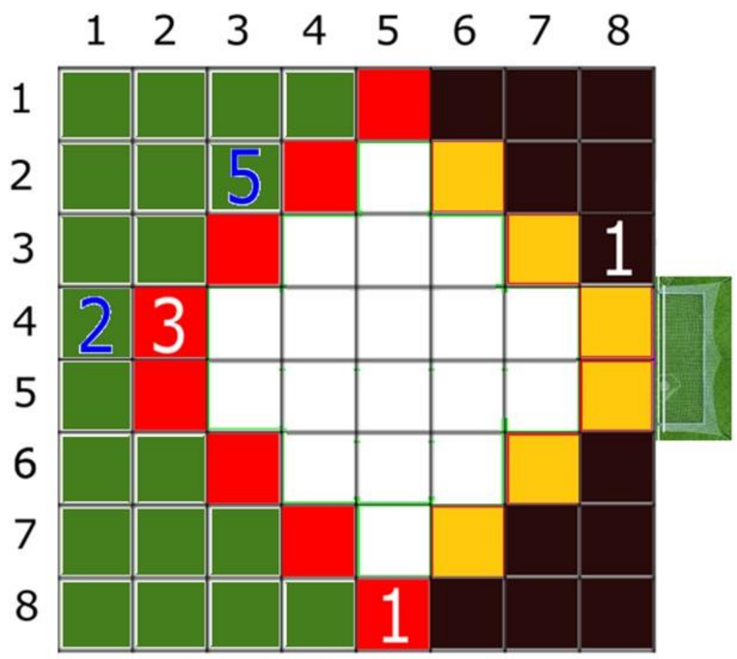
A new sport (the Villain's League) has just been invented. It is played with the foot, like soccer, but it has completely innovative rules.

There are goals, but no goalkeepers, which is why more goals are scored. Furthermore, the field's floor is colored in such a way that depending on the zone from which a goal is scored, a different point value is awarded.

The rules state that:

- Half of the field in the Villain's League is divided into an 8x8 grid.
- Goals from the green zone are worth 3 points.
- Goals from the red zone are worth 2.5 points.
- Goals from the white zone are worth 2 points.
- Goals from the yellow zone are worth 1 point.
- Goals from the black zone do not add any points to the score.

See the following image:



The numbers displayed on the grid correspond to the number of goals scored from that square.

Exercise

You are asked to develop a program that calculates the total number of points scored by a team.

Input

The input corresponds to the position on the field from where the goals were scored (row and column separated by a colon ":") and the number of goals scored:

Row:Column:NumberOfGoals,Row2:Column2:Goals2, ...

For instance:

```
2:3:5,1:4:2,2:4:3,3:8:1,8:5:1
```

Output

The output is simply the total number of points for the team:

The output should be displayed as follows:

PUNTOS: TotalPoints

For instance:

```
PUNTOS: 31
```

Note: These inputs and outputs values correspond to the data represented in the field-board diagram.

Example 1

Input

```
2:3:5,1:4:2,2:4:3,3:8:1,8:5:1
```

Output

```
PUNTOS: 31
```

Example 2

Input

```
1:2:3,4:5:6,2:3:4,5:6:7,3:4:5
```

Output

```
PUNTOS: 57
```

Example 3

Input

```
1:2:3,3:2:1,3:3:3,2:2:2,1:1:1
```

Output

```
PUNTOS: 28
```

Solutions

Python

```
# Define the field map with color zones
field_map = [
    ['V', 'V', 'V', 'V', 'R', 'N', 'N', 'N'],
    ['V', 'V', 'V', 'R', 'B', 'A', 'N', 'N'],
    ['V', 'V', 'R', 'B', 'B', 'B', 'A', 'N'],
    ['V', 'R', 'B', 'B', 'B', 'B', 'B', 'A'],
    ['V', 'R', 'B', 'B', 'B', 'B', 'B', 'A'],
    ['V', 'V', 'R', 'B', 'B', 'B', 'A', 'N'],
    ['V', 'V', 'V', 'R', 'B', 'A', 'N', 'N'],
    ['V', 'V', 'V', 'V', 'R', 'N', 'N', 'N']
]
# Points map for each zone (corrected values)
points_map = {
    'V': 3.0,  # Green zone
    'R': 2.5,  # Red zone
    'B': 2.0,  # White zone
    'A': 1.0,  # Yellow zone
    'N': 0.0   # Black zone
}
# Function to calculate the total points from the input
def calculate_total_points(input_str):
    total_points = 0.0
    # Split the input into goal descriptions
    goals = input_str.split(',')
    for goal in goals:
        row, col, num_goals = map(int, goal.split(':'))
        if row < 1 or row > 8 or col < 1 or col > 8 or num_goals < 0:
            print("Invalid input: coordinates or number of goals out of
range!")
            return 0
        row -= 1
        col -= 1
        zone = field_map[row][col]
        points = points_map[zone]
        total_points += points * num_goals
    return total_points
# Main function
if __name__ == "__main__":
    input_str = input().strip()
    total_points = calculate_total_points(input_str)
    print(f"PUNTOS: {int(total_points)}")
```

C++

```
#include <iostream>
#include <string>
#include <map>
#include <sstream>
#include <vector>
using namespace std;
// Define the field map with color zones
char field_map[8][8] = {
    {'V', 'V', 'V', 'V', 'R', 'N', 'N', 'N'},
    {'V', 'V', 'V', 'R', 'B', 'A', 'N', 'N'},
    {'V', 'V', 'R', 'B', 'B', 'B', 'A', 'N'},
    {'V', 'R', 'B', 'B', 'B', 'B', 'B', 'A'},
    {'V', 'R', 'B', 'B', 'B', 'B', 'B', 'A'},
    {'V', 'V', 'R', 'B', 'B', 'B', 'A', 'N'},
    {'V', 'V', 'V', 'R', 'B', 'A', 'N', 'N'},
    {'V', 'V', 'V', 'V', 'R', 'N', 'N', 'N'}
};
// Points map for each zone (corrected values)
map<char, double> points_map = {
    {'V', 3.0},  // Green zone
    {'R', 2.5},  // Red zone
    {'B', 2.0},  // White zone
    {'A', 1.0},  // Yellow zone
    {'N', 0.0}   // Black zone
};
// Function to calculate the total points from the input
double calculate_total_points(const string& input_str) {
    double total_points = 0.0;
    stringstream ss(input_str);
    string goal;
    // Split the input into goal descriptions
    while (getline(ss, goal, ',')) {
        int row, col, num_goals;
        sscanf(goal.c_str(), "%d:%d:%d", &row, &col, &num_goals);
        if (row < 1 || row > 8 || col < 1 || col > 8 || num_goals < 0) {
            cout << "Invalid input: coordinates or number of goals out of
range!" << endl;
            return 0;
        }
        row -= 1;
        col -= 1;
        char zone = field_map[row][col];
        double points = points_map[zone];
        total_points += points * num_goals;
    }
    return total_points;
}
int main() {
    string input_str;
    getline(cin, input_str);
    double total_points = calculate_total_points(input_str);
    cout << "PUNTOS: " << static_cast<int>(total_points) << endl;
    return 0;
}
```

Java

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
public class Main {
    // Define the field map with color zones
    static char[][] fieldMap = {
        {'V', 'V', 'V', 'V', 'R', 'N', 'N', 'N'},
        {'V', 'V', 'V', 'R', 'B', 'A', 'N', 'N'},
        {'V', 'V', 'R', 'B', 'B', 'B', 'A', 'N'},
        {'V', 'R', 'B', 'B', 'B', 'B', 'B', 'A'},
        {'V', 'R', 'B', 'B', 'B', 'B', 'B', 'A'},
        {'V', 'V', 'R', 'B', 'B', 'B', 'A', 'N'},
        {'V', 'V', 'V', 'R', 'B', 'A', 'N', 'N'},
        {'V', 'V', 'V', 'V', 'R', 'N', 'N', 'N'}
    };
    // Points map for each zone (corrected values)
    static Map<Character, Double> pointsMap = new HashMap<>();
    static {
        pointsMap.put('V', 3.0);  // Green zone
        pointsMap.put('R', 2.5);  // Red zone
        pointsMap.put('B', 2.0);  // White zone
        pointsMap.put('A', 1.0);  // Yellow zone
        pointsMap.put('N', 0.0);  // Black zone
    }
    // Function to calculate the total points from the input
    public static double calculateTotalPoints(String inputStr) {
        double totalPoints = 0.0;
        // Split the input into goal descriptions
        String[] goals = inputStr.split(",");
        for (String goal : goals) {
            String[] parts = goal.split(":");
            int row = Integer.parseInt(parts[0]);
            int col = Integer.parseInt(parts[1]);
            int numGoals = Integer.parseInt(parts[2]);
            if (row < 1 || row > 8 || col < 1 || col > 8 || numGoals < 0) {
                System.out.println("Invalid input: coordinates or number of
goals out of range!");
                return 0;
            }
            row -= 1;
            col -= 1;
            char zone = fieldMap[row][col];
            double points = pointsMap.get(zone);
            totalPoints += points * numGoals;
        }
        return totalPoints;
    }
    // Main function
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String inputStr = scanner.nextLine().trim();
        double totalPoints = calculateTotalPoints(inputStr);
        System.out.println("PUNTOS: " + (int) totalPoints);
    }
}
```

28 Toy Workshop

12 points

Introduction

You work in a toy workshop, and your job is to build toy cars and toy bikes from a collection of parts. Each toy car needs 4 car wheels, 1 car body, and 2 figures of people to be placed inside and each toy bike needs 2 bike wheels, 1 bike body and 1 figures of people to be placed inside.

It is important to build in this order: one car, one bike, one car, one bike… Be careful because if you don't have pieces to build a car, you can continue building a bike or the other way around (if you don't have pieces to build a bike, you can continue building a car).

Given the total number of car wheels, car bodies, bike wheels, bike bodies, and figures available, how many complete toy cars can you make?

Exercise

Write a program that receives as input all available parts and calculate how many cars and bikes can be built.

Input will be given with CSV format in this order: car wheels, car bodies, bike wheels, figures of people.

Input

```
20,10,3,10,7
```

...that is, 20 car wheels, 10 car bodies, 3 bike wheels, 10 bike bodies and 7 figures of people

Output

```
Cars[3] Bikes[1]
```

...that means: Total number of cars built: 3 - Total number of bikes built: 1

Example 1

Input

```
20,10,3,10,7
```

Output

```
Cars[3] Bikes[1]
```

Solutions

Python

```
# Clase para almacenar las piezas
class Pieces:
    def __init__(self, car_wheels=0, car_bodies=0, bike_wheels=0,
bike_bodies=0, figures=0):
        self.car_wheels = car_wheels
        self.car_bodies = car_bodies
        self.bike_wheels = bike_wheels
        self.bike_bodies = bike_bodies
        self.figures = figures
# Función para analizar la entrada y devolver una instancia de la clase
Pieces
def parse_input(line):
    try:
        values = list(map(int, line.split(',')))
        if len(values) != 5:
            raise ValueError("Input does not contain exactly 5 values.")
        return Pieces(*values)
    except ValueError as e:
        print(f"Error: {e}")
        sys.exit(1)
# Función para intentar construir un auto
def build_car(pieces):
    if pieces.car_bodies >= 1 and pieces.car_wheels >= 4 and pieces.figures
>= 2:
        pieces.car_bodies -= 1
        pieces.car_wheels -= 4
        pieces.figures -= 2
        return True
    return False
# Función para intentar construir una bicicleta
```

```python
def build_bike(pieces):
    if pieces.bike_bodies >= 1 and pieces.bike_wheels >= 2 and pieces.figures
>= 1:
        pieces.bike_bodies -= 1
        pieces.bike_wheels -= 2
        pieces.figures -= 1
        return True
    return False
def main():
    line = input().strip()
    pieces = parse_input(line)
    cars_built = 0
    bikes_built = 0
    while True:
        built_something = False
        if build_car(pieces):
            cars_built += 1
            built_something = True
        if build_bike(pieces):
            bikes_built += 1
            built_something = True
        if not built_something:
            break
    print("Cars[",cars_built,"] Bikes[",bikes_built,"]", sep='')
if __name__ == "__main__":
    main()
```

C++

```cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
using namespace std;
// Struct of pieces
struct Pieces
```

```
{
    int car_wheels{ 0 };
    int car_bodies{ 0 };
    int bike_wheels{ 0 };
    int bike_bodies{ 0 };
    int figures{ 0 };
};
// Parse the input data and return an instance of the struct
Pieces parse_input(const std::string& line) {
    std::stringstream ss(line);
    Pieces pieces;
    char delimiter = ',';
    std::string temp;
    int values[5];
    int count = 0;
    while (std::getline(ss, temp, delimiter)) {
        try {
            values[count] = std::stoi(temp);
        }
        catch (const std::exception&) {
            cerr << "Error: Input contains non-integer values." << endl;
            std::exit(EXIT_FAILURE);
        }
        count++;
    }
    if (count != 5) {
        cerr << "Error: Input does not contain exactly 5 values." << endl;
        std::exit(EXIT_FAILURE);
    }
    pieces.car_wheels = values[0];
    pieces.car_bodies = values[1];
    pieces.bike_wheels = values[2];
    pieces.bike_bodies = values[3];
    pieces.figures = values[4];
    return pieces;
```

```
}
// Try to build a car and return true or false if we have enough pieces
bool build_car(Pieces* pieces)
{
    if (pieces->car_bodies >= 1 && pieces->car_wheels >= 4 && pieces->figures
>= 2)
    {
        pieces->car_bodies -= 1;
        pieces->car_wheels -= 4;
        pieces->figures -= 2;
        return true;
    }
    return false;
}
// Try to build a bike and return true or false if we have enough pieces
bool build_bike(Pieces* pieces)
{
    if (pieces->bike_bodies >= 1 && pieces->bike_wheels >= 2 && pieces-
>figures >= 1)
    {
        pieces->bike_bodies -= 1;
        pieces->bike_wheels -= 2;
        pieces->figures -= 1;
        return true;
    }
    return false;
}
int main() {
    string line;

    getline(cin, line);
    Pieces pieces = parse_input(line);
    int cars_built = 0;
    int bikes_built = 0;
    bool built_something = false;
    do
    {
        built_something = false;
        if (build_car(&pieces)) {
            cars_built++;
            built_something = true;
        }
        if (build_bike(&pieces)) {
            bikes_built++;
            built_something = true;
        }
    } while (built_something);
    cout << "Cars[" << cars_built << "] Bikes[" << bikes_built << "]" <<
endl;
    return 0;
}
```

Java

```
import java.util.Scanner;
class Pieces {
    int carWheels;
    int carBodies;
    int bikeWheels;
    int bikeBodies;
    int figures;
    public Pieces(int carWheels, int carBodies, int bikeWheels, int
bikeBodies, int figures) {
        this.carWheels = carWheels;
        this.carBodies = carBodies;
        this.bikeWheels = bikeWheels;
        this.bikeBodies = bikeBodies;
        this.figures = figures;
    }
}
public class toyworkshop {
    // Método para analizar la entrada y devolver una instancia de Pieces
    public static Pieces parseInput(String line) {
        try {
            String[] parts = line.split(",");
            if (parts.length != 5) {
                throw new IllegalArgumentException("Input does not contain
exactly 5 values.");
            }
            int carWheels = Integer.parseInt(parts[0]);
            int carBodies = Integer.parseInt(parts[1]);
            int bikeWheels = Integer.parseInt(parts[2]);
            int bikeBodies = Integer.parseInt(parts[3]);
            int figures = Integer.parseInt(parts[4]);
            return new Pieces(carWheels, carBodies, bikeWheels, bikeBodies,
figures);
        } catch (NumberFormatException e) {
            System.err.println("Error: Input contains non-integer values.");
            System.exit(1);
        } catch (IllegalArgumentException e) {
            System.err.println("Error: " + e.getMessage());
            System.exit(1);
        }
        return null; // Este punto nunca se alcanzará debido al System.exit
    }
    // Método para intentar construir un auto
    public static boolean buildCar(Pieces pieces) {
        if (pieces.carBodies >= 1 && pieces.carWheels >= 4 && pieces.figures
>= 2) {
            pieces.carBodies -= 1;
            pieces.carWheels -= 4;
            pieces.figures -= 2;
            return true;
        }
        return false;
    }
    // Método para intentar construir una bicicleta
    public static boolean buildBike(Pieces pieces) {
        if (pieces.bikeBodies >= 1 && pieces.bikeWheels >= 2 &&
pieces.figures >= 1) {
            pieces.bikeBodies -= 1;
            pieces.bikeWheels -= 2;
            pieces.figures -= 1;
            return true;
        }
        return false;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String line = scanner.nextLine().trim();
        Pieces pieces = parseInput(line);
        int carsBuilt = 0;
        int bikesBuilt = 0;
        while (true) {
            boolean builtSomething = false;
            if (buildCar(pieces)) {
                carsBuilt++;
                builtSomething = true;
            }
            if (buildBike(pieces)) {
                bikesBuilt++;
                builtSomething = true;
            }
            if (!builtSomething) {
                break;
            }
        }
        // Imprimir el resultado
        System.out.println("Cars[" + carsBuilt + "] Bikes[" + bikesBuilt +
"]");
        scanner.close();
        }
    }
```

29 Latin square solver

13 points

Introduction

A Latin square is an n×n matrix filled with n different symbols, each of which appears exactly once in each row and exactly once in each column. For example...

```
A B C
C A B
B C A
```

You can see that latin squares are the building block of the sudoku games.

Exercise

You are asked to write a program that reads the top row of a Latin square in one line, with an array representation of a Latin square of arbitrary order n. For example, the input can be:

```
123
```

or

```
UYTFGE
```

or

```
Q2D89L6F
```

...we'll consider only letters and numbers.

The following are non-valid inputs:

```
1L1
UYT5Y
```

...because they have repeated elements, and as it's well known, a Latin square cannot have repeated elements.

You are asked to write a program that fills in the rest of the rows to obtain array representations of n×n matrices with the same set of symbols from the first row's Latin squares.

The program should output all possible matrices that are Latin squares with that requirement. If the number of elements is 3, there will be 6 solutions (3!), and these solutions should be displayed separated by the text "===". If there are 4 elements, the number of solutions will be 24 (4!), etc.

When printed, the solutions should appear ordered alphabetically.

You are asked to display all the ordered combinations, separated by a line with three "===" symbols. The concept of "ordered" refers to the position of the element

Input

A single line identifying the components of the first row of the latin square box.

Output

The whole latin square box.

Example 1

Input

```
1234
```

Output

```
1234
2341
3412
4123
===
1243
2431
4312
3124
===
1324
3241
2413
4132
===
1342
3421
4213
2134
```

```
===
1423
4231
2314
3142
===
1432
4321
3214
2143
===
2134
1342
3421
4213
===
2143
1432
4321
3214
===
2314
3142
1423
4231
===
2341
3412
4123
1234
===
2413
4132
1324
3241
```

```
===
2431
4312
3124
1243
===
3124
1243
2431
4312
===
3142
1423
4231
2314
===
3214
2143
1432
4321
===
3241
2413
4132
1324
===
3412
4123
1234
2341
===
3421
4213
2134
1342
```

```
===
4123
1234
2341
3412
===
4132
1324
3241
2413
===
4213
2134
1342
3421
===
4231
2314
3142
1423
===
4312
3124
1243
2431
===
4321
3214
2143
1432
```

Example 2

Input

```
ABC
```

Output

```
ABC
BCA
CAB
===
ACB
CBA
BAC
===
BAC
ACB
CBA
===
BCA
CAB
ABC
===
CAB
ABC
BCA
===
CBA
BAC
ACB
```

Solutions

Python

```python
import itertools
def es_cuadrado_latino(fila):
    # Comprobar si la fila tiene caracteres repetidos
    return len(set(fila)) == len(fila)
def generar_cuadrado_latino(fila):
    n = len(fila)
    cuadrado = []
    # Añadir la primera fila
    cuadrado.append(fila)
    # Rellenar las siguientes filas con rotaciones cíclicas de la fila
superior
    for i in range(1, n):
        fila_rotada = fila[i:] + fila[:i]
        cuadrado.append(fila_rotada)
    return cuadrado
def agregar_a_soluciones(cuadrado, soluciones):
    # Convertir el cuadrado latino en una cadena representativa para las
soluciones
    solucion = '\n'.join(''.join(fila) for fila in cuadrado)
    soluciones.append(solucion)
def main():
    # Leer la fila superior
    fila_superior = input().strip()
    # Verificar que la fila sea válida (sin elementos repetidos)
    if es_cuadrado_latino(fila_superior):
        soluciones = []  # Lista para almacenar las soluciones
        # Generar todas las permutaciones de la fila
        permutaciones = set(itertools.permutations(fila_superior))
        # Para cada permutación, generar el cuadrado latino y agregarlo a las
soluciones
        for perm in permutaciones:
            cuadrado = generar_cuadrado_latino(''.join(perm))
            agregar_a_soluciones(cuadrado, soluciones)
        # Ordenar las soluciones alfabéticamente
        soluciones.sort()
        # Imprimir las soluciones ordenadas
        for idx, solucion in enumerate(soluciones):
            print(solucion)
            if idx != len(soluciones) - 1:
                print("===")
    else:
        print("La fila no es válida para un cuadrado latino (tiene elementos
repetidos).")
if __name__ == "__main__":
    main()
```

C++

```
#include <iostream>
#include <vector>
#include <set>
#include <algorithm>
#include <string>
// Función para verificar si la fila tiene caracteres repetidos
bool es_cuadrado_latino(const std::string &fila) {
    std::set<char> elementos;
    for (char c : fila) {
        elementos.insert(c);
    }
    return elementos.size() == fila.size();
}
// Función para generar un cuadrado latino a partir de una fila
std::vector<std::string> generar_cuadrado_latino(const std::string &fila) {
    int n = fila.size();
    std::vector<std::string> cuadrado;
    // Añadir la primera fila
    cuadrado.push_back(fila);
    // Rellenar las siguientes filas con rotaciones cíclicas de la fila
superior
    for (int i = 1; i < n; ++i) {
        std::string fila_rotada = fila.substr(i) + fila.substr(0, i);
        cuadrado.push_back(fila_rotada);
    }
    return cuadrado;
}
// Función para agregar el cuadrado latino al vector de soluciones
```

```
void agregar_a_soluciones(const std::vector<std::string> &cuadrado,
std::vector<std::string> &soluciones) {
    std::string solucion;
    for (const auto &fila : cuadrado) {
        solucion += fila + "\n";
    }
    soluciones.push_back(solucion);
}
// Función principal
int main() {
    // Leer la fila superior
    std::string fila_superior;
    std::getline(std::cin, fila_superior);
    // Verificar que la fila sea válida (sin elementos repetidos)
    if (es_cuadrado_latino(fila_superior)) {
        std::vector<std::string> soluciones;  // Lista para almacenar las
soluciones
        // Generar todas las permutaciones de la fila
        std::set<std::string> permutaciones;
        std::sort(fila_superior.begin(), fila_superior.end());
        do {
            permutaciones.insert(fila_superior);
        } while (std::next_permutation(fila_superior.begin(),
fila_superior.end()));
        // Para cada permutación, generar el cuadrado latino y agregarlo a
las soluciones
        for (const auto &perm : permutaciones) {
            std::vector<std::string> cuadrado =
generar_cuadrado_latino(perm);
            agregar_a_soluciones(cuadrado, soluciones);
        }
        // Ordenar las soluciones alfabéticamente
        std::sort(soluciones.begin(), soluciones.end());
        // Imprimir las soluciones ordenadas
        for (size_t idx = 0; idx < soluciones.size(); ++idx) {
            std::cout << soluciones[idx];
            if (idx != soluciones.size() - 1) {
                std::cout << "===" << std::endl;
            }
        }
    } else {
        std::cout << "La fila no es válida para un cuadrado latino (tiene
elementos repetidos)." << std::endl;
    }
    return 0;
}
```

Java

```
import java.util.*;
public class CuadradoLatino {
    // Función para verificar si la fila tiene caracteres repetidos
    public static boolean esCuadradoLatino(String fila) {
        Set<Character> elementos = new HashSet<>();
        for (char c : fila.toCharArray()) {
            elementos.add(c);
        }
        return elementos.size() == fila.length();
    }
    // Función para generar un cuadrado latino a partir de una fila
    public static List<String> generarCuadradoLatino(String fila) {
        int n = fila.length();
        List<String> cuadrado = new ArrayList<>();
        // Añadir la primera fila
        cuadrado.add(fila);
        // Rellenar las siguientes filas con rotaciones cíclicas de la fila
superior
        for (int i = 1; i < n; ++i) {
            String filaRotada = fila.substring(i) + fila.substring(0, i);
            cuadrado.add(filaRotada);
        }
        return cuadrado;
    }
    // Función para agregar el cuadrado latino al array de soluciones
    public static void agregarASoluciones(List<String> cuadrado, List<String>
soluciones) {
        StringBuilder solucion = new StringBuilder();
        for (String fila : cuadrado) {
            solucion.append(fila).append("\n");
        }
        soluciones.add(solucion.toString());
    }
    // Función principal
    public static void main(String[] args) {
        // Leer la fila superior
        Scanner scanner = new Scanner(System.in);
        String filaSuperior = scanner.nextLine().trim();
        // Verificar que la fila sea válida (sin elementos repetidos)
        if (esCuadradoLatino(filaSuperior)) {
            List<String> soluciones = new ArrayList<>();  // Lista para
almacenar las soluciones
            // Generar todas las permutaciones de la fila
            Set<String> permutaciones = new TreeSet<>();  // Usamos TreeSet
para ordenarlas automáticamente
            // Generar todas las permutaciones de la fila
            char[] filaArray = filaSuperior.toCharArray();
            Arrays.sort(filaArray);  // Ordenamos la fila alfabéticamente
antes de generar permutaciones
            do {
                permutaciones.add(new String(filaArray));  // Añadimos cada
permutación
            } while (nextPermutation(filaArray));
            // Para cada permutación, generar el cuadrado latino y agregarlo
a las soluciones
            for (String perm : permutaciones) {
                List<String> cuadrado = generarCuadradoLatino(perm);
                agregarASoluciones(cuadrado, soluciones);
            }
            // Imprimir las soluciones ordenadas
            Iterator<String> iterator = soluciones.iterator();
            while (iterator.hasNext()) {
                System.out.print(iterator.next());
                if (iterator.hasNext()) {
                    System.out.println("===");
                }
            }
        } else {
            System.out.println("La fila no es válida para un cuadrado latino
(tiene elementos repetidos).");
        }
    }
    // Función para generar la siguiente permutación (modificación de la
función next_permutation)
    public static boolean nextPermutation(char[] array) {
        int i = array.length - 1;
        while (i > 0 && array[i - 1] >= array[i]) {
            i--;
        }
        if (i <= 0) {
            return false;
        }
        int j = array.length - 1;
        while (array[j] <= array[i - 1]) {
            j--;
        }
        char temp = array[i - 1];
        array[i - 1] = array[j];
        array[j] = temp;
        j = array.length - 1;
        while (i < j) {
            temp = array[i];
            array[i] = array[j];
            array[j] = temp;
            i++;
            j--;
        }
        return true;
    }
}
```

30 This lottery is crazy

14 points

Introduction

In a school in Albuquerque, from time to time they receive gifts (from spinning tops to tablets) that they want to give to their students and they have come up with an ingenious way to raffle these products.The goal is to randomly select one student from each class and find the combination that yields the highest score based on a mathematical formula.

Here’s what we have:

A number L, which is the limit of students to consider from each list.

Three classes of students, each represented by a list of student numbers.

The combination of one student from each class will be evaluated using the following formula:

S=(A^3+B^3+C^3)

Where:

- a is the student number from the first class.
- b is the student number from the second class.
- c is the student number from the third class.
- The score S will be calculated for all possible combinations of students from the three classes.

The combination that yields the highest score S should be selected.

Exercise

Input

The input consists of several lines:

- The first line contains the value of L, the number of students to consider from each class.
- The following three lines contain the lists of students for the three classes. Each list is a comma-separated list of student numbers.

For instance:

7

```
2,18,4,12,9,10,22,11,1,3,24,23,12,14,5,6
3,7,8,9,25,24,23,12,14,1,10,22,11,6,2,4,5
5,10,8,9,1,19,17,15,13,6,1,2,3,24,25,4,11
```

Output

The output will consist of the winning numbers of each class and the value of S obtained among the three.

In this example it would be:

```
a,b,c = S
```

Example 1

Input

```
7
2,18,4,12,9,10,22,11,1,3,24,23,12,14,5,6
3,7,8,9,25,24,23,12,14,1,10,22,11,6,2,4,5
5,10,8,9,1,19,17,15,13,6,1,2,3,24,25,4,11
```

Output

```
18,25,10 = 22457
```

Example 2

Input

```
5
20,12,14,18,2,22
11,13,15,19,21,23
9,12,16,18,25,27
```

Output

```
20,21,25 = 32886
```

Solutions

Python

```
import math
```

```
L = 0  # Límite de estudiantes a considerar
GANADORES = ()  # Combinación ganadora (a, b, c)
MAX_S = -1  # Puntaje máximo
# Función para calcular el puntaje S dado a, b, y c
def calcular_s(a, b, c):
    return int(math.pow(a, 3) + math.pow(b, 3) + math.pow(c, 3))
# Función para obtener el máximo puntaje y la combinación correspondiente por
separado
def obtener_combinaciones(clase_a, clase_b, clase_c):
    global MAX_S, GANADORES
    MAX_S = -1
    # Iterar sobre todas las combinaciones posibles
    for a in clase_a:
        for b in clase_b:
            for c in clase_c:
                s = calcular_s(a, b, c)
                if s > MAX_S:
                    MAX_S = s
                    GANADORES = (a, b, c)
# Función para parsear una línea de entrada en una lista de enteros
def parse_linea(linea):
    return [int(numero) for numero in linea.split(',')]
# Función principal
def main():
    global L, GANADORES, MAX_S
    # Leer L
    L = int(input().strip())
    # Leer las tres listas de números de cada clase
    clases = []
    for _ in range(3):
        clase = parse_linea(input().strip())
        if L > len(clase):
            print("El valor de L no puede ser mayor que el número de
estudiantes en una clase.")
            return
        # Tomar solo los primeros L números
        clases.append(clase[:L])
    obtener_combinaciones(clases[0], clases[1], clases[2])
    print(f"{GANADORES[0]},{GANADORES[1]},{GANADORES[2]} = {MAX_S}")
if __name__ == "__main__":
    main()
```

C++

```
#include <iostream>
#include <vector>
#include <sstream>
#include <tuple>
#include <cmath>
int L; // Límite de estudiantes a considerar
std::tuple<int, int, int> GANADORES;
int MAX_S;
// Función para calcular el puntaje S dado a, b, y c
int calcular_s(int a, int b, int c) {
    return static_cast<int>(std::pow(a, 3) + std::pow(b, 3) + std::pow(c,
3));
}
// Función para obtener el máximo puntaje y la combinación correspondiente
por separado
void obtener_combinaciones(const std::vector<int>& clase_a,
    const std::vector<int>& clase_b,
    const std::vector<int>& clase_c) {
    MAX_S = -1;
    // Iterar sobre todas las combinaciones posibles
    for (int a : clase_a) {
        for (int b : clase_b) {
            for (int c : clase_c) {
                int s = calcular_s(a, b, c);
                if (s > MAX_S) {
                    MAX_S = s;
                    GANADORES = std::make_tuple(a, b, c);
                }
            }
        }
    }
}
// Función para parsear una línea de entrada en un vector de enteros
std::vector<int> parse_linea(const std::string& linea) {
    std::istringstream iss(linea);
    std::vector<int> resultado;
    std::string numero;
    while (std::getline(iss, numero, ',')) {
        resultado.push_back(std::stoi(numero));
    }
    return resultado;
}
int main() {
    std::string input;
    // Leer L
    std::getline(std::cin, input);
    L = std::stoi(input);
    // Leer las tres listas de números de cada clase
    std::vector<std::vector<int>> clases(3);
    for (int i = 0; i < 3; ++i) {
        std::getline(std::cin, input);
        std::vector<int> clase = parse_linea(input);
        if (L > clase.size()) {
            std::cout << "El valor de L no puede ser mayor que el numero de
estudiantes en una clase." << std::endl;
            return 1;
        }
        // Tomar solo los primeros L números
        clases[i] = std::vector<int>(clase.begin(), clase.begin() + L);
    }
    obtener_combinaciones(clases[0], clases[1], clases[2]);
    std::cout << std::get<0>(GANADORES) << ","
```

```
        << std::get<1>(GANADORES) << ","
        << std::get<2>(GANADORES) << " = "
        << MAX_S << std::endl;
    return 0;
}
```

Java

```
import java.util.*;
public class Main {
    // Límite de estudiantes a considerar
    static int L;
    // Combinación ganadora (a, b, c)
    static int[] GANADORES = new int[3];
    // Puntaje máximo
    static int MAX_S = -1;
    // Función para calcular el puntaje S dado a, b, y c
    public static int calcularS(int a, int b, int c) {
        return (int) (Math.pow(a, 3) + Math.pow(b, 3) + Math.pow(c, 3));
    }
    // Función para obtener el máximo puntaje y la combinación
correspondiente por separado
    public static void obtenerCombinaciones(List<Integer> claseA,
List<Integer> claseB, List<Integer> claseC) {
        MAX_S = -1;
        // Iterar sobre todas las combinaciones posibles
        for (int a : claseA) {
            for (int b : claseB) {
                for (int c : claseC) {
                    int s = calcularS(a, b, c);
                    if (s > MAX_S) {
                        MAX_S = s;
                        GANADORES[0] = a;
                        GANADORES[1] = b;
                        GANADORES[2] = c;
                    }
                }
            }
        }
    }
    // Función para parsear una línea de entrada en una lista de enteros
    public static List<Integer> parseLinea(String linea) {
        List<Integer> resultado = new ArrayList<>();
        String[] numeros = linea.split(",");
        for (String numero : numeros) {
            resultado.add(Integer.parseInt(numero.trim()));
        }
        return resultado;
    }
    // Función principal
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Leer L
        L = Integer.parseInt(scanner.nextLine().trim());
        // Leer las tres listas de números de cada clase
        List<List<Integer>> clases = new ArrayList<>();
        for (int i = 0; i < 3; i++) {
            String input = scanner.nextLine().trim();
            List<Integer> clase = parseLinea(input);
            if (L > clase.size()) {
                System.out.println("El valor de L no puede ser mayor que el
número de estudiantes en una clase.");
                return;
            }
            // Tomar solo los primeros L números
            clases.add(clase.subList(0, L));
        }
        obtenerCombinaciones(clases.get(0), clases.get(1), clases.get(2));
        System.out.println(GANADORES[0] + "," + GANADORES[1] + "," +
GANADORES[2] + " = " + MAX_S);
    }
    }
```