CODEWARS | hp

LEÓN - MADRID 2023

CodeWars solutions 2023

Toda la información en:
hpscds.com/codewars2023/

hp SCDS

1 Welcome to Codewars

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 2023!".

Exercise

Write a program that outputs the string "Welcome to Codewars 2023!".

Input

None.

Output

The string "Welcome to Codewars 2023!".

Example 1

Input

No input for this problem.

Output

```
Welcome to Codewars 2023!
```

Solutions

Python

```
def main():
    print("Welcome to Codewars 2023!")
if __name__ == "__main__":
    main()
```

C++

```
#include <iostream>
int main(void) {
```

```
    std::cout << "Welcome to Codewars 2023!" << std::endl;
    return 0;
}
```

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 2023, TeamName!*, where *TeamName* is the name of the team you input.

Exercise

Write a program that outputs the string *Welcome to Codewars 2023, 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 2023, TeamName!* where *TeamName* is the name of the team you input.

Example 1

Input

```
Trambolikos
```

Output

```
Welcome to Codewars 2023, Trambolikos!
```

Example 2

Input

```
Los top del Mundo
```

Output

```
Welcome to Codewars 2023, Los top del Mundo!
```

Solutions

Python

```
def main():
```

```
    team = input()
    print("Welcome to Codewars 2023, " + 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 2023, " << team << "!" << std::endl;
    return 0;
}
```

3 3TPG yb srawedoC ot emocleW

4 points

Introduction

DISCLAIMER: This problem has been written completely by GPT3, a model language that for sure you have heard about lately.

You are a member of an archeological team that's exploring a dense, wild forest in search of ancient ruins. One of the ruins you discover has inscriptions on the walls, but the inscriptions are in an ancient language and written in reverse order.

Exercise

Your task is to write a program that takes in the encoded inscription as a string, and then prints out the original text in the correct order.

Input

A single string, representing the encoded inscription on the ancient ruins.

Output

A single string, representing the inscription in the correct order.

Example 1

Input

```
!3TPG yb ,3202 sraWedoC ot emocleW
```

Output

```
Welcome to CodeWars 2023, by GPT3!
```

Example 2

Input

```
...terces neddih a seil sniur eseht nI
```

Output

```
In these ruins lies a hidden secret...
```

Solutions

Python

```
def main():
    print(input()[::-1])
if __name__ == "__main__":
    main()
```

C++

```
#include <iostream>
#include <string>
std::string reverseString(std::string str) {
    int n = str.length();
    for (int i = 0; i < n / 2; i++) {
        std::swap(str[i], str[n - i - 1]);
    }
    return str;
}
int main() {
    std::string str;
    std::getline(std::cin, str);
    std::cout << reverseString(str) << std::endl;
    return 0;
}
```

4 Literal Acronyms

5 points

Introduction

An alien race called the Literalites has arrived to Earth. They had no trouble learning our languages, however they have the "problem" that they always interpret everything in a literal sense. So, whenever they see one of our acronyms that include a number and then some letters, they think its meaning is to repeat the letters as many times as the precedent number states. For example, when they see "4K" (for the resolution), they think it means "KKKK". And when they see 5G (for the mobile network speed), they think it means "GGGGG". Same for other acronyms like "1080p" (for number of lines in a screen) or "30fps" (for frames per second).

Exercise

You are asked to write a program that takes one of those acronyms and outputs what the aliens will interpret. See examples below.

Input

An acronym like "720p" or "3G".

Output

The interpretation by the aliens. For example, for "3G" it would be "GGG".

Example 1

Input

```
4K
```

Output

```
KKKK
```

Example 2

Input

```
1080p
```

Output

```
pppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp
pppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp
pppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp
```

```
ppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp
ppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp
ppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp
ppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp
ppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp
ppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp
ppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp
ppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp
ppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp
ppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp
ppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp
pp
```

Example 3

Input

```
30fps
```

Output

```
fpsfpsfpsfpsfpsfpsfpsfpsfpsfpsfpsfpsfpsfpsfpsfpsfpsfpsfpsfpsfpsfpsfpsfpsfpsfp
sfpsfpsfpsfps
```

Solutions

Python

```python
import re
def main():
    match = re.match(r'(?P<number>\d+)(?P<string>\w+)', input())
    if match is None:
        print('Error: invalid input')
        return
    times = int(match.group('number'))
    string = match.group('string')
    print(string*times)
if __name__ == '__main__':
    main()
```

C++

```
#include <iostream>
#include <regex>
int main()
{
    std::string input;
    std::getline(std::cin, input);
    std::regex pattern(R"((\d+)(\w+))");
    std::smatch match;
    if (!std::regex_match(input, match, pattern))
    {
        std::cout << "Error: invalid input" << std::endl;
        return 0;
    }
    int times = std::stoi(match.str(1));
    std::string string = match.str(2);
    for (int i = 0; i < times; ++i)
    {
        std::cout << string;
    }
    std::cout << std::endl;
    return 0;
}
```

5 The Neptune moons

5 points

Introduction

Neptune has 14 satellites, but almost nobody knows the order of proximity to the planet because, among other reasons, their orbital excentricities are very different and the order changes depending on the orbital location. It's generally agreed that the order of the fourteen Neptune moons (starting by the closest one) is Naiad, Thalassa, Despina, Galatea, Larissa, Hippocamp, Proteus, Triton, Nereid, Halimede, Sao, Laomedeia, Psamathe and Neso.

Exercise

You are asked to write a program to return the name of a moon given its order number.

Input

The input will be a number from 1 to 14.

Output

If the input is not a number, or is not a number from 1 to 14, the program will return

ERROR

Otherwise it will return the name of the moon, as written in the Introduction.

Example 1

Input

```
4
```

Output

```
Galatea
```

Example 2

Input

```
9
```

Output

```
Nereid
```

Example 3

Input

```
99
```

Output

```
ERROR
```

Solutions

Python

```
import sys
def main(argv):
   nNumero=int(input(""))
   moons=["Naiad","Thalassa", "Despina", "Galatea", "Larissa", "Hippocamp",
"Proteus", "Triton", "Nereid", "Halimede", "Sao", "Laomedeia",
"Psamathe","Neso"];
   if (nNumero<1):
      print("ERROR")
   elif (nNumero>14):
      print("ERROR")
   else:
      print(moons[nNumero-1])
if __name__ == "__main__":
   main(sys.argv)
```

C++

```
#include <string>
#include <iostream>
#include <vector>
#include <sstream>
int main(int argc, char **argv)
{
    std::string moons[14]={"Naiad","Thalassa", "Despina", "Galatea",
"Larissa", "Hippocamp", "Proteus", "Triton", "Nereid", "Halimede", "Sao",
"Laomedeia", "Psamathe","Neso"};
    std::string input_text="";
    //INPUT
```

```
    std::cin >> input_text;
    int input_val=std::stoi(input_text);
    if (input_val<=0 || input_val>14)
        std::cout<<"ERROR";
    else
        std::cout<<moons[input_val-1];
    return 0;
}
```

6 Automorph

6 points

Introduction

In mathematics, an automorphic number (sometimes referred to as a circular number) is a natural number in a given number base *b* whose square "ends" in the same digits as the number itself. For example, in base 10:

```
(25)^2 = 625
(76)^2 = 5776
```

Exercise

You are asked to write a program to test if a given number is an automorphic number. To check whether any positive number is an automorphic number or not, simply check that the last digits of the resulting square number are the original number.

Input

Input will consist in a positive number (greater or equal to zero).

Output

If the number is an automorphic number the output will be:

```
Automorphic!!
```

Otherwise the output will be:

```
No automorphic :(
```

Example 1

Input

```
25
```

Output

```
Automorphic!!
```

Example 2

Input

```
21
```

Output

```
No automorphic :(
```

Example 3

Input

```
0
```

Output

```
Automorphic!!
```

Solutions

Python

```
import sys
#  ::   .::::::::::::.
# ,;;   ;;,`;;;```.;;;
#,[[[,,,[[[ `]]nnn]]'
#"$$$"""$$$  $$$""
# 888   "88o 888o
# MMM    YMM YMMMb
#  .,-:::::     ...    :::::::-.  .,::::::      .::    .   .:::::::.
:::::::..   .:::::::.
#,;;;'````'  .;;;;;;;.  ;;,   `';,;;;;''''      ';;,  ;;  ;;;' ;;`;;
;;;;``;;;; ;;;`    `
#[[[        ,[[     \[[,`[[     [[ [[cccc        '[[, [[, [[' ,[[ '[[,
[[[,/[[[' '[==/[[[[,
#$$$        $$$,     $$$ $$,    $$ $$""""          Y$c$$$c$P c$$$cc$$$c
$$$$$$c     '''    $
#`88bo,__,o,"888,_ _,88P 888_,o8P' 888oo,__         "88"888   888   888,888b
"88bo,88b    dP
#  "YUMMMMMP" "YMMMMMP"  MMMMP"`   """"YUMMM         "M "M"   YMM   ""` MMMM
"W"  "YMmMY"

#In mathematics, an automorphic number (sometimes referred to as a circular
number) is a natural
```

```
#number in a given number base b whose square "ends" in the same digits as
the number itself.
#(25)2 = 625
#(76)2 = 5776
def is_automorphic(number):
    n_sqr = number*number
    while (number > 0):
        if number%10 != n_sqr%10:
            return False
        number = number//10
        n_sqr = n_sqr//10

    return True
if __name__ == "__main__":
    number = int(input(""))
    # Test Automorphic
    if (is_automorphic(number)):
        print("Automorphic!!")
    else:
        print("No automorphic :(")
```

C++

```
#include <iostream>
#include <string>
using namespace std;
/*
  ::   .::::::::::::.
 ,;;   ;;,`;;;```.;;;
,[[[,,,[[[ `]]nnn]]'
"$$$"""$$$  $$$""
 888   "88o 888o
 MMM    YMM YMMMb
  .,-:::::     ...    :::::::-.  .,::::::      .::    .   .:::::::.
:::::::..   .:::::::.
```

```
,;;;'````'  .;;;;;;;.  ;;,    `';,;;;;''''      ';;,  ;;  ;;;' ;;`;;
;;;;``;;;; ;;;`    `
[[[        ,[[     \[[,`[[     [[ [[cccc        '[[, [[, [[' ,[[ '[[,
[[[,/[[[' '[==/[[[[,
`88bo,__,o,"888,_ _,88P 888_,o8P' 888oo,__         "88"888   888   888,888b
"88bo,88b    dP
  "YUMMMMMP" "YMMMMMP"  MMMMP"`   """"YUMMM         "M "M"   YMM   ""` MMMM
"W"  "YMmMY"
```

In mathematics, an automorphic number (sometimes referred to as a circular number) is a natural
number in a given number base b whose square "ends" in the same digits as the number itself.
(25)2 = 625
(76)2 = 5776
*/

```
bool isAutomorphic( int nNumero )
{
    int nSqr = nNumero * nNumero;
    while( nNumero > 0 )
    {
        if( nNumero % 10 != nSqr % 10)
            return false;
        nNumero /= 10;
        nSqr /= 10;
    }
    return true;
}
int main()
{
    int nNumber;
    cin >> nNumber;
    // Test Automorphic
    if( isAutomorphic( nNumber ) )
      cout << "Automorphic!!";
```

```
    else
      cout << "No automorphic :(";
}
```

7 DNI letter

6 points

Introduction

You are new in a company that's developing a web interface for the the Spanish government.

In order to authenticate, users need to type their DNI numbers with the appropiate letter.

Your boss has asked you to write a program to check the letter typed by users is correct.

Exercise

To calculate the letter associated to the number, the number has to be divided by 23.

The remainder on this calculation indicates an index in a list of letters, in which you can get the associated letter.

This is the list of letters:

`TRWAGMYFPDXBNJZSQVHLCKE`

Given an ID number, generate the complete number with the associated letter.

Input

A positive number (normally 7 or 8 digits long, but there are shorter ones).

Output

The output will be the letter for the DNI corresponding to the number. Letter will be in uppercase, like in the list above.

Examples

Test the numbers in your DNIs. They should work.

Solutions

Python

```
import sys
def main(argv):
   nNumero=int(input(""))
   cDniLetters = "TRWAGMYFPDXBNJZSQVHLCKE";
```

```
    nLetterIndex = nNumero % 23;
    print(cDniLetters[nLetterIndex%23])
if __name__ == "__main__":
    main(sys.argv)
```

C++

```
#include <iostream>
#include <string>
using namespace std;
/*
  ::    .::::::::::::.
 ,;;    ;;,`;;;```.;;;
,[[[,,,[[[ `]]nnn]]'
"$$$"""$$$  $$$""
 888   "88o 888o
 MMM    YMM YMMMb
  .,-:::::     ...    :::::::-.  .,::::::      .::    .   .:::::::.
:::::::..   .:::::::.
,;;;'````'  .;;;;;;;.  ;;,   `';,;;;;''''      ';;,  ;;  ;;;' ;;`;;
;;;;``;;;; ;;;`    `
[[[        ,[[     \[[,`[[     [[ [[cccc        '[[, [[, [[' ,[[ '[[,
[[[,/[[[' '[==/[[[[,
`88bo,__,o,"888,_ _,88P 888_,o8P' 888oo,__         "88"888   888   888,888b
"88bo,88b    dP
  "YUMMMMMP" "YMMMMMP"  MMMMP"`   """"YUMMM         "M "M"   YMM   ""` MMMM
"W"  "YMmMY"
```

Calculate Spanish DNI number
To calculate the letter associated to the number, the number has to be divided by 23
The rest on this calculation indicates a index in a list of letters, in which you can get the associated letter
This is the list of letters : TRWAGMYFPDXBNJZSQVHLCKE
Given an ID number, generate the complete number with the associated leter
*/

```
int main()
{
    int nNumero;
    cin >> nNumero;
    //
    string cDniLetters = "TRWAGMYFPDXBNJZSQVHLCKE";
    int nLetterIndex = nNumero % 23;
    //cout << nNumero;
    cout << cDniLetters.substr( nLetterIndex , 1 );
    cout << "\r\n";
}
```

8 Prime calculator

7 points

Introduction

A prime number is a natural number larger than 1 that is divisible only by itself and 1 (the first prime number is 2). Prime numbers are very important in cryptography. Every time you use HTTPS, or use a password, you are using prime numbers.

Prime numbers are used because is hard to check if a number is prime or not and determine which two prime numbers has been used to multiply one by the other one and obtain a number. So lets take a look how it makes slowe and slower as the number to be checked makes bigger and bigger.

Smaller prime numbers used in cryptography are 2048 bits in length which means more than 600 digits (log10(2^2048))... Computers are amazing!!!

Exercise

To calculate if a number is a prime, you need to check this sentence: "A primer number is a positive number that is divisible only by itself and 1". You are asked to write a program that checks if a number is a prime.

Input

Input will consist in a positive number (greater than zero).

Output

If the input is not a positive integer number, the program will return:

```
ERROR
```

Otherwise it will return any of these sentences:

```
The number is a prime.
```

or

```
The number is NOT a prime.
```

Example 1

Input

```
13
```

Output

```
The number is a prime.
```

Example 2

Input

```
428
```

Output

```
The number is NOT a prime.
```

Example 3

Input

```
-7141
```

Output

```
ERROR
```

Solutions

Python

```
def main():
    to_check = int(input())
    if to_check <= 0:
        print("ERROR")
    else:
        is_prime = True
        if to_check == 1: # 1 is not prime by definition
            is_prime = False
        else:
            # no need to go up to the number, just half of it rounded down
            for i in range(2, to_check//2 + 1):
                if (to_check % i) == 0:
                    is_prime = False
                    break
        print("The number is a prime." if is_prime else "The number is NOT a
prime.")
if __name__ == "__main__":
```

```
    main()
```

C++

```
#include <stdio.h>
#define YES 1
#define NO 0
int main()
{
    int target = 0;
    int numTesting = 0;
    char isPrime = YES;

    scanf("%d", &target);
    if (target < 0){
        printf("ERROR");
        return 0;
    }

    if (target == 1 ){
        return YES;
    }
    for (numTesting = 2; numTesting < target; numTesting ++)
    {
        if (target % numTesting == 0){
            isPrime = NO;
        }
    }

    if (isPrime){
        printf("The number is a prime.");
    }
    else
    {
        printf("The number is NOT a prime.");
    }
```

```
    return 0;
}
```

9 Hotel Lemniscata

8 points

Introduction

Welcome to the Lemniscata Hotel, the hotel to host the entire humanity and one person more, or whole humanity plus one person and one person more, any amount of guests is allowed and even one more. And endless place endlessly.

Exercise

After building the hotel and before accepting guests, the hotel's director wants to know the average age of the guests. You are asked to write a program that receives the age of every guest arriving at the hotel until a negative value is entered and then prints on the screen the average age of all the guests.

Input

The input will be a variable number of lines indicating the age of each guest (one number in each line). The input ends when a negative value is typed.

Output

The output will consist in two lines:

```
The number of hosts is: <NN>
The average age of hosts is: <XX.YY>
```

NN will be a positive integer value. The average age will be a positive value with two decimals, rounded to closest fraction.

Example 1

Input

```
13
12
88
45
-4
```

Output

```
The number of hosts is: 4
```

```
The average age of hosts is: 39.50
```

Example 2

Input

```
-28
```

Output

```
The number of hosts is: 0
The average age of hosts is: 00.00
```

Example 3

Input

```
7
14
1
55
12
11
44
55
66
-1
```

Output

```
The number of hosts is: 9
The average age of hosts is: 29.44
```

Solutions

Python

```
import sys
def main(argv):
    edadTemp = 0
    cantidadHuespedes = 0
    edadAcumulada = edadTemp
    while (edadTemp >= 0):
        edadTemp = int(input(""))
```

```
        if edadTemp >= 0:
            edadAcumulada += edadTemp
            cantidadHuespedes += 1
    print(f"\nThe number of hosts is: {cantidadHuespedes}")
    if cantidadHuespedes == 0:
        print(f"\nThe average age of hosts is: 0")
    else:
        val = edadAcumulada / cantidadHuespedes
        print(f"\nThe average age of hosts is: {val:.2f}")
if __name__ == "__main__":
    main(sys.argv)
```

C++

```
#include <stdio.h>
#define EDAD_MIN 0
int main()
{
    unsigned long long edadAcumulada = 0;
    unsigned long long cantidadHuespedes = 0;
    int edadTemp = 0;
    do
    {
        scanf ("%i", &edadTemp);
        if (edadTemp >= EDAD_MIN){
            edadAcumulada += edadTemp;
            cantidadHuespedes += 1;
        }
    } while (edadTemp >= EDAD_MIN);
    printf ("\nThe number of hosts is: %lli", cantidadHuespedes);
    printf ("\nThe average age of hosts is: %.2f", cantidadHuespedes != 0 ?
edadAcumulada / (float)cantidadHuespedes : 0);
    return 0;
}
```

10 About Primes

9 points

Introduction

You are asked to find the first *n* consecutive NON-prime numbers greater than *x*. Both values will be passed as parameters in the format *n;x* and both will be positive integers. See examples below.

Exercise

Write the program for to solve the problem above.

Input

Two positive integers in the same line separated by a semicolon, being the first one *n* and the second one *x*.

Output

First of the *n* consecutive NON-prime numbers greater than *x*.

Example 1

Input

```
10;39000000
```

Output

```
39000002
```

Given that 39000001 is prime, and then 39000002, 39000003, 39000004, 39000005, 39000006, 39000007, 39000008, 39000009, 39000010, 39000011, 39000012 are all non-prime, the first of the 10 consecutive non-prime numbers greater than 39000000 is 39000002.

Example 2

Input

```
7;5
```

Output

```
90
```

Starting at 6 (greater than 5), the first 7 consecutive non-prime numbers are 90, 91, 92, 93, 94, 95 and 96. Hence, the output should be 90.

Solutions

Python

```
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True
def main():
    n_non_primes, start = map(int, input().split(';'))
    n = start + 1
    count = 0
    result = 0
    while count < n_non_primes:
        if not is_prime(n):
            if (count == 0):
                result = n
            count += 1
        else:
            count = 0
        n += 1
    print(result)
if __name__ == "__main__":
    main()
```

C++

```
#include <iostream>
#include <cstdint>
#include <cstdio>
using I = unsigned int;
bool is_prime(I num)
{
    for (I i = 2; i < num; i++)
    {
```

```
        if (num % i == 0)
            return false;
    }
    return true;
}
// 10;39000000
// 10;39916702
// 39916802
int main()
{
    I non_primes_count, starting_point;
    // std::cout << (is_prime(11) ? "primo" : "no primo") << "\n";
    auto scanned = std::scanf("%u;%u", &non_primes_count, &starting_point);
    if (scanned != 2)
    {
        std::cerr << "Error: incorrect input\n";
        return 1;
    }
    /*
    std::cout << "non_primes_count: " << non_primes_count << "\n";
    std::cout << "starting_point: " << starting_point << "\n";
    //*/
    I count = 0;
    I number = starting_point + 1;
    I result;
    while (count < non_primes_count)
    {
        if (!is_prime(number))
        {
            if (count == 0)
                result = number;
            count++;
        }
        else
        {
```

```
            count = 0;
        }
        number++;
        // std::cout << "number: " << number << "\tcount: " << count << "\n";
    }
    std::cout << result << "\n";
    return 0;
}
```

11 Peak my streak

9 points

Introduction

Given a positive number N a streak is defined as the smallest positive integer K such that N+K is not divisible by K+1.

E.g: Given number N=13...

13 is divisible by 1

14 is divisible by 2

15 is divisible by 3

16 is divisible by 4

17 is NOT divisible by 5

Therefore the streak of 13 is 17.

Exercise

You are asked to write a program to calculate the streak of a given number. The program will receive a positive number as input and will simply return a number which is the streak of the previously inputted number.

Input

Input will consist in a positive number (greater than zero).

Output

The output value will be the streak of the number taken as input.

Example 1

Input

```
13
```

Output

```
17
```

Example 2

Input

421

Output

428

Example 3

Input

7141

Output

7148

Solutions

Python

```
def main() -> int:
    input_val = int(input())
    cont = True
    streak = 0
    i = 1
    while cont:
        if (input_val + i - 1) % i != 0:
            streak = input_val + i - 1
            cont = False
        else:
            i += 1
    print(streak)
if __name__ == '__main__':
    main()
```

C++

```
#include <string>
#include <iostream>
#include <regex>
int main(int argc, char **argv)
```

```
{
    std::string inputted="";
    std::cin >> inputted;
    int input_val=std::stoi(inputted);

    //Uncomment to see many possible solutions
    //
    //for (int input_val=2; input_val<9999; input_val++)
    //{
    bool cont=true;
    int streak=0;
    int i=1;
    while (cont)
    {
          if (((input_val+i-1) % i)!=0)
          {
                streak=input_val+i-1;
                cont=false;
          }
          else
          {
            i++;
          }
    }

    //int length =streak -input_val;
    //if (length>6)
    //{
    //std::cout<<std::endl<<input_val<<" --> ";
    std::cout<<streak;
    //}
    //}
    return 0;
}
```

12 The Broken Calculator

9 points

Introduction

The electronic calculators of the 80s, sometimes had the problem that some key was broken. Suppose a calculator in which to make a calculation, the number appears on the display but then is not taken into account. This problem has the advantage that its errors are consistent and with a small program it is possible to determine which is the failed key.

Exercise

You are asked to write a program to determine the broken key. The program will receive a series of integers, which will be added (only addition), and the final result, separated by ";", for example:

```
526;845;913;28;231;1511
```

Which correspond to the sum 526+845+913+28+231=1511. This sum is incorrect because one of the digits is not taken into account. In this case, the broken number is 3, because 526+845+91+28+21=1511. So the output should be:

```
3
```

In case of no broken number found, the output will be:

```
No broken numbers
```

Input

List of numbers to sum (including broken digit if any) and the final result, separated by ";".

Output

Broken digit or "No broken numbers" if no broken digit is found.

Example 1

Input

```
526;845;913;28;231;1511
```

Output

```
3
```

Example 2

Input

```
143;143;86
```

Output

```
1
```

Example 3

Input

```
564;1272;1836
```

Output

```
No broken numbers
```

Solutions

Python

```
def main():
    strInput = input()
    numbersStr = strInput.split(";")
    numbers = list(map(int, numbersStr))
    result = numbers.pop()
    total = sum(numbers)
    if(total == result):
        print("No broken numbers")
        return
    for i in range(0,10):
        ciphersSum = 0
        for n in numbers:
            units = 1
            currentCipher = n
            while currentCipher > 0:
                digit = currentCipher%10
                if digit != i:
                    ciphersSum+=digit*units
                    units*=10
                currentCipher//=10
        if ciphersSum == result:
            print(i)
```

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

C++

```
#include <iostream>
#include <vector>
#include <sstream>
int main()
{
    std::string input;
    std::cin >> input;
    // split the input using ; as delimiter
    std::stringstream ssinput(input);
    std::vector<int> numbers;
    std::string number;
    while (std::getline(ssinput, number, ';'))
    {
        numbers.push_back(std::stoi(number));
    }
    // get the result, last element of input, and remove it from vector
    int result = numbers.back();
    numbers.pop_back();
    int initialSum = 0;
    for (const auto &n : numbers)
    {
        initialSum += n;
    }
    if (initialSum == result)
    {
        // no broken digit
        std::cout << "No broken numbers";
        return 0;
    }
```

```
    // try to get the same result removing all digits equal to i from the
numbers
    for (int i = 0; i < 10; i++)
    {
        int sum = 0;
        for (const auto &n : numbers)
        {
            // for each number in vector, sum the cipher removing the
"suspect" number
            int units = 1;
            int currentCipher = n;
            while (currentCipher > 0)
            {
                int digit = currentCipher % 10;
                // if the digit is different of the suspect number, sum and
increase the units
                if (digit != i)
                {
                    sum += digit * units;
                    units = units * 10;
                }
                currentCipher /= 10;
            }
        }
        if (sum == result)
        {
            // broken digit found
            std::cout << i;
            break;
        }
    }
    return 0;
}
```

13 The Cult of the Curly Numbers

9 points

Introduction

A new thinking movement considers that only the numbers with curly traces are worthy of God. Only the digits 0, 3, 6, 8 and 9 are considered as curly, because all the other have some straight parts and that is considered to be a "dirty" number. As they don't know exactly how many valid numbers they have, they have asked you to develop a program that helps us identify, for a given number of digits, how many positive integers (including zero) with those digits are formed only by curly traces and thus are worthy numbers.

Exercise

The program will receive a positive integer indicating the number of desired digits and will return the number of all the posible numbers with those digits than can be created using only curly numbers.

Please notice that a number starting by "0" has a smaller number of digits. For example, the number 0986 counts as having 3 digits and not four, because the zero on the left doesn't count. However, the number 0 itself counts as a one digit number.

Input

A positive number (number of digits for the numbers to be counted).

Output

Number of numbers with those digits that are only formed with curly digits.

Example 1

Input

1

Output

5

Example 2

Input

2

Output

20

For this last example, the numbers found are 30, 33, 36... up to 99, which is the last 2 digit number with valid curly digits.

Solutions

Python

```
CURLY_NUMBERS = {'0', '3', '6', '8', '9'}
def main():
    n = int(input())
    min = pow(10, n-1) if n > 1 else 0
    max = pow(10, n)
    only_curly = 0
    for i in range(min, max):
        number_str = str(i)
        is_curly = True
        for j in range(0, len(number_str)):
            if not number_str[j] in CURLY_NUMBERS:
                is_curly = False
                break
        if is_curly:
            only_curly += 1
    print(only_curly)
if __name__ == "__main__":
    main()
```

C++

```
#include <iostream>
#include <math.h>
#include <string>
int main()
{
    int n;
```

```
    std::cin >> n;
    int min = n > 1 ? pow(10, n - 1) : 0;
    int max = pow(10, n);
    int onlyCurly = 0;
    for (int i = min; i < max; i++)
    {
        std::string numberStr = std::to_string(i);
        bool isCurly = true;
        for (int j = 0; j < numberStr.length(); j++)
        {
            char numberChar = numberStr[j];
            if (numberChar != '0' && numberChar != '3' && numberChar != '6'
&& numberChar != '8' && numberChar != '9')
            {
                isCurly = false;
                break;
            }
        }
        if (isCurly)
        {
            onlyCurly++;
        }
    }
    std::cout << onlyCurly << std::endl;
}
```

14 Coffee or Trees

10 points

Introduction

The HP employees decided to spend a day in November 2021 planting trees at Castrocontrigo, León, because a big fire in 2012 completely burned a large area of the forest there. When they get there they see that the holes for the trees have been digged with a drill because that's much quicker than using a shovel. They also see that there is a tent with free coffee because, although the day is not as cold as expected, the wind blows unforgivingly.

They start planting trees in the open holes at 11:00 in the morning, but soon they realize that they are really efficient doing that, so all the holes start having a tree already planted in them. Desperate they try to find a new hole, however some of the employees are not able to find a new one and decide to flee to the coffee tent to get a well deserved treat for a job well done.

Exercise

You are asked to create a program that takes as input the number of holes that are drilled, the number of HP employees planting trees, the time (in minutes) it takes for them to plant a tree and the current time in format HH:MM (remember the reforestation activity started at 11:00, also, see input section below for leading zeroes). Example:

```
155 20 5 11:35
```

And outputs the number of people planting trees and the number of people having coffee (assume that as soon as they finish planting a tree they can check the open holes and decide to go for coffee or keep planting immediately). That way we can know if a lot of people is already having coffee in order to go for lunch or drill more holes. For the previous example, output would be:

```
15 people are still planting trees and 5 people have gone for coffee
```

Because in 35 minutes they have planted (35 / 5) 20 = 140 trees so only 15 holes are still empty and the other 5 people are free to go for coffee.

If everybody is still planting trees, the output would be:

```
Everybody planting trees! More work to do!
```

However, if everybody has gone for coffee, the output would be:

```
COFFEE TIME!
```

There is no need to change the word *people* if there is just a single person doing one of the activies, the program doesn't intend to be gramatically perfect.

Input

Four values: 3 of them are positive integers (there are always trees to plant, employees to plant them, and the minimum time per tree is 1 minute), the last one is the current time. The time can have leading zeroes for the hours or not but for minutes it always has them (example: 06:05 and 6:05, but not 6:5). Assume that the time is always later than the start time, so it can be on the next day. And remember that the start time for the activity is 11:00.

Output

See examples, the output depends on the number of people planting trees or having coffee.

Example 1

Input

```
155 20 5 11:37
```

Output

```
15 people are still planting trees and 5 people have gone for coffee
```

Example 2

Input

```
100 10 10 2:20
```

Output

```
COFFEE TIME!
```

Example 3

Input

```
1000 52 15 11:25
```

Output

```
Everybody planting trees! More work to do!
```

Solutions

Python

```
START_HOUR = 11
def main():
```

```
    input_values = input().split()
    trees = int(input_values[0])
    employees = int(input_values[1])
    minutes_per_tree = int(input_values[2])
    current_time = input_values[3]
    splitted_time = current_time.split(sep=':')
    hours = int(splitted_time[0])
    minutes = int(splitted_time[1])
    if hours >= START_HOUR:
        elapsed_minutes = (hours - START_HOUR) * 60 + minutes
    else:
        elapsed_minutes = (hours + 24 - START_HOUR) * 60 + minutes
    planted_trees = (elapsed_minutes // minutes_per_tree) * employees
    if planted_trees < trees:
        pending_trees = trees - planted_trees
        if pending_trees >= employees:
            print("Everybody planting trees! More work to do!")
        else:
            print(
                f"{pending_trees} people are still planting trees and
{employees-pending_trees} people have gone for coffee")
    else:
        print("COFFEE TIME!")
if __name__ == "__main__":
    main()
```

C++

```
#include <iostream>
#include <string>
int main()
{
    const int StartHour = 11;
    int trees, employees, minutesPerTree;
    std::string currentTime;
    std::cin >> trees;
```

```
    std::cin >> employees;
    std::cin >> minutesPerTree;
    std::cin >> currentTime;
    const size_t delimiterPos = currentTime.find(':');
    const int hours = std::stoi(currentTime.substr(0, delimiterPos));
    const int minutes = std::stoi(currentTime.substr(delimiterPos + 1,
currentTime.length() - delimiterPos + 1));
    int elapsedMinutes;
    if (hours >= StartHour)
        elapsedMinutes = (hours - StartHour) * 60 + minutes;
    else
        elapsedMinutes = (hours + 24 - StartHour) * 60 + minutes;
    const int plantedTrees = (elapsedMinutes / minutesPerTree) * employees;
    if (plantedTrees < trees)
    {
        const int pendingTrees = trees - plantedTrees;
        if (pendingTrees >= employees)
            std::cout << "Everybody planting trees! More work to do!" <<
std::endl;
        else
            std::cout << pendingTrees << " people are still planting trees
and " << employees - pendingTrees << " people have gone for coffee" <<
std::endl;
    }
    else
        std::cout << "COFFEE TIME!";
}
```

Java

```
import java.util.Scanner;
public class Coffee {
    public static final int START_HOUR = 11;
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int trees = scan.nextInt();
```

```
        int employees = scan.nextInt();
        int minutesPerTree = scan.nextInt();
        String currentTime = scan.next();
        int delimiterPos = currentTime.indexOf(':');
        int hours = Integer.parseInt(currentTime.substring(0, delimiterPos));
        int minutes = Integer.parseInt(currentTime.substring(delimiterPos +
1));
        int elapsedMinutes;
        if (hours >= START_HOUR) {
            elapsedMinutes = (hours - START_HOUR) * 60 + minutes;
        } else {
            elapsedMinutes = (hours + 24 - START_HOUR) * 60 + minutes;
        }
        int plantedTrees = (elapsedMinutes / minutesPerTree) * employees;
        if (plantedTrees < trees) {
            int pendingTrees = trees - plantedTrees;
            if (pendingTrees >= employees) {
                System.out.println("Everybody planting trees! More work to
do!");
            } else {
                System.out.println(pendingTrees + " people are still planting
trees and " + (employees - pendingTrees) + " people have gone for coffee");
            }
        } else {
            System.out.println("COFFEE TIME!");
        }
    }
}
```

15 Dice Roller

10 points

Introduction

In HP we love board games. But there's always somebody who losses their dices and cannot play anymore.

We need somebody who write a program that throws virtual dices in order to play board games. Every game is different and may require from 2 to 5 dices. And every dice is different, they may have from 4 to 20 faces (numbered consecutively from 1 to 4 or from 1 to 20...).

But... pssst, only between you and me... let's modify the dice behaviour so that we can know the next dice score... Odd rolls will always increase by 3 while even rolls will always decrease by 2 in respect to the previous roll. Of course, if in a 6-face dice last roll was a 5 and you need to increase it by 3, it cannot be an 8, you need to start back from the lowest score: 4-5-6-1-2... so 5+3 means 2, or 1 - 2 means 5. This is an operation called "modulo", and its symbol is %. So (5+3) % 6=2.

But... remember: nobody should know about this trick.

Exercise

You are asked to write a program to obtain next dice roll, given a number of constraints. You will be given the number of dices and the number of faces of the dices. And then you will be given the last dice roll. And you need to write the next dice roll.

Input

The first input line will indicate the number of dices of the game and the number of faces of the dices. All dices have the same number of faces, separated by a comma:

```
<# dices>,<# faces>
```

The second input line will show the last dice roll, for all dices, separated by commas. Remember that this roll needs comply with the number of dices and the number of faces:

```
<dice 1 roll>,<dice 2 roll>,...,<dice N roll>
```

Output

The output will show next dice roll given that odd rolls (first, third, fifth...) will increase by 3 modulo num_faces while even rolls (second, fourth...) will decrease by 2 modulo num_faces. The output line will separate dice results by an space and will end up with an space and a dot.

Example 1

Input

```
5,15                    <---- This means 5 dices of 15 faces each
13,2,15,11,7            <---- This is the last roll
```

Output

```
1 15 3 9 10 .
```

This is because first roll (odd) was a 13, so 13+3=16, which is 1 modulo 15. Second roll (even) was a 2, so 2-2=0, which corresponds to 15 (because dice start at 1). Third roll (odd) was a 15, so 15+3=18, which is 3 modulo 15. Fourth roll (even) was a 11, so 11-2=9. Fifth roll (odd) was a 7, so 7+3=10.

Example 2

Input

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

Output

```
2 2 4 .
```

Example 3

Input

```
2,19
13,11
```

Output

```
16 9 .
```

Solutions

Python

```
#!/usr/bin/env python3
def is_odd(number):
    return bool(number % 2)
def is_even(number):
    return not(is_odd(number))
if __name__ == "__main__":
    dice_desc=input("")
```

```
    last_roll=input("")
    try:
        dices_n = int(dice_desc.split(',')[0])
        dices_faces = int(dice_desc.split(',')[1])
        rolls = list(map(int,last_roll.split(',')))
    except:
        print("The input is invalid.")
        exit(-1)
    if len(rolls) != dices_n:
        print(f"Hey, that's an invalid rol. You are playing with {dices_n}
dices but only entered {len(rolls)}")
        exit(-1)
    result = []
    for i,current_dice in enumerate(rolls):
        if is_odd(i):
            calc= current_dice-2 if current_dice-2 >0 else
dices_faces+current_dice-2
            result.append(str(calc))
        else:
            sol=str((current_dice+3)%dices_faces)
            if sol=="0":
                sol=str(dices_faces)
            result.append(sol)
    final_str = " ".join(result)
    print(f"{final_str} .")
```

C++

```
#include <string>
#include <iostream>
#include <vector>
#include <sstream>
int main(int argc, char **argv)
{
    std::string input_text1="";
    std::cin >> input_text1;
```

```
std::vector<int> vect;
std::vector<int> vect2;
std::stringstream s1(input_text1);
for (int i; s1 >> i;) {
    vect.push_back(i);
    if (s1.peek() == ',')
        s1.ignore();
}
int num_dices=vect[0];
int num_faces=vect[1];
std::string input_text2="";
std::cin >> input_text2;
std::stringstream s2(input_text2);
for (int j; s2 >> j;) {
    vect2.push_back(j);
    if (s2.peek() == ',')
        s2.ignore();
}
if (vect2.size()!=num_dices)
{
    std::cout << "ERROR";
    return -1;
}
for (int k=0;k<vect2.size(); k++)
{
    if (k%2==0)
  {
      vect2[k]=(vect2[k]+3)%num_faces;
     if (vect2[k]==0) vect2[k]=6;
  }
  else
  {
      int extra;
      if (vect2[k]-2 <= 0 )
     {
```

```
                extra=num_faces;
            }
            else
            {
                extra=0;
            }

            vect2[k]=extra+vect2[k]-2;
        }
        std::cout << vect2[k] << " ";
    }
    std::cout << ".";
    return 0;
}
```

16 Power Problems

10 points

Introduction

For a nuclear station, it is required to obtain what is the lowest natural number *n* greater than 1 that is, at the same time a *x* power, a *y* power, and a *z* power of other natural numbers; where *x*, *y*, and *z* are natural numbers greater than 0 and lower than 10.

Exercise

Write a program that reads three different natural numbers between 1 and 9 and outputs the lowest number which complies with all three requisites. For instance if we introduce numbers 1, 2 and 3, the lowest number is 64 because 64 is 64^1, 8^2 and 4^3. If we introduce 2, 3 and 4, the expected number is 4096 because it's 64^2 and 16^3 and 8^4.

Input

Three different integers greater than 0 and lower than 10, i.e., *x*, *y*, and *z*.

Output

The lowest natural number *n* greater than 1 that is equal to *r*^*x*, *s*^*y*, and *t*^*z* respectively, where *r*, *s*, and *t* are natural numbers.

Example 1

Input

1

2

3

Output

64

Example 2

Input

2

3

5

Output

```
1073741824
```

Solutions

Python

```
import sys
def read_number() -> int:
    number = int(input())
    if number <= 0 or number >= 10:
        print("Input number must be greater than 0 and lower than 10.")
        sys.exit()
    return number
def get_gcd(a: int, b: int) -> int:
    if (b == 0):
        return abs(a)
    return get_gcd(b, a % b)
def get_lcm(a: int, b: int) -> int:
    return abs(a * b) / get_gcd(a, b)
def main() -> None:
    powers_lcm = 1
    for _ in range(3):
        powers_lcm = get_lcm(read_number(), powers_lcm)
    lowest_number = 2
    print(int(pow(lowest_number, powers_lcm)))
if __name__ == '__main__':
    main()
```

C++

```
#include <iostream>
#include <cmath>
#include <iomanip>
int read_number() {
    int number;
    std::cin >> number;
    if ((number <= 0) || (number >= 10)) {
```

```
        std::cout << "Input number must be greater than 0 and lower than 10."
<< std::endl;
        exit(0);
    }
    return number;
}
int get_gcd(int a, int b) {
    if (b == 0) return abs(a);
    return get_gcd(b, a % b);
}
int get_lcm(int a, int b) {
    return abs(a * b) / get_gcd(a, b);
}
int main() {
    int powers_lcm = 1;
    for (int i = 0; i < 3; i++) {
        powers_lcm = get_lcm(read_number(), powers_lcm);
    }
    int lowest_number = 2;
    std::cout << std::fixed << std::setprecision(0) <<
        pow(lowest_number, powers_lcm) << std::endl;
    return 0;
}
```

17 The Kaprekar magic constant

10 points

Introduction

The Kaprekar mathematician D. R. Kaprekar (1905-1986) made some amazing findings about digit sum.

Another example of his findings was the "magic constant of Kaprekar": choose any positive 4 digit number (for instance 2435), reorder the digits increasingly and decreasingly and subtract the lower number to the greater. In the example, we should do 5432-2345=3087. Repeat the process with 3087 to construct a series until the same number is obtained twice. This will be the the Kaprekar Magic constant and it's number 6174. ALWAYS.

This is NOT valid for all 4-digit positive numbers. It cannot have more than two equal successive digits. For instance 5566 would be valid, but 5556 would not be valid.

Write a program to identify how many steps needs to be done until number 6174 is obtained. Or if the number is not valid, print "INVALID NUMBER".

Input

The input will be a single line with a number of 4 digits.

Output

The output will be a single number with the number of steps to be followed until 6174 appears in the series. Or "INVALID NUMBER" if the number is not valid.

Example 1

Input

2435

Output

3

Explaination: 5432-2345=3087; 8730-0378=8352; 8532-2358=6174. Therefore, we needed 3 substractions to obtain 6174.

Example 2

Input

```
5556
```

Output

```
INVALID NUMBER
```

Example 3

Input

```
5566
```

Output

```
4
```

Explaination: 6655-5566=1089; 9801-0189=9621; 9621-1296=8352; 8532-2358=6174. Therefore, we needed 4 substractions to obtain 6174.

Solutions

Python

```
import sys
initial = input().strip()
aux = 0
def three_times(string):
    char_count = {}
    for char in string:
        if char in char_count:
            char_count[char] += 1
        else:
            char_count[char] = 1
    for char in char_count:
        if char_count[char] >= 3:
            return True
    return False
if len(initial) > 4 or len(initial) < 4 or not initial:
    print("INVALID NUMBER")
elif three_times(initial):
    print("INVALID NUMBER")
```

```
elif initial.isalpha():
    print("INVALID NUMBER")
else:
    count = 0
    while initial != '6174':
        aux = []
        for i in range(len(initial)):
            aux.append(int(initial[i]))
        dec = sorted(aux, reverse=True)
        inc = sorted(aux)
        d1 = ' '
        d2 = ' '
        for j in range(len(dec)):
            d1 += str(dec[j])
            d2 += str(inc[j])
        initial = str(int(d1)-int(d2))
        count += 1
    print(count)
```

C++

```
#include <string>
#include <iostream>
#include <algorithm>
#include <map>
bool threeTimes(std::string str)
{
    std::map<char, int> char_count;
    for (char c : str)
    {
        char_count[c]++;
    }
    for (const auto &item : char_count)
    {
        if (item.second >= 3)
        {
```

```
            return true;
        }
    }
    return false;
}
int main(int argc, char **argv)
{
    std::string initial;
    std::string iter;
    long num = 0;
    long count = 0;
    int aux = 0;
    std::cin >> initial;
    if ((initial.length() > 4) || (initial.length() < 4) || initial.empty())
    {
        std::cout << "INVALID NUMBER";
        return -1;
    }
    // to take into account the case presented in example 2
    else if (threeTimes(initial))
    {
        std::cout << "INVALID NUMBER";
        return 0;
    }
    if (initial.find_first_not_of("0123456789") != std::string::npos)
    {
        std::cout << "INVALID NUMBER";
        return 0;
    }
    num = stol(initial);
    count = 0;
    while (num != 6174)
    {
        std::string increasing = initial;
        std::sort(increasing.begin(), increasing.end());
```

```
        std::string decreasing = increasing;
        std::reverse(decreasing.begin(), decreasing.end());
        num = stol(decreasing) - stol(increasing);
        initial = std::to_string(num);
        count++;
    }
    std::cout << count << std::endl;
    return 0;
}
```

18 Lychrel

11 points

Introduction

A Lychrel number is a natural number that cannot form a palindrome through the iterative process of repeatedly reversing its digits and adding the resulting numbers.

Some examples of this process are:

- 56 becomes palindromic after one iteration: 56+65 = 121.
- 57 becomes palindromic after two iterations: 57+75 = 132, 132+231 = 363.
- 59 becomes a palindrome after three iterations: 59+95 = 154, 154+451 = 605, 605+506 = 1111
- 89 takes an unusually large 24 iterations to reach the palindrome 8813200023188.

The smallest number that is not known to form a palindrome is 196. It is the smallest Lychrel number candidate.

Exercise

Your are asked to write a program to find its palindrome and determine if a number is a Lychrel number.

If a number has no palindrome solution after more than 500 iterations, it is considered to be a Lychrel number.

Input

Input will consist in a positive number (greater than or equal to zero).

Output

The outputted value will be the obtained palindrome and if it's a Lychrel number, the output will be:

`<NUM> is NOT a Lychrel number!!`

...being *<NUM>* the input number.

Example 1

Input

```
23
```

Output

```
55
```

Example 2

Input

```
89
```

Output

```
8813200023188
```

Example 3

Input

```
196
```

Output

```
196 is NOT a Lychrel number!!
```

Example 4

Input

```
0
```

Output

```
0
```

Example 5

Input

```
2
```

Output

```
4
```

Example 6

Input

```
56
```

Output

```
121
```

Example 7

Input

57

Output

363

Example 8

Input

59

Output

1111

Solutions

Python

```
import sys
def numReverse( nNumber ):
	rem = 0
	while nNumber > 0:
		rem = ( rem * 10 ) + ( nNumber % 10 )
		nNumber = nNumber // 10
	return rem
def isPalindrome( nNumber ):
	return ( nNumber == numReverse( nNumber ) )
def isLychrel( nNumber, iterCount = 500 ):
	temp = nNumber
	for i in range(iterCount):
		rev = numReverse( temp )
		if isPalindrome( rev + temp ):
			print(rev+temp)
			return True
		temp = temp + rev
	return False
if __name__ == "__main__":
	nNumero = int(input(""))
```

```
      if isLychrel(nNumero) is False:
            print(nNumero, 'is NOT a Lychrel number!!')
```

C++

```
#include <iostream>
#include <string>
using namespace std;
/*
  ::   .::::::::::::.
 ,;;   ;;,`;;;```.;;;
,[[[,,,[[[ `]]nnn]]'
"$$$""""$$$  $$$""
 888   "88o 888o
 MMM    YMM YMMMb
  .,-:::::     ...    :::::::-.  .,::::::      .::    .   .:::::::.
:::::::..   .:::::::.
,;;;'````'  .;;;;;;;;.  ;;,    `';,;;;;''''      ';;,  ;;  ;;;' ;;`;;
;;;;``;;;; ;;;`    `
[[[        ,[[     \[[,`[[     [[ [[cccc        '[[, [[, [[' ,[[ '[[,
[[[,/[[[' '[==/[[[[,
`88bo,__,o,"888,_ _,88P 888_,o8P' 888oo,__         "88"888   888   888,888b
"88bo,88b    dP
  "YUMMMMMP" "YMMMMMP"  MMMMP"`   """"YUMMM         "M "M"   YMM   ""` MMMM
"W"  "YMmMY"


*/
long long int numReverse( long long int nNumber )
{
    long long int rem = 0;
    while( nNumber > 0 )
    {
      rem = ( rem * 10 ) + ( nNumber % 10 );
      nNumber = nNumber / 10;
    }
    return rem;
```

```
}
bool isPalindrome( long long int nNumber )
{
    return ( nNumber == numReverse( nNumber ) );
}
bool isLychrel( int nNumber, const int iterCount = 500 )
{
    long long int temp = nNumber;
    long long int rev;
    for( int i = 0; i < iterCount; i++ )
    {
        rev = numReverse( temp );
        if (isPalindrome( rev + temp ))
      {
            std::cout << rev+temp;
            return true;
      }
        temp = temp + rev;
    }
    return false;
}
int main()
{
    int num;
    std::cin >> num;
    if( isLychrel( num ) == false )
    {
        cout << num << " is NOT a Lychrel number!!" << endl;
    }
}
```

19 The Falling Stone

11 points

Introduction

A stone (or any other object with a minimum weight) takes a certain time to fall to the ground depending on the height from which it is thrown.

Assuming that there is no air resistance and the initial velocity is zero, the free fall formula to obtain the time is:

`T = sqrt(2h/g),`

where *h* is the height and *g* is the Earth's gravity, i.e., 9.8 m/s^2.

Exercise

A stone in free fall from certain height takes exactly *n* seconds to travel the **second half of the entire distance**.

Next, write a program that calculates the total height from which the stone initially fell.

Input

An integer representing the seconds.

Output

The total height in meters rounded to two decimal digits.

Example 1

Input

```
2
```

Output

```
228.47
```

Example 2

Input

```
4
```

Output

```
913.9
```

In this case, the decimals are "90" so they get printed as just "9".

The above results come from the formula:

```
n = sqrt(2h/g) - sqrt(h/g)
```

Where *n* is the time in seconds. So if you want you can clear the *h* in that equation and use it for the implementation.

Solutions

Python

```
#! /usr/bin/python3
from math import sqrt
from random import random
def function(h, t):
    return sqrt(2*h/9.8) - sqrt(h/9.8) - t
def derivate_function(h):
    return 0.0661578/sqrt(h)
def Newton_Raphson_step(h, t):
    return h - (function(h, t)/derivate_function(h))
def Newton_Raphson(t):
    h = random()
    old_h = -1
    while abs(h - old_h) > 1e-10:
        old_h = h
        h = Newton_Raphson_step(h, t)
    return h
def main():
    h = Newton_Raphson(int(input()))
    #print(f"{h = }")
    #print(f"{sqrt(2*h/9.8)-sqrt(h/9.8)}")
    print(round(h, 2))
if __name__ == '__main__':
    main()
```

C++

```
#include <iostream>
```

```
#include <cmath>
#include <iomanip>
int read_time() {
    int time;
    std::cin >> time;
    return abs(time);
}
double get_height(int time){
    double h = pow(time, 2) * 9.8 / (3 - 2 * sqrt(2));
    return round(h * 100.0) / 100.0;
}
int main() {
    std::cout << std::setprecision(16);
    std::cout << get_height(read_time()) << std::endl;
    return 0;
}
```

20 A Good Ending for a Good Beginning

12 points

Introduction

Powers of numbers 2, 3, 7 and 8 always end (their last digit) in 4 different numbers, repeating in an infinite fashion. In example, the endings for the powers of 2 are 2, 4, 8 and 6, and then 2, 4, 8 and 6 in an infinite way because powers of 2 are 2, 4, 8, 16, 32, 64, 128, 256, 512, etc. Same happens with 3 (3, 9, 27, 81, 243, 729, 2187, etc), 7 (7, 49, 343, 2401, 16807, 117649, etc) and 8 (8, 64, 512, 4096, 32768, 262144, etc). Thus, for 3 the endings are 3, 9, 7 and 1; for 7 they are 7, 9, 3 and 1; and for 8 they are 8, 4, 2 and 6, repeating up to infinity.

On the other hand, the powers of numbers 0, 1, 5 and 6 always end in that very same digit. For 0 and 1 the results are trivial (0 and 1, respectively). For 5 (5, 25, 125, 615, etc) and 6 (6, 236, 216, 1296, etc) you can easility check it.

Powers of 4 and 9, however, only have two possible endings. For 4 they are 4, 6, 4, 6, etc, and for 9 they are 9, 1, 9, 1, etc, and so on.

Once the numbers go over 10, the same rules apply. For example, powers of 12 will work as powers of 2, ending in 2, 4, 8 and 6 in an infinite fashion. Same happens with 23 (endings of 3), 987 (endings of 7) or 10000 (endings of 0, that is, always 0).

Exercise

You are asked to write a program to calculate in which digit will a given power end. The program will receive two integers, in different lines, being the first one the base and the second one the exponent. The program will output the last digit of the power.

Do not try to calculate the power directly as it will overflow. Instead, calculate the last digit of the power and output it using the rules above.

Input

Two positive integers in different lines, being the first one the base and the second one the exponent.

Output

Last digit of the power.

Example 1

Input

```
17
83578
```

Output

```
9
```

Powers of 7 or 17 end in 7, 9, 3 or 1, repeating in an infinite fashion every 4 powers. Dividing 83578 by 4, the remainder is 2, so the power ends in 9 given the sequence.

Example 2

Input

```
4
50000
```

Output

```
6
```

Solutions

Python

```
def main():
    ends = {
        0: [0],
        1: [1],
        2: [6, 2, 4, 8],
        3: [1, 3, 9, 7],
        4: [6, 4],
        5: [5],
        6: [6],
        7: [1, 7, 9, 3],
        8: [6, 8, 4, 2],
        9: [1, 9]
    }
    number = int(input())
    power = int(input())
    while number >= 9:
```

```
        number = number % 10
    if power < 1:
        print(1)
        return
    rest = power % len(ends[number])
    endVectors = ends[number]
    endDigit = endVectors[rest]
    print(endDigit)
if __name__ == "__main__":
    main()
```

C++

```
#include <map>
#include <vector>
#include <iostream>
int main()
{
    // create the map of possible ends depending on the units of given number
    std::map<int, std::vector<int>> ends =
        {
            {0, {0}},
            {1, {1}},
            {2, {6, 2, 4, 8}},
            {3, {1, 3, 9, 7}},
            {4, {6, 4}},
            {5, {5}},
            {6, {6}},
            {7, {1, 7, 9, 3}},
            {8, {6, 8, 4, 2}},
            {9, {1, 9}}};
    int number = 0;
    int power = 0;
    std::cin >> number;
    std::cin >> power;
    // get the units of given number
```

```
    while (number > 9)
    {
        number = number % 10;
    }
    // if power is 0, result is always 1
    if (power < 1)
    {
        std::cout << 1;
        return 1;
    }
    // compute rest and get the correspondent end for given number
    int rest = power % ends[number].size();
    auto endVectors = ends[number];
    int end = endVectors[rest];
    std::cout << end;
    return 0;
}
```

21 Fill the form

12 points

Introduction

In a Canarian island there's been a volcano eruption. The island's Cabildo has gathered a series of financial aids for residents who have lost their houses. They have asked you to prepare a program to allow residents to calculate their losses.

Unfortunately the financial aids can be no greater than 100000 per citizen, and single goods to be paid cannot be more expensive than 50000.

Exercise

It's necessary to develop a program that allow user to fill a form stating the goods lost under the volcano's lava. The program will receive different number of inputs (different for every user) with a specific format:

```
<REASON>: <AMOUNT OF MONEY>
```

For instance:

```
LA CASA DE LA ABUELA: 47500
```

The input will end with a dot (.) line. Take into account that this is a hand written form so any variable number of spaces can be found before and after the colon (including zero).

As indicated, a good cannot be more expensive than 50000. This will have to be checked. And the total amount of the goods filled in the form cannot sum an amount greather than 100000.

If all the form and the goods comply these constraints, your program will return OK, colon, a space and the total amount of the form. For instance:

```
OK: 99435
```

If the total amount exceeds 100000 of the amount applied to a good exceeds 50000. Your application will return the text:

```
AMOUNT EXCEEDED
```

If the input does not have the appropiate format, your application will end with the text:

```
ERROR
```

Input

It's necessary to develop a program that allow user to fill a form stating the goods lost under the volcano's lava. The program will receive different number of inputs (different for every user) with a specific format:

```
<REASON>: <AMOUNT OF MONEY>
```

The input will end with a dot (.) line. Again, take into account the variable number of spaces, so it could be something like:

```
<REASON>                    :<AMOUNT OF MONEY>
```

Or:

```
<REASON>                    :          <AMOUNT OF MONEY>
```

Or:

```
<REASON>:          <AMOUNT OF MONEY>
```

Or even:

```
<REASON>:<AMOUNT OF MONEY>
```

Output

If all the form and the goods comply these constraints, your program will return OK, colon, a space and the total amount of the form. For instance:

```
OK: 99435
```

If the total amount exceeds 100000 of the amount applied to a good exceeds 50000. Your application will return the text:

```
AMOUNT EXCEEDED
```

If the input does not have the appropiate format, your application will end with the text:

```
ERROR
```

Example 1

Input

```
CASA: 10000
HUERTA: 10000
.
```

Output

```
OK: 20000
```

Example 2

Input

```
Residencia: 10000
Plantio: 3244
Maldito volcan
.
```

Output

```
ERROR
```

...because the last line (before the dot) does not have the appropiate format.

Example 3

Input

```
Residencia: 98700
Platanera: 9244
.
```

Output

```
AMOUNT EXCEEDED
```

Example 4

Input

```
Mi casita: 54300
La autocaravana                 : 19200
.
```

Output

```
AMOUNT EXCEEDED
```

Solutions

Python

```
def main():
    total = 0
    done = False
    while not done:
        line = input().strip()
```

```
        if line == '.':
            done = True
        else:
            parts = [v.strip() for v in line.split(sep=':')]
            if len(parts) != 2:
                print("ERROR")
                quit()
            try:
                amount = int(parts[1])
            except:
                print("ERROR")
                quit()
            if amount > 50000:
                print("AMOUNT EXCEEDED")
                quit()
            total += amount
    if total > 100000:
        print("AMOUNT EXCEEDED")
    else:
        print(f"OK: {total}")
if __name__ == "__main__":
    main()
```

C++

```
#include <stdio.h>
#include <vector>
#include <string>
#include <iostream>
int main()
{
    std::vector<std::pair<std::string, float>> listado;
    std::string inputted = "";
    std::string delimiter = ":";
    do
    {
```

```
        getline(std::cin, inputted);
        if (inputted == ".")
        {
            break;
        }
        std::string token = inputted.substr(0, inputted.find(delimiter));
        std::string amount_str = inputted.substr(inputted.find(delimiter) +
1, inputted.size());
        float valor = 0;
        try
        {
            valor = std::stof(amount_str);
        }
        catch (const std::invalid_argument &e)
        {
            std::cout << "ERROR" << std::endl;
            return 0;
        }
        if (valor < 0)
        {
            std::cout << "ERROR" << std::endl;
            return 0;
        }
        else if (valor > 50000)
        {
            std::cout << "AMOUNT EXCEEDED" << std::endl;
            return 0;
        }
        listado.push_back(std::make_pair(token, valor));
    } while (inputted != ".");
    float total = 0.0;
    for (auto it : listado)
    {
        total += it.second;
    }
```

```
    if (total > 100000)
    {
        std::cout << "AMOUNT EXCEEDED" << std::endl;
        return 0;
    }
    std::cout << "OK: " << total << std::endl;
    return 0;
}
```

22 The McCarthy 91 function

12 points

Introduction

The McCarthy 91 function is defined as follows:

```
        |-> n - 10 (if n > 100)
M91(n)=|
        |-> M91(M91(n+11)) (if n <= 100)
```

This function is special because all numbers under 100 will end up in 91.

Generalizing it, the McCarthy function will be:

```
             |-> n - XX (if n > ZZ)
McCarthy(n)=|
             |-> McCarthy(McCarthy(n+YY)) (if n <= ZZ)
```

Exercise

You are asked to write a program to calculate the McCarthy function of a given value.

Input

The first input line will consist in three numeric positive values XX, YY and ZZ, separated by a comma:

```
XX,YY,ZZ
```

The second input line will consist in a numeric positive number not greater than 9999, which is the value of *n*.

Output

The program will return McCarthy function of the given integer.

Example 1

Input

```
10,11,100
74
```

Output

```
91
```

Example 2

Input

```
10,11,100
774
```

Output

```
764
```

Example 3

Input

```
10,20,299
45
```

Output

```
295
```

Solutions

Python

```
import sys
XX = None
YY = None
ZZ = None
def mc_carthy(number):
    if (number > ZZ):
        return (number - XX)
    else:
        return mc_carthy(mc_carthy(number+YY))
if __name__ == "__main__":
    args = input("").split(',')
    XX = int(args[0])
    YY = int(args[1])
    ZZ = int(args[2])
    number = int(input())
    print(mc_carthy(number))
```

C++

```
#include <string>
#include <iostream>
#include <vector>
#include <sstream>
int XX, YY, ZZ;
int mccarthy(int n)
{
      if (n>ZZ)
            return n-XX;
      else
            return mccarthy(mccarthy(n+YY));
}
int main(int argc, char **argv)
{
    std::vector<int> vect;
    std::string input_text="";
    std::string input_text2="";
    std::cin >> input_text;
    std::stringstream ss(input_text);
    for (int i; ss >> i;) {
        vect.push_back(i);
        if (ss.peek() == ',')
            ss.ignore();
    }
    XX=vect[0];
    YY=vect[1];
    ZZ=vect[2];
    std::cin >> input_text2;
    int input_val=std::stoi(input_text2);

    std::cout<<mccarthy(input_val);
    return 0;
}
```

23 All Mixed Up

13 points

Introduction

We are organizing a dinner for our software developer friends. However, in order to avoid them talking all the night about coding, frameworks and the latest trends in the design of web services, we have also decided to invite some non-developer friends to the dinner. We want to make sure that developers are not sitting together in our long dinning table so they don't talk about the boring stuff.

We want to develop a program that will calculate the number of possible combinations of arrangements of developers and non-developer friends in the table with the restriction stated above. For example, if we have 2 developers and 2 non-developers, the possible arrangements are:

dndn

ndnd

dnnd

So it would be 3 possible arrangements. Take into account that developers and non-developers act as white and black balls, so the same arrangement of developers/non-developers is the same if the first one is Pablo (developer) or Juan (developer).

For 2 developers and 3 non-developers, the possible arrangements are:

dnnnd

dnndn

dndnn

ndndn

ndnnd

nndnd

Hint: this problem can be solved using combinatorics instead of brute force.

Exercise

A program that will take a number of developers and non-developers and will calculate the number of possible arrangements of developers and non-developers in the table with the restriction stated above.

Input

Two values separated by ';'. The first one will be the number of developers and the second one will be the number of non-developers.

Output

Number of possible arrangements of developers and non-developers in the table. If it's not possible to satisfy the restriction, the output will be 0 (and the dinner will be, of course, cancelled).

Example 1

Input

```
2;3
```

Output

```
6
```

Example 2

Input

```
4;8
```

Output

```
126
```

Example 3

Input

```
3;2
```

Output

```
1
```

Solutions

Python

```
def main():
    strInput = input()
    x,y = strInput.split(";")
    x=int(x)
    y=int(y)
    y+=1
```

```
    num=1;den=1
    for _ in range(0,x):
        num*=y
        y-=1
        den*=x
        x-=1
    print(num//den)
if __name__ == "__main__":
    main()
```

C++

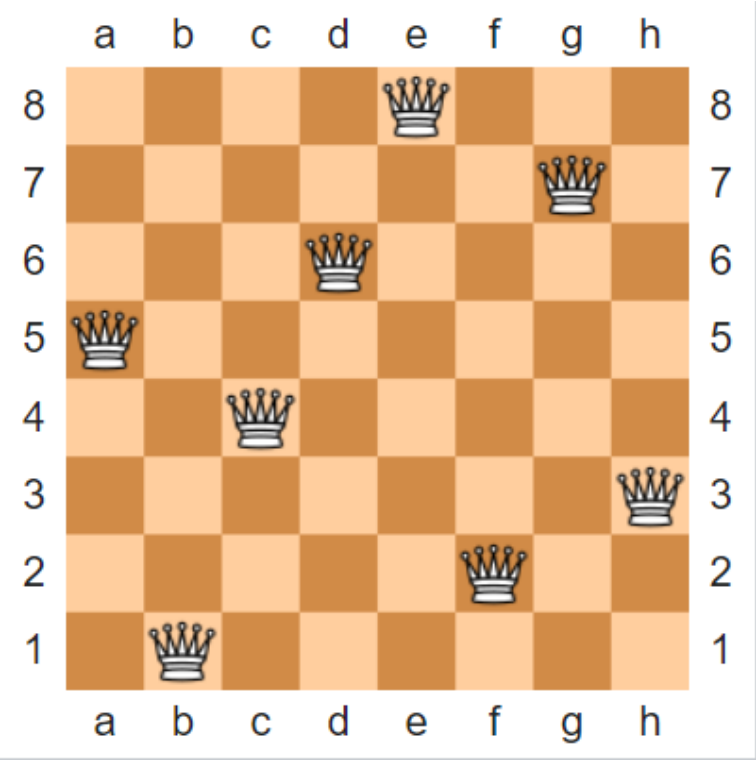
```
#include <iostream>
int main ()
{
    int x=0,y=0;
    auto input = std::scanf("%d;%d",&x,&y);
    int num=1,den=1;
    y++;
    int counter = x;
    for(int i = counter; i > 0; i --)
    {
        num*=y;
        y--;
        den*=x;
        x--;
    }
    std::cout << num/den;
    return 0;
}
```

24 The thousandth queen

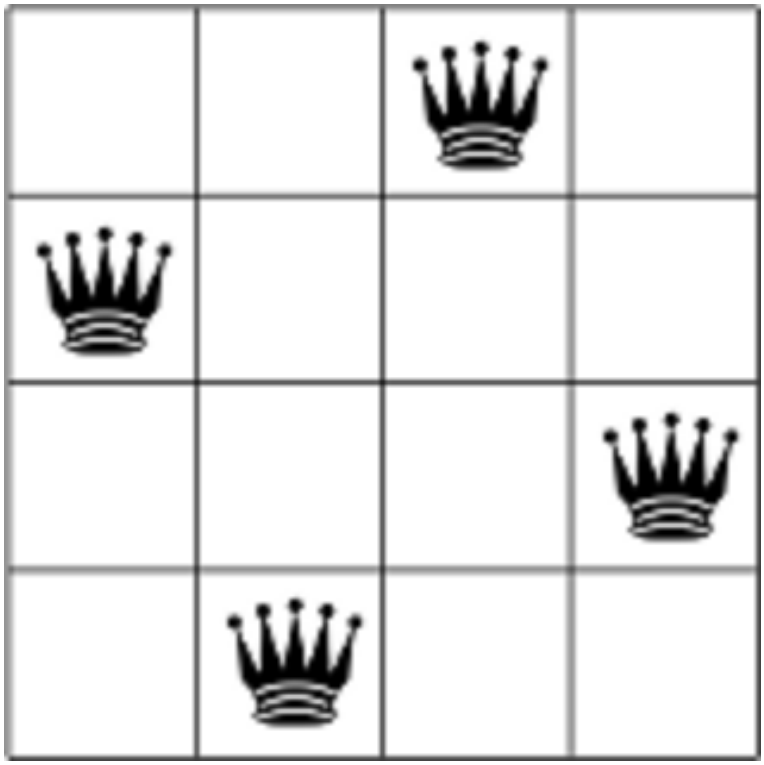
14 points

Introduction

The eight queens puzzle is the problem of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other; thus, a solution requires that no two queens share the same row, column, or diagonal (queens move in every direction as many squares as they want). An example solution is:



This problem can be generalized to the *n-queens* problem where you have to place *n* queens on an *n*×*n* chessboard. An example of a 4-queens solution:



Solving this problem is very expensive in computational terms, however, checking if a solution is valid is quite cheap. If you manage to solve it for a 1000×1000 chessboard (and present a working computer program to do so) the Clay Mathematics Institute in USA will give you... ONE MILLION DOLLARS!

Exercise

You are asked to write a program to **validate** an *n-queens* solution. In order to do so, the program will take as input two lines, the first line will specify the *n* for the number of queens (and the *n*×*n* chessboard) and the second line will be the positions of the queens for every **column** (see columns *a*, *b*, *c*, etc, in first image), using the numbers that start at the bottom of the board as reference.

The input for the 8x8 solution above will be:

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

Then, the output for a valid solution will be:

```
This is a valid solution.
```

However if the solution is not valid, the output should be:

```
This is NOT a valid solution.
```

Input

As specified above, two lines, first one with a positive integer, second one with as many positive integers as the number specified in the first line.

Output

If the solution is valid:

```
This is a valid solution.
```

Otherwise:

```
This is NOT a valid solution.
```

Example 1

Input

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

Output

```
This is a valid solution.
```

Example 2

Input

```
4
3 1 4 2
```

Output

```
This is a valid solution.
```

Example 3

Input

```
2
1 2
```

Output

```
This is NOT a valid solution.
```

Solutions

Python

```
def check_queens(n, chessboard, queens):
    for queen in queens:
        (x, y) = queen
        # check horizontal
        for i in range(n):
            if i == x:
                continue
            if chessboard[i][y]:
                return False
        # check vertical
        for i in range(n):
            if i == y:
                continue
            if chessboard[x][i]:
                return False
        # check main diagonal
        diag_x, diag_y = (x - 1, y - 1)
        while diag_x >= 0 and diag_y >= 0:
            if chessboard[diag_x][diag_y]:
                return False
            diag_x, diag_y = (diag_x - 1, diag_y - 1)
        diag_x, diag_y = (x + 1, y + 1)
        while diag_x < n and diag_y < n:
            if chessboard[diag_x][diag_y]:
                return False
            diag_x, diag_y = (diag_x + 1, diag_y + 1)
        # check other diagonal
        diag_x, diag_y = (x - 1, y + 1)
        while diag_x >= 0 and diag_y < n:
            if chessboard[diag_x][diag_y]:
                return False
            diag_x, diag_y = (diag_x - 1, diag_y + 1)
        diag_x, diag_y = (x + 1, y - 1)
        while diag_x < n and diag_y >= 0:
            if chessboard[diag_x][diag_y]:
```

```
                return False
            diag_x, diag_y = (diag_x + 1, diag_y - 1)
    return True
def main():
    n = int(input())
    positions = [int(v) for v in input().split()][:n]
    chessboard = [[False]*n for i in range(n)]
    # place the queens (as True) and save their positions for easier checking
    column = 0
    queens = []
    for position in positions:
        x = column
        y = n - position
        queens.append((x, y))
        chessboard[x][y] = True
        column += 1
    valid = check_queens(n, chessboard, queens)
    if valid:
        print("This is a valid solution.")
    else:
        print("This is NOT a valid solution.")
if __name__ == "__main__":
    main()
```

C++

```
#include <iostream>
#include <vector>
bool checkQueens(
    const int n,
    const std::vector<std::vector<bool>>& chessboard,
    const std::vector<std::pair<int, int>>& queens)
{
    for (std::pair<int, int> const& queen: queens)
    {
        const int x = queen.first;
```

```
const int y = queen.second;
// check horizontal
for (int i = 0; i < n; i++)
{
    if (i == x)
        continue;
    if (chessboard[i][y])
        return false;
}
// check vertical
for (int i = 0; i < n; i++)
{
    if (i == y)
        continue;
    if (chessboard[x][i])
        return false;
}
// check main diagonal
int diagX = x - 1;
int diagY = y - 1;
while (diagX >= 0 && diagY >= 0)
{
    if (chessboard[diagX][diagY])
        return false;
    diagX--;
    diagY--;
}
diagX = x + 1;
diagY = y + 1;
while (diagX < n && diagY < n)
{
    if (chessboard[diagX][diagY])
        return false;
    diagX++;
    diagY++;
```

```
        }
        // check other diagonal
        diagX = x - 1;
        diagY = y + 1;
        while (diagX >= 0 && diagY < n)
        {
            if (chessboard[diagX][diagY])
                return false;
            diagX--;
            diagY++;
        }
        diagX = x + 1;
        diagY = y - 1;
        while (diagX < n && diagY >= 0)
        {
            if (chessboard[diagX][diagY])
                return false;
            diagX++;
            diagY--;
        }
    }
    return true;
}
int main()
{
    int n;
    std::cin >> n;
    std::vector<int> positions;
    for (int i = 0; i <n; i++)
    {
        int pos;
        std::cin >> pos;
        positions.push_back(pos);
    }
    std::vector<std::vector<bool>> chessboard(n, std::vector<bool>(n));
```

```
    // place the queens (as true) and save their positions for easier
checking
    std::vector<std::pair<int, int>> queens;
    int column = 0;
    for (int const& position: positions)
    {
        const int x = column;
        const int y = n - position;
        queens.push_back({ x,y });
        chessboard[x][y] = true;
        column++;
    }
    const bool valid = checkQueens(n, chessboard, queens);
    std::cout << "This is " << (valid ? "" : "NOT ") << "a valid solution."
<< std::endl;
}
```

Java

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
public class Queen {
    static boolean checkQueens(
        final int n,
        final boolean[][] chessboard,
        final List<int[]> queens
    ) {
        for (int[] queen : queens) {
            final int x = queen[0];
            final int y = queen[1];
            // check horizontal
            for (int i = 0; i < n; i++) {
                if (i == x) {
                    continue;
                }
```

```
    if (chessboard[i][y]) {
        return false;
    }
}
// check vertical
for (int i = 0; i < n; i++) {
    if (i == y) {
        continue;
    }
    if (chessboard[x][i]) {
        return false;
    }
}
// check main diagonal
int diagX = x - 1;
int diagY = y - 1;
while (diagX >= 0 && diagY >= 0) {
    if (chessboard[diagX][diagY]) {
        return false;
    }
    diagX--;
    diagY--;
}
diagX = x + 1;
diagY = y + 1;
while (diagX < n && diagY < n) {
    if (chessboard[diagX][diagY]) {
        return false;
    }
    diagX++;
    diagY++;
}
// check other diagonal
diagX = x - 1;
diagY = y + 1;
```

```
            while (diagX >= 0 && diagY < n) {
                if (chessboard[diagX][diagY]) {
                    return false;
                }
                diagX--;
                diagY++;
            }
            diagX = x + 1;
            diagY = y - 1;
            while (diagX < n && diagY >= 0) {
                if (chessboard[diagX][diagY]) {
                    return false;
                }
                diagX++;
                diagY--;
            }
        }
        return true;
    }
    public static void main(final String[] args) {
        final Scanner sc = new Scanner(System.in);
        final int n = sc.nextInt();
        final List<Integer> positions = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            positions.add(sc.nextInt());
        }
        final boolean[][] chessboard = new boolean[n][n];
        // place the queens (as true) and save their positions for easier
checking
        final List<int[]> queens = new ArrayList<>();
        int column = 0;
        for (int position : positions) {
            final int x = column;
            final int y = n - position;
            queens.add(new int[]{ x, y });
```

```
            chessboard[x][y] = true;
            column++;
        }
        final boolean valid = checkQueens(n, chessboard, queens);
        // std::cout << "This is " << (valid ? "" : "NOT ") << "a valid
solution." << std::endl;
        System.out.println("This is " + (valid ? "" : "NOT ") + "a valid
solution.");
    }
}
```

25 Factorizatorrr

16 points

Introduction

The NASA (National Aeronautics and Space Administration) has sent a rocket to space which will hit meteorite. The objective is to reduce the mass of the meteorite to sections no bigger than 1 Ton of mass. After thorough studies they've found that if the rocket is sent with a determinate speed and angle, the meteorite breaks in different parts with different masses following a factorization model, but in reverse order.

Let's imagine, a meteorite has a mass of 380 Tons. The factors of 380 are 19, 5, 2 and 2. A first hit will split the meteorite in two parts, a second hit will split each part in another 2, a third hit will split each resulting section in 5 parts, and the last hit will split each resulting part in 19 1-Ton rocks.

Exercise

You are asked to develop a program that receives the mass of the meteorite in Tons (this will be an integer number) and finds all its factors.

Input

The program will receive a unique positive number which will be factorized.

Output

Depending on the input number, the program output may be different:

If number is 0 or under zero, the output will be:

```
Error!! There are no meteorites of this size.
```

If the number is a prime, the output will be:

```
The meteorite is made of Kryptonite!!! We all are going to die!!!!
```

Otherwise the output will consist in the input number followed by an equals symbol and the divisors of number starting from the greatest to the lowest with the format:

```
number^exponent
```

For example, for an input number 2920, the output will be:

```
2920 = 73^1 * 5^1 * 2^3
```

Take note of the spaces, and take note that the divisors start from the greatest (73) to the lowest (2), independently of the exponent value.

Example 1

Input

```
7290
```

Output

```
7290 = 5^1 * 3^6 * 2^1
```

Example 2

Input

```
1234567890
```

Output

```
1234567890 = 3803^1 * 3607^1 * 5^1 * 3^2 * 2^1
```

Example 3

Input

```
11
```

Output

```
The meteorite is made of Kryptonite!!! We all are going to die!!!!
```

Example 4

Input

```
0
```

Output

```
Error!! There are no meteorites of this size.
```

Solutions

Python

```
def main():
    n = int(input())
    if n <= 0:
        print("Error!! There are no meteorites of this size.")
        quit()
```

```
    else:
        factors = []
        remainder = n
        max = n  # this is the maximum up we go, however when finding factors
it decreases, see break below
        for i in range(2, max):
            if i > max:
                break
            while(remainder % i == 0):
                factors.append(i)
                # update max, that's why we don't go up to n//2 or anything
like that, this will make it fast enough
                max = remainder // i
                remainder //= i
    # reverse the list, they are ordered in ascending order
    factors.reverse()
    if (len(factors) <= 1):
        print("The meteorite is made of Kryptonite!!! We all are going to
die!!!!")
    else:
        print(f"{n} = ", end='')
        is_first = True
        previous_factor = None
        for factor in factors:
            if factor == previous_factor:
                continue
            if not is_first:
                print(" * ", end='')
            print(f"{factor}^{factors.count(factor)}", end='')
            is_first = False
            previous_factor = factor
        print("")  # just the end of the line
if __name__ == "__main__":
    main()
```

C++

```
#include <stdio.h>
#include <stdlib.h>
#define YES 1
#define NO 0
int anadirFactor(int divisor, int *NDivisors, int **factors, int **exponents)
{
    static int lastElement = 0;
    if (lastElement == 0 || (divisor != (*factors)[lastElement - 1]))
    {
        lastElement++;
        *factors = (int *)realloc(*factors, (sizeof(int) * (lastElement)));
        *exponents = (int *)realloc(*exponents, (sizeof(int) *
(lastElement)));
        (*factors)[lastElement - 1] = divisor;
        (*exponents)[lastElement - 1] = 1;
        (*NDivisors)++;
    }
    else
    {
        (*exponents)[lastElement - 1] += 1;
    }
    return ((*factors) == NULL || (*exponents) == NULL);
}
int isPrime(int target)
{
    int testingNumber = 0;
    char isPrime = YES;
    if (target == 1)
    {
        return YES;
    }
    for (testingNumber = 2; testingNumber < target; testingNumber++)
    {
```

```
        if (target % testingNumber == 0)
        {
            return NO;
        }
    }
    return YES;
}
int main()
{
    int isDivisor = 0;
    int target = 0;
    int number = 0;
    int NDivisors = 0;
    int *factors = NULL;
    int *exponents = NULL;
    scanf("%d", &target);
    number = target;
    if (!target || number <= 0)
    {
        printf("Error!! There are no meteorites of this size.");
        return 0;
    }

    if (isPrime(target))
    {
        printf("The meteorite is made of Kryptonite!!! We all are going to
die!!!!");
        return 0;
    }
    for (isDivisor = 2; isDivisor <= target || target != 1; isDivisor++)
    {
        if (isPrime(isDivisor))
        {
            while (target % isDivisor == 0)
            {
```

```
                if (anadirFactor(isDivisor, &NDivisors, &factors,
&exponents))
                {
                    printf("\nMemory Error");
                }
                target /= isDivisor;
            }
        }
    }
    printf("%d = ", number);
    for (; NDivisors > 0; NDivisors--)
    {
        printf("%i^%i", factors[NDivisors - 1], exponents[NDivisors - 1]);
        if (NDivisors > 1)
        {
            printf(" * ");
        }
    }
    return 0;
}
```

26 Premium bank accounts

17 points

Introduction

In an Internet bank thousands of operations are being performed every day. Since certain customers do business more actively than others, some of the bank accounts occur many times in the list of operations. The bank owners want to know which are these "premium" accounts. Your task is to sort the bank account numbers in ascending order. Write the number of ocurrences right after the account number.

The format of accounts is as follows: 2 digits stating the country code, a 4-digit code of the bank, and 14 digits identifying the branch office, control code and finally the bank account, for example (at the end of each line there is exactly one space):

```
30 1010 22123361600142
```

Exercise

You are asked to write a program that receives blocks of accounts by operation indicating the accounts that made an entry or withdrawal operation, gathers all the operations and sort the bank accounts indicating the number of operations a bank account has performed.

Input

The program wil receive an input consisting of:

```
t [the number of seconds for the operations]
nX [the number of accounts in second X]
[list of accounts]
nY [the number of accounts in second Y]
[list of accounts]
...
```

For instance a valid input may be:

```
3                       <--- 3 seconds in this batch
2                       <--- In the first second, there two bank account with
operations
03 3538 22123361600142 <---
03 3538 22123361600141 <---  The bank accounts for this second
```

```
4                       <--- In the second second, there are four bank account
operations
30 3538 22123361600141
30 3538 22123361600142
30 3538 22123361600141
30 3538 22123361600142
5                       <--- In the third second, there are five operations
30 3538 22123361600144
30 3538 22123361600142
30 3538 22123361600145
30 3538 22123361600146
30 3538 22123361600143
```

Output

If the input is incorrect by any reason:

- The bank accounts do not have the appropiate format
- The receive blocks do not have the indicated number of accounts
- The program did not receive the appropiate number of seconds
- Any other format error that may occur

....in these cases the program will return the text:

```
ERROR
```

Otherwise, the program will return the whole list of accounts sorted ascending by number of operations, **if two accounts have the same number of operations they will appear sorted by account number**.

The format of the output will be:

```
<BANK ACCOUNT> --- <TOTAL NUMBER OF OPERATIONS>
```

So, in the previous example, the output would be:

```
03 3538 22123361600141 --- 1
03 3538 22123361600142 --- 1
30 3538 22123361600143 --- 1
30 3538 22123361600144 --- 1
30 3538 22123361600145 --- 1
```

```
30 3538 22123361600146 --- 1
30 3538 22123361600141 --- 2
30 3538 22123361600142 --- 3
```

Example 1

Input

```
3
2
03 3538 22123361600142
03 3538 22123361600141
4
30 3538 22123361600141
30 3538 22123361600142
30 3538 22123361600141
30 3538 22123361600142
5
30 3538 22123361600144
30 3538 22123361600142
30 3538 22123361600145
30 3538 22123361600146
30 3538 22123361600143
```

Output

```
03 3538 22123361600141 --- 1
03 3538 22123361600142 --- 1
30 3538 22123361600143 --- 1
30 3538 22123361600144 --- 1
30 3538 22123361600145 --- 1
30 3538 22123361600146 --- 1
30 3538 22123361600141 --- 2
30 3538 22123361600142 --- 3
```

Example 2

Input

```
2
```

```
2
04 3538 56721233616142
4
30 3538 32421233616141
30 3538 32321233616142
30 3538 32231233616141
```

Output

```
ERROR
```

Example 3

Input

```
2
4
04 3538 56721233616142
04 3538 56721233616142
04 3538 56721233616142
04 3538 56721233616142
4
30 3538 32421233616141
30 3538 32231233616141
30 3538 32421233616141
30 3538 32231233616141
```

Output

```
30 3538 32231233616141 --- 2
30 3538 32421233616141 --- 2
04 3538 56721233616142 --- 4
```

Solutions

Python

```
from sys import stdin
def validate_number(num):
    try:
        return int(num)
    except:
```

```
        return None
def validate_account(account):
    split_account = [part.strip() for part in account.split(" ")]
    n_parts = len(split_account)
    # I'm going to be lenient with the extra space at the end
    if n_parts > 4 or n_parts < 3:
        return None
    # if it has the extra space validate that it's that and not anything else
    if len(split_account) == 4 and split_account[3] != '':
        return None
    # validate length of numbers
    if len(split_account[0]) != 2 or len(split_account[1]) != 4 or
len(split_account[2]) != 14:
        return None
    # validate they are numbers
    parts = [validate_number(part) for part in split_account[0:3]]
    if any(not part for part in parts):
        return None
    # finally return the account in comparable format
    return " ".join(split_account[0:3])
def error():
    print("ERROR")
    quit()
def main():
    expected_seconds = validate_number(input())
    if not expected_seconds:
        error()
    seconds = 0
    reading_accounts = False
    read_accounts = 0
    accounts = {}
    for line in stdin:  # we have to do it like this so we allow errors on
invalid number or number of seconds/accounts
        if not reading_accounts:
            expected_accounts = validate_number(line)
```

```
            if not expected_accounts:
                error()
            seconds += 1
            reading_accounts = True
            read_accounts = 0
        else:
            account = validate_account(line)
            if not account:
                error()
            if account in accounts:
                accounts[account] += 1
            else:
                accounts[account] = 1
            # if there are less than expected and another second comes it'll
fail when validating it's an account,
            # if there are more than expected it'll switch to read a second
and will fail too
            read_accounts += 1
            if read_accounts == expected_accounts:
                reading_accounts = False
    # validate last iteration
    if read_accounts != expected_accounts:
        error()
    if seconds != expected_seconds:
        error()
    # now print the results
    accounts_to_print = sorted([(k, v) for k, v in accounts.items(
    )], key=lambda acc: (acc[1], int(acc[0].replace(" ", ""))))
    for account in accounts_to_print:
        print(f"{account[0]} --- {account[1]}")
if __name__ == "__main__":
    main()
```

C++

```
#include <string>
```

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <iterator>
#include <map>
bool isNumber(const std::string &s)
{
    for (char const &c : s)
    {
        if (std::isdigit(c) == 0)
            return false;
    }
    return true;
}
int main(int argc, char **argv)
{
    std::string timestr = "";
    std::cin >> timestr;
    std::vector<std::string> operations;
    if (!isNumber(timestr))
    {
        std::cout << "ERROR" << std::endl;
        return 0;
    }
    int time_val = std::stoi(timestr);
    for (int i = 0; i < time_val; i++)
    {
        std::string numoperationstr = "";
        std::cin >> numoperationstr;
        if (!isNumber(numoperationstr))
        {
            std::cout << "ERROR" << std::endl;
            return 0;
        }
        int num_operations = std::stoi(numoperationstr);
```

```
        getline(std::cin, timestr);
        for (int j = 0; j < num_operations; j++)
        {
            std::string operation = "";
            getline(std::cin, operation);

            operation.erase(std::remove(operation.begin(), operation.end(),
'\r'), operation.end());
            operation.erase(std::remove(operation.begin(), operation.end(),
'\n'), operation.end());
            std::string original_operation = operation;
            operation.erase(std::remove(operation.begin(), operation.end(), '
'), operation.end());

            if ((operation.size() != 20) || !isNumber(operation))
            {
                std::cout << "ERROR" << std::endl;
                return 0;
            }
            else
            {
                operations.push_back(original_operation);
            }
        }
    }
    std::sort(operations.begin(), operations.end());
    std::vector<std::pair<int, std::string>> mapa;
    std::string old = "";
    int count = 0;
    for (auto x : operations)
    {
        if (x != old && old != "")
        {
            mapa.push_back(std::pair<int, std::string>(count, old));
            count = 1;
```

```
        }
        else
        {
            count++;
        }
        old = x;
    }
    mapa.push_back(std::pair<int, std::string>(count, old));
    std::sort(mapa.begin(), mapa.end());
    for (std::pair<int, std::string> i : mapa)
    {
        std::cout << i.second << " --- " << i.first << std::endl;
    }
    return 0;
}
```

27 Digital Panoramix

18 points

Introduction

In the 22nd century, druids like those of the ancient germanic folks have become fashionable again and they leave their messages for posterity on digital rocks.

In a small town in Gaul in the 22nd century, a druid wants to leave messages forming spirals, because the spirals is a symbol for black holes and it's clear that the future of our society lies in travelling through black holes.

However, due to his age, the druid has a hard time writing spiral messages. Luckily, in his town a young digital warrior, with blonde hair to be exact, has the ability to encode the messages that are recorded in the digital rocks, although since he is a warrior and not a computer scientist he needs some help.

Exercise

You are asked to write a program that receives a message of at least 15 characters and writes it in a spiral in such a way that each section of the spiral has a length of one character less than the previous one (see examples below). If the message doesn't have one of the perfect lengths to create a spyral (15, 21, 28, 36, etc), it will be padded with dots ('.') at the end of the string.

Input

The string to be encoded in a spiral.

Output

The output spiral with spaces for the empty positions.

Example 1

Input

```
AAAAAABBBBBCCCCDDDEEF
```

Notice that this is 6 A's, 5 B's, 4 C's, etc.

Output

```
AAAAAA
     B
```

```
DEE B
D F B
D   B
CCCCB
```

Example 2

Input

```
Solo se que no se nada
```

Output

```
Solo se


a... q
d  . u
a .. e
n
 es on
```

The input string length was 22, so it had to be padded with dots up to 28.

Example 3

Input

```
En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho
tiempo que vivia un hidalgo de los de lanza
```

Output

```
En un lugar de
              l
 ordarme, no  a
 c          h
 a un hidal a M
          g   a
 o a anza o m n
 r i l  .   u c
 e v   .. d c h
 i i e    e h a
 u v d sol  o ,
```

```
 q
   euq opmeit d
 o            e
 n erbmon oyuc
```

For a length of 117, the size of the drawing is 15x15. And 3 dots need to be added up to 120. Think about those two numbers and how they are calculated.

Solutions

Python

```
def calculate_lengths(sentence_length):
    length = 1
    total = 0
    while(total < sentence_length):
        total += length
        length += 1
    return length - 1, total
def print_spyral(matrix, long_arm_size):
    for x in range(long_arm_size):
        for y in range(long_arm_size):
            print(matrix[x][y], end='')
        print()
def main():
    sentence = input().strip()
    long_arm_size, sentence_length = calculate_lengths(len(sentence))
    padded_sentence = sentence.ljust(sentence_length, '.')
    matrix = [[' ' for _ in range(long_arm_size)]
              for _ in range(long_arm_size)]
    x = 0
    y = 0
    x_delta = 1
    y_delta = 0
    current_arm = long_arm_size
    pending_arm = current_arm
    for i in range(sentence_length):
        matrix[y][x] = padded_sentence[i]
```

```
        pending_arm -= 1
        if pending_arm == 0:
            current_arm -= 1
            pending_arm = current_arm
            if x_delta > 0:
                x_delta = 0
                y_delta = 1
            elif x_delta < 0:
                x_delta = 0
                y_delta = -1
            elif y_delta > 0:
                x_delta = -1
                y_delta = 0
            elif y_delta < 0:
                x_delta = 1
                y_delta = 0
        x += x_delta
        y += y_delta
    print_spyral(matrix, long_arm_size)
if __name__ == "__main__":
    main()
```

C++

```
#include <iostream>
#include <string>
#include <vector>
int main(void)
{
    std::string sentence;
    std::getline(std::cin, sentence);
    sentence.erase(sentence.find_last_not_of(" \t") + 1);
    // calculate values
    int length = 1;
    int total = 0;
    while (total < sentence.length())
```

```
    {
        total += length;
        length++;
    }
    int longArmSize = length - 1;
    int sentenceLength = total;
    // pad string
    sentence.append(sentenceLength - sentence.length(), '.');
    // initialize matrix
    std::vector<std::vector<char>> matrix(longArmSize,
std::vector<char>(longArmSize));
    for (int i = 0; i < longArmSize; i++)
    {
        for (int j = 0; j < longArmSize; j++)
        {
            matrix[i][j] = ' ';
        }
    }
    // put the characters
    int x = 0;
    int y = 0;
    int xDelta = 1;
    int yDelta = 0;
    int currentArm = longArmSize;
    int pendingArm = currentArm;
    for (int i = 0; i < sentenceLength; i++)
    {
        matrix[y][x] = sentence[i];
        pendingArm--;
        if (pendingArm == 0)
        {
            currentArm--;
            pendingArm = currentArm;
            if (xDelta > 0)
            {
```

```
                xDelta = 0;
                yDelta = 1;
            }
            else if (xDelta < 0)
            {
                xDelta = 0;
                yDelta = -1;
            }
            else if (yDelta > 0)
            {
                xDelta = -1;
                yDelta = 0;
            }
            else if (yDelta < 0)
            {
                xDelta = 1;
                yDelta = 0;
            }
        }
        x += xDelta;
        y += yDelta;
    }
    // print the result
    for (int i = 0; i < longArmSize; i++)
    {
        for (int j = 0; j < longArmSize; j++)
        {
            std::cout << matrix[i][j];
        }
        std::cout << std::endl;
    }
    return 0;
}
```

28 Intergalactic Excel

19 points

Introduction

Elon Musk has decided to create an interplanetary accounting system and he wants to use Excel for it. However, as the bitrate in interplanetary communications is quite scarce, he has decided to dispense with a graphical interface and he wants to implement it in a simplified way and in text mode.

Furthermore, as the guy is short on money, he has decided to ask for help from some kids from the high school of his friend's nieces and it turns out that you are those kids.

Exercise

You are asked to write a program that reads 5 lines of text simulating an Excel spreadsheet. As you know, each cell of an Excel spreadsheet is numbered according to its row and column. So for example, cell C2 corresponds to column 3 and row 2. In the Excel that is to be implemented, it is the same. The cells are separated by commas and for this proof of concept, it will only have 5 rows and 5 columns.

Cells can be empty, have numbers or have operations. For this proof of concept, the operations will be sums, subtractions or multiplications.

Output should be the same Excel sheet received, but with the operations solved. For example:

```
0,4,5,,
3+1,,5*4,,
,,,,
7*3,,,,
,,1-3,,
```

The output should be:

```
0,4,5,,
4,,20,,
,,,,
21,,,,
,,-2,,
```

However, things are not going to be that easy. Mr. Musk is also asking us for the Excel to be able to do "formulas". That is, in the operations we must be able to reference other cells. So if a cell appears as *$XY*, where X is a letter and Y is a number, we must read the referenced cell and substitute that *$XY* for the number referenced. For example:

```
0,4,5,,
3+6,,$A1*$A2,,
$A2*2,,,,5+$A2
,,,,
,$B1,,,
```

Should have this output:

```
0,4,5,,
9,,0,,
8,,,,9
,,,,
,9,,,
```

As said, for cells the operations will also be sum, substraction and multiplication. Also, the maximum number of cells that can be referenced is 2, but they can also appear involved in an operation with an escalar number (see above) or by themselves, just referencing the value of the cell.

Another limitation is that **there will be no chain references**. If a cell references another cell, that cell will either have a value or an operation between scalars, but not a reference to another cell.

Input

Five lines of text with the Excel spreadsheet, containing empty cells, values and operations, organized in 5 columns separated by commas.

Output

Five lines of text with the Excel spreadsheet with the operations solved and formulas resolved, organized in 5 columns separated by commas.

Example 1

Input

```
0,4,5,,
3+1,,5*4,,
```

```
,,,,
7*3,,,,
,,1-3,,
```

Output

```
0,4,5,,
4,,20,,
,,,,
21,,,,
,,-2,,
```

Example 2

Input

```
0,4,5,,
3+6,,$A1*$A2,,
$A2*2,,,,5+$A2
,,,,
,$B1,,,
```

Output

```
0,4,5,,
9,,0,,
8,,,,9
,,,,
,9,,,
```

Solutions

Python

```
import re
def get_numbers_operation(value):
    try:
        match = re.search(r'^(\d+)([\+\-\*])(\d+)$', value)
    except:
        return None, None, None
    if match:
        return match.groups()
```

```
    else:
        return None, None, None
def calculate_operation(n1, op, n2):
    if op == '+':
        return n1 + n2
    elif op == '-':
        return n1 - n2
    elif op == '*':
        return n1 * n2
def evaluate_operations(spreadsheet):
    # this mutates spreadsheet
    for row in range(5):
        for column in range(5):
            value = spreadsheet[row][column]
            n1, op, n2 = get_numbers_operation(value)
            if n1 and op and n2:
                spreadsheet[row][column] = calculate_operation(
                    int(n1), op, int(n2))
    return spreadsheet
def replace_references_single(spreadsheet):
    # this mutates spreadsheet
    for row in range(5):
        for column in range(5):
            value = spreadsheet[row][column]
            if isinstance(value, str):
                match = re.search(r'(.*)(\$[A-E])(\d+)(.*)', value)
                if match:
                    column_ref = match.group(2)[1:]  # remove the $
                    column_index = ord(column_ref) - ord('A')
                    row_index = int(match.group(3)) - 1
                    spreadsheet[row][column] =
f"{match.group(1)}{spreadsheet[column_index][row_index]}{match.group(4)}"
    return spreadsheet
def print_spreadsheet(spreadsheet):
    for row in range(5):
```

```
        for column in range(5):
            print(spreadsheet[row][column], end=(',' if column < 4 else ''))
        print()
def main():
    spreadsheet = []
    # read the spreadsheet and store the raw values
    for _ in range(5):
        line = input().strip()
        values_in_line = line.split(',')
        spreadsheet.append(values_in_line)
    # first pass: evaluate simple operations
    spreadsheet = evaluate_operations(spreadsheet)
    # second pass: replace references
    spreadsheet = replace_references_single(spreadsheet)
    # third pass: replace references again (because one formula can involve
two references)
    spreadsheet = replace_references_single(spreadsheet)
    # fourth pass: evaluate operations resulting from formulas
    spreadsheet = evaluate_operations(spreadsheet)
    # now just print it
    print_spreadsheet(spreadsheet)
if __name__ == '__main__':
    main()
```

29 Diophantine Equations

20 points

Introduction

Diophantine equations are simply algebraic equations whose coefficients and solutions, if they exist, are integer numbers.

In particular, we are interested in the classical example: the lineal diophantine equation with two unknows, i.e., $ax + by = c$ where a, b and c, and the unknown solutions (if they exist) x and y are integer numbers. In this case, the equation has solution if the greatest common divisor of a and b divides c, and when so the solutions are infinite.

For instance:

$3x + 2y = -1$

has a particular solution $x\_p = -1$ and $y\_p = 1$. Then, the general solution is:

x = x_p + lambda $b / d = -1 + lambda$ 2 / 1

y = y_p - lambda $a / d = 1 - lambda$ 3 / 1

where d is the greatest common divisor of a and b, and *lambda* is any integer number. Therefore $x = -3$ and $y = 4$ (when *lambda* $= -1$); $x = 5$ and $y = -8$ (when *lambda* $= 3$), etc. are also solutions.

Exercise

Write a program that, for a given lineal diophantine equation with two unknows $ax + by = c$ where a, b and c are different from zero, finds the particular solution closest to zero (i.e., when the sum of the absolute solution values is smaller: $abs(x) + abs(y)$) if it exists. In case there is more than one print them sorted. Otherwise, when the equation has no solution then print the following message:

There are no solutions.

Input

Three int numbers different from zero separated by `;` which correspond to a, b and c respectively:

```
a;b;c
```

Output

The particular solution closest to zero:

`x;y`

or the sorted particular solutions when there are more than one:

```
x0;y0
x1;y1
[...]
```

In case the equation has no solution then:

```
There are no solutions.
```

Example 1

Input

```
1;1;-1
```

Output

```
-1;0
0;-1
```

Example 2

Input

```
162;-508;5123866
```

Output

```
45;-10072
```

Solutions

Python

```
def main() -> None:
    a, b, c = read_input()
    d = get_gcd(a, b)
    if c % d != 0:
        print("There are no solutions.")
        return
    a, b, c = a // d, b // d, c // d
    x0, y0, x1, y1 = solve_positive_equation(a, b, c)
    x0, y0 = add_signs_to_solution(x0, y0, a, b, c)
    solutions = [f"{x0};{y0}"]
```

```
    if a == b: solutions.append(f"{y0};{x0}")
    elif a == -1 * b: solutions.append(f"{-1 * y0};{x0}")
    if x1 != None and y1 != None and x1 != 0 and y1 != 0:
        x1, y1 = add_signs_to_solution(x1, y1, a, b, c)
        solutions.append(f"{x1};{y1}")
    for s in sorted(solutions): print(s)
def read_input() -> list:
    input_elements = input()
    assert ';' in input_elements
    input_numbers = [int(i) for i in input_elements.split(';')]
    assert len(input_numbers) == 3
    for i in input_numbers: assert i != 0
    return input_numbers
def get_gcd(a: int, b: int) -> int:
    if (b == 0): return abs(a)
    return get_gcd(b, a % b)
def solve_positive_equation(a: int, b: int, c: int) -> tuple:
    a, b, c = abs(a), abs(b), abs(c)
    x0, y0 = get_euler_particular_solution(a, b, c)
    x0, y0, x1, y1 = get_smaller_solution(x0, y0, a, b)
    return x0, y0, x1, y1
def get_euler_particular_solution(a: int, b: int, c: int) -> tuple:
    for i in range(abs(b)):
        if (c - a * i) % b == 0:
            y = (c - a * i) // b
            x = (c - b * y) // a
    return x, y

def get_smaller_solution(x0: int, y0: int, a: int, b: int) -> tuple:
    _lambda = -1 if x0 >= y0 else 1
    sum_abs = lambda x, y: abs(x) + abs(y)
    while True:
        x1, y1 = get_another_particular_solution(x0, y0, a, b, _lambda)
        if sum_abs(x0, y0) > sum_abs(x1, y1):
            x0, y0 = x1, y1
```

```
            x1, y1= None, None
        elif sum_abs(x0, y0) == sum_abs(x1, y1): break
        else:
            x1, y1 = None, None
            break
    return x0, y0, x1, y1
def get_another_particular_solution(x: int, y: int, a: int, b: int, _lambda:
int) -> tuple:
    x = x + _lambda * b
    y = y - _lambda * a
    return x, y

def add_signs_to_solution(x: int, y: int, a: int, b: int, c: int) -> tuple:
    sing_a, sing_b, sing_c = a // abs(a), b // abs(b), c // abs(c)
    x *= sing_a * sing_c
    y *= sing_b * sing_c
    return x, y
if __name__ == '__main__':
    main()
```

C++

```
#include <iostream>
#include <cmath>
#include <assert.h>
#include <tuple>
#include <string>
#include <algorithm>
int get_gcd(int a, int b){
    if (b == 0){
        return abs(a);
    }
    return get_gcd(b, a % b);
}
std::tuple<int, int> get_euler_particular_solution(int a, int b, int c){
    int x, y;
```

```
    for (int i = 0; i < abs(b); i++){
        if ((c - a * i) % b == 0){
            y = (int) (c - a * i) / b;
            x = (int) (c - b * y) / a;
        }
    }
    return std::make_tuple(x, y);
}
std::tuple<int, int> get_another_particular_solution(int x, int y, int a, int
b, int _lambda){
    x = x + _lambda * b;
    y = y - _lambda * a;
    return std::make_tuple(x, y);
}
std::tuple<int, int, int, int> get_smaller_solution(int x0, int y0, int a,
int b){
    int _lambda;
    x0 >= y0 ? _lambda = -1 : _lambda = 1;
    int x1 = 0, y1 = 0;
    while (true){
        std::tie(x1, y1) = get_another_particular_solution(x0, y0, a, b,
_lambda);
        if (abs(x0) + abs(y0) > abs(x1) + abs(y1)){
            x0 = x1, y0 = y1;
            x1 = 0, y1 = 0;
        } else if (abs(x0) + abs(y0) == abs(x1) + abs(y1)){
            break;
        } else {
            x1 = 0, y1 = 0;
            break;
        }
    }
    return std::make_tuple(x0, y0, x1, y1);
}
std::tuple<int, int, int, int> solve_positive_equation(int a, int b, int c){
```

```
    a = abs(a), b = abs(b), c = abs(c);
    int x0, y0, x1, y1;
    std::tie(x0, y0) = get_euler_particular_solution(a, b, c);
    std::tie(x0, y0, x1, y1) = get_smaller_solution(x0, y0, a, b);
    return std::make_tuple(x0, y0, x1, y1);
}
std::tuple<int, int> add_signs_to_solution(int x, int y, int a, int b, int
c){
    int sing_a = a / abs(a), sing_b = b / abs(b), sing_c = c / abs(c);
    x *= sing_a * sing_c;
    y *= sing_b * sing_c;
    return std::make_tuple(x, y);
}
int main(int argc, char **argv){
    int a = 0, b = 0, c = 0;
    auto scanned = std::scanf("%i;%i;%i", &a, &b, &c);
    assert (a != 0 && b != 0 && c != 0);

    int d = get_gcd(a, b);
    if (c % d != 0){
        std::cout << "There are no solutions." << std::endl;
        return 0;
    }
    a = (int) a / d, b = (int) b / d, c = (int) c / d;
    int x0, y0, x1, y1;
    std::tie(x0, y0, x1, y1) = solve_positive_equation(a, b, c);
    std::tie(x0, y0) = add_signs_to_solution(x0, y0, a, b, c);
    std::string solutions[3] = {std::to_string(x0) + ";" +
std::to_string(y0), "", ""};
    if (a == b){
        solutions[1] = std::to_string(y0) + ";" + std::to_string(x0);
    } else if (a == -1 * b){
        solutions[1] = std::to_string(-1 * y0) + ";" + std::to_string(x0);
    }
    if (x1 != 0 && y1 != 0){
```

```
        std::tie(x1, y1) = add_signs_to_solution(x1, y1, a, b, c);
        solutions[2] = std::to_string(x1) + ";" + std::to_string(y1);
    }
    sort(std::begin(solutions), std::end(solutions));
    for(auto i: solutions){
        if (i != ""){
            std::cout << i << std::endl;
        }
    }
    return 0;
}
```

30 The cryptogram issue

20 points

Introduction

A cryptogram is a mathematical puzzle in which various symbols or letters are used to represent numeric digits, typically in the form of a sum or equation. This type of verbal arithmetic is called a cryptogram or cryptarithm.

The classic example, published in the July 1924 issue of Strand Magazine by Henry Dudeney, is:

```
SEND
+
MORE
========
MONEY
```

Each letter can only be replaced by a different digit (from 0 to 9). In this example the value of the letters in the solution is:

```
D = 7
E = 5
M = 1
N = 6
O = 0
R = 8
S = 9
Y = 2
```

(as SEND + MORE = 9567 + 1085 = 10652, which is MONEY).

Note: For this example, there's more than one solution because with D = 1, E = 5, M = 0, N = 3, O = 8, R = 2, S = 7, Y = 6, the cryptogram is also valid (as SEND + MORE = 7531 + 0825 = 08356, which is MONEY).

In the tests used to check your code there will be only a possible solution.

Exercise

We ask you to write a program that receives three inputs (first addend, second addend and result):

```
SEND
MORE
MONEY
```

and returns a line like with the numeric values of the sum and the result all in a row:

```
9567+1085=10652
```

Constraints:

- Addends must be only letters and must be capital letters
- Addends cannot be longer than 9 letters
- Letter 'Ñ' is not allowed
- If there is no solution, the program will return the text "NO SOLUTION"

Input

The input will be three lines:

- The first line contains a capital letter word, no longer than 10 letters, with no 'Ñ', corresponding with the first addend
- The second line contains a capital letter word, no longer than 10 letters, with no 'Ñ', corresponding with the second addend
- The third line contains a capital letter word, no longer than 10 letters, with no 'Ñ', corresponding with the result of the sum

Output

A line representing the sum, using only numbers and the symbols '+' and '='.

NOTE: In some cases there are several solutions for a single cryptarithmetic. This will not be the case in the tests used to check your code.

Example 1

```
DAME
MAS
AMOR
```

Output

```
8931+394=9325
```

Example 2

Input

```
PAPA
MAMA
BEBE
```

Output

```
4141+3131=7272
```

Example 3

Input

```
TRES
DOS
CINCO
```

Output

```
5724+384=06108
```

Example 4

Input

```
LUIS
BEA
ALBA
```

Output

```
1790+342=2132
```

Solutions

Python

```
import numpy as np
import sys
class Node(object):
    def __init__(self, c, v):
        self.c = c
        self.v = v
use = np.full(10, 0, dtype=int)
```

```
def check(nodeArr, count, s1, s2, s3):
    val1 = 0
    val2 = 0
    val3 = 0
    m = 1
    for s in s1[::-1]:
        for j in range(count):
            if nodeArr[j].c == s:
                break
        val1 += m * nodeArr[j].v
        m *= 10
    m = 1
    for s in s2[::-1]:
        for j in range(count):
            if nodeArr[j].c == s:
                break
        val2 += m * nodeArr[j].v
        m *= 10
    m = 1
    for s in s3[::-1]:
        for j in range(count):
            if nodeArr[j].c == s:
                break
        val3 += m * nodeArr[j].v
        m *= 10
    if val3 == (val1+val2):
        return True
    return False
def permutation(count, nodeArr, n, s1, s2, s3):
    if n == count-1:
        for i in range(10):
            if use[i] == 0:
                nodeArr[n].v = i
                if check(nodeArr, count, s1, s2, s3):
                    return True
```

```
        return False
    for i in range(10):
        if use[i] == 0:
            nodeArr[n].v = i
            use[i] = 1
            if permutation(count, nodeArr, n + 1, s1, s2, s3):
                return True
            use[i] = 0
    return False
def solveCryptographic(s1, s2, s3):
    count = 0
    freq = np.full(26, 0, dtype=int)
    for s in s1:
        freq[ord(s)-ord('A')] += 1
    for s in s2:
        freq[ord(s)-ord('A')] += 1
    for s in s3:
        freq[ord(s)-ord('A')] += 1
    for f in freq:
        if f > 0:
            count += 1
    if(count > 10):
        print("ERROR")
    nodeArr = np.empty(count, dtype=Node)
    for i in range(count):
        nodeArr[i] = Node('', 0)
    j = 0
    for i in range(26):
        if freq[i] > 0:
            nodeArr[j].c = chr(i + ord('A'))
            j += 1
    output = ''
    sol = permutation(count, nodeArr, 0, s1, s2, s3)
    if sol:
        for s in s1:
```

```
            for j in range(count):
                if nodeArr[j].c == s:
                    output += str(nodeArr[j].v)
        output += '+'
        for s in s2:
            for j in range(count):
                if nodeArr[j].c == s:
                    output += str(nodeArr[j].v)
        output += '='
        for s in s3:
            for j in range(count):
                if nodeArr[j].c == s:
                    output += str(nodeArr[j].v)
        print(output)
    return sol
def main():
    s1 = input()
    s2 = input()
    s3 = input()
    if solveCryptographic(s1, s2, s3) == False:
        sys.exit("NO SOLUTION")
if __name__ == "__main__":
    main()
```

C++

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
vector<int> use(10);
struct node{
        char c;
        int v;
};
int check(node * nodeArr, const int count, string s1, string s2, string s3)
```

```
{
    int val1 = 0, val2 = 0, val3 = 0, m = 1, j, i;
    // calculate number corresponding to first string
    for (i = s1.length() - 1; i >= 0; i--)
    {
        char ch = s1[i];
        for (j = 0; j < count; j++)
            if (nodeArr[j].c == ch)
                break;
        val1 += m * nodeArr[j].v;
        m *= 10;
    }
    m = 1;
    // calculate number corresponding to second string
    for (i = s2.length() - 1; i >= 0; i--)
    {
        char ch = s2[i];
        for (j = 0; j < count; j++)
            if (nodeArr[j].c == ch)
                break;
        val2 += m * nodeArr[j].v;
        m *= 10;
    }
    m = 1;
    // calculate number corresponding to third string
    for (i = s3.length() - 1; i >= 0; i--)
    {
        char ch = s3[i];
        for (j = 0; j < count; j++)
            if (nodeArr[j].c == ch)
                break;
        val3 += m * nodeArr[j].v;
        m *= 10;
    }
    // sum of first two number equal to third return true
```

```
    if (val3 == (val1 + val2))
        return 1;
    // else return false
    return 0;
}
// Recursive function to check solution for all permutations
bool permutation(const int count, node* nodeArr, int n, string s1, string s2,
string s3)
{
    // Base case
    if (n == count - 1)
    {
        // check for all numbers not used yet
        for (int i = 0; i < 10; i++)
        {
            // if not used
            if (use[i] == 0)
            {
                // assign char at index n integer i
                nodeArr[n].v = i;
                // if solution found
                if (check(nodeArr, count, s1, s2, s3) == 1)
                {
//                    for (int j = 0; j < count; j++)
//                        cout << "" << nodeArr[j].c << "=" <<
nodeArr[j].v<<";";
                    return true;
                }
            }
        }
        return false;
    }
    for (int i = 0; i < 10; i++)
    {
        // if ith integer not used yet
```

```
        if (use[i] == 0)
        {
            // assign char at index n integer i
            nodeArr[n].v = i;
            // mark it as not available for other char
            use[i] = 1;
            // call recursive function
            if (permutation(count, nodeArr, n + 1, s1, s2, s3))
                return true;
            // backtrack for all other possible solutions
            use[i] = 0;
        }
    }
    return false;
}
bool solveCryptographic(string s1, string s2, string s3)
{
    // count to store number of unique char
    int count = 0;
    // Length of all three strings
    int l1 = s1.length();
    int l2 = s2.length();
    int l3 = s3.length();
    // vector to store frequency of each char
    vector<int> freq(26);
    for (int i = 0; i < l1; i++)
        ++freq[s1[i] - 'A'];
    for (int i = 0; i < l2; i++)
        ++freq[s2[i] - 'A'];
    for (int i = 0; i < l3; i++)
        ++freq[s3[i] - 'A'];
    // count number of unique char
    for (int i = 0; i < 26; i++)
        if (freq[i] > 0)
            count++;
```

```
    // solution not possible for count greater than 10
    if (count > 10)
    {
        cout << "ERROR";
        return 0;
    }
    // array of nodes
    node nodeArr[count];
    // store all unique char in nodeArr
    for (int i = 0, j = 0; i < 26; i++)
    {
        if (freq[i] > 0)
        {
            nodeArr[j].c = char(i + 'A');
            j++;
        }
    }
    bool sol= permutation(count, nodeArr, 0, s1, s2, s3);
    if (sol)
    {
    string output;
    for (int i=0; i<s1.length();i++)
    {
            char letter=s1[i];
            for (int j = 0; j < count; j++)
            {
                    if (nodeArr[j].c==letter)
output.append(to_string(nodeArr[j].v));
            }
    }
    output.append("+");
    for (int i=0; i<s2.length();i++)
    {
            char letter=s2[i];
            for (int j = 0; j < count; j++)
```

```
            {
                    if (nodeArr[j].c==letter)
output.append(to_string(nodeArr[j].v));
            }
    }
    output.append("=");
    for (int i=0; i<s3.length();i++)
    {
            char letter=s3[i];
            for (int j = 0; j < count; j++)
            {
                    if (nodeArr[j].c==letter)
output.append(to_string(nodeArr[j].v));
            }
    }
    cout << output;
    }
    return sol;
}
int main()
{
    string s1, s2, s3;
    cin>>s1;
    cin>>s2;
    cin>>s3;
    if (solveCryptographic(s1, s2, s3) == false)
        cout << "NO SOLUTION";
    return 0;
}
```