1. Python Program to count string and Avoid Spaces in string length.

```
user_input = input("Enter a string: ")
# Remove spaces from the input string
string_without_spaces = user_input.replace(" ", "")
# Count the length of the string without spaces
length_without_spaces = len(string_without_spaces)
print(f"The length of the string without spaces is: {length_without_spaces}")
```

---

2. Create a Python code that takes a string as input and counts the number of numbers, vowels and constants in the string

```
# Initialize counters

num_count = 0

vowel_count = 0

consonant_count = 0

vowels=['a','e','i','o','u']

input_string=input('input tour text: ')

# Iterate through each character in the input string

for char in input_string:

# Check if the character is a number

    if char.isdigit():

        num_count += 1
        # Check if the character is a vowel

    elif char.lower() in vowels:

        vowel_count += 1
    # Check if the character is a consonant (alphabetic character excluding vowels)

    elif char.isalpha():

        consonant_count += 1


print(f"Number of numbers: {num_count}")
print(f"Number of vowels: {vowel_count}")
print(f"Number of consonants: {consonant_count}")
```

---

3. Write a program to arrange string characters such that lowercase letters should come first.

```
user_input = input("Enter a string: ")

 lowercase_chars = ""
 uppercase_chars = ""
  other_chars = ""
    for char in user_input:
        if char.islower():
            lowercase_chars += char
        elif char.isupper():
            uppercase_chars += char
        else:
            other_chars += char

    # Concatenate the results
    arranged_string = lowercase_chars + uppercase_chars + other_chars
   print(f"The arranged string is: { arranged_string }")
```

---

4. Write a function reverse_words(sentence) that takes a string representing a sentence as a parameter and returns a new string with the words reversed.

```
>>> "Hello World! Python is amazing."
>>> "olleH .dlroW! nohtyP si .gnizama"

def reverse_words(sentence):
    # Split the sentence into words
    words = sentence.split()
    # Reverse each word in the list
    reversed_words = [word[::-1] for word in words]
    # Join the reversed words back into a sentence
    reversed_sentence = ' '.join(reversed_words)
    return reversed_sentence
# Example usage:
input_sentence = "Hello World! Python is amazing."
result = reverse_words(input_sentence)

print(result)
```

---

5. Write python function to Remove Duplicates from Sorted Array

```
def remove_duplicates(nums):
    # Check for an empty list
    if not nums:
        return 0
    # Initialize two pointers, one for the current position and one for the next position to insert unique
elements
    current = 0
    # Iterate through the array
    for i in range(1, len(nums)):
        # If the current element is not equal to the previous one, update the current position and insert
the unique element
        if nums[i] != nums[current]:
            current += 1
            nums[current] = nums[i]

    # The length of the array with duplicates removed is the current position plus one
    return current + 1
# Example usage:
sorted_array = [1, 1, 2, 2, 2, 3, 4, 4, 5]
new_length = remove_duplicates(sorted_array)

print("Array after removing duplicates:", sorted_array[:new_length])
```

---

6. write python function to Prime checking

```
def is_prime(number):
    # Prime numbers are greater than 1
    if number > 1:
        # Check for factors from 2 to the square root of the number
        for i in range(2, int(number**0.5) + 1):
            if (number % i) == 0:
                # If there is a factor, the number is not prime
                return False
        # If no factors are found, the number is prime
        return True
    else:
        # Numbers less than or equal to 1 are not prime
        return False

# Example usage:
user_input = int(input("Enter a number to check for primality: "))

if is_prime(user_input):
    print(f"{user_input} is a prime number.")
else:
    print(f"{user_input} is not a prime number.")
```

---

7. write python function to calculate GCD

```
def calculate_gcd(a, b):
    while b:
        a, b = b, a % b
    return abs(a)

# Example usage:
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))

gcd_result = calculate_gcd(num1, num2)

print(f"The GCD of {num1} and {num2} is: {gcd_result}")
```

---

8. write python function to acts as a basic calculator.

```
def calculator(num1, num2, operator):
    if operator == '+':
        result = num1 + num2
    elif operator == '-':
        result = num1 - num2
    elif operator == '*':
        result = num1 * num2
    elif operator == '/':
        # Check if the divisor is not zero
        if num2 != 0:
            result = num1 / num2
        else:
            return "Error: Division by zero is not allowed."
    else:
        return "Error: Invalid operator."

    return result

# Example usage:
number1 = float(input("Enter the first number: "))
number2 = float(input("Enter the second number: "))
operation = input("Enter the operator (+, -, *, /): ")

result = calculator(number1, number2, operation)

print(f"The result of {number1} {operation} {number2} is: {result}")
```

---

9. Write a Python program to make each word in the file start with capital letter.

```
def capitalize_words_in_file(file_path):
    # Read content from the file
    with open(file_path, 'r') as file:
        content = file.read()

    # Capitalize the first letter of each word
    modified_content = ' '.join(word.capitalize() for word in content.split())

    # Write the modified content back to the same file
    with open(file_path, 'w') as file:
        file.write(modified_content)

    print("File successfully updated.")
# Example usage:
file_path = "example.txt"  # Replace with the actual path to your file
capitalize_words_in_file(file_path)
```

---

10. Write a Python program to remove any (“# / * $ % @“) from the text file and print text.

```
def remove_special_characters(file_path):
    # Read content from the file
    with open(file_path, 'r') as file:
        content = file.read()
    # Remove specified characters
    special_characters = ["#", "/", "*", "$", "%", "@"]
    for char in special_characters:
        content = content.replace(char, "")
    # Print the modified text
    print("Modified Text:")
    print(content)
# Example usage:
file_path = "example.txt"  # Replace with the actual path to your file
remove_special_characters(file_path)
```

11.Write a program in Python to count uppercase character in a text file

```
def count_uppercase_characters(file_path):
    with open(file_path, 'r') as file:
        content = file.read()
    uppercase_count = sum(1 for char in content if char.isupper())


    print(f"Number of uppercase characters in the file: {uppercase_count}")
# Example usage:
file_path = "example.txt"
count_uppercase_characters(file_path)
```

12.Write a Python program to write a list to a file

```
def write_list_to_file(file_path, my_list):
    # Open the file in write mode
    with open(file_path, 'w') as file:
        # Write each element of the list to the file
        for item in my_list:
            file.write(str(item) + '\n')
    print(f"The list has been successfully written to the file.")
# Example usage:
my_list = ["apple", "banana", "orange", "grape"]
file_path = "output.txt"  # Specify the desired file path
write_list_to_file(file_path, my_list)
```

13.Write a Python program to extract characters from various text files and puts them into a list.

```
def extract_characters_from_files(file_paths):
  all_characters = []
```

```
    # Iterate through each file path
    for file_path in file_paths:
        # Read content from the file
        with open(file_path, 'r') as file:
            content = file.read()
        # Extract characters and append to the list
        characters = [char for char in content]
        all_characters.extend(characters)

    return all_characters
# Example usage:
file_paths = ["file1.txt", "file2.txt", "file3.txt"]  # Replace with the actual file paths
result_characters = extract_characters_from_files(file_paths)
print("Characters extracted from files:")
print(result_characters)
```

14.Write a Python program to read a file and print unique words

```
def get_unique_words(file_path):
    unique_words = set()
    # Read content from the file
    with open(file_path, 'r') as file:
        content = file.read()
    # Split the content into words
    words = content.split()
    # Add unique words to the set
    unique_words.update(words)
    return unique_words
# Example usage:
file_path = "sample.txt"  # Replace with the actual file path
```

```
result_unique_words = get_unique_words(file_path)
print("Unique words in the file:")
print(result_unique_words)
```

15.Create a dictionary with student names as keys and their corresponding grades as values. Write a program to calculate the average grade.

```
def calculate_average_grade(student_grades):
    if not student_grades:
        return "No grades provided."
    total_grades = sum(student_grades.values())
    average_grade = total_grades / len(student_grades)
    return average_grade
# Example usage:
students = {
    "Alice": 85,
    "Bob": 92,
    "Charlie": 78,
    "David": 90,
    "Eva": 88
}
average = calculate_average_grade(students)
print("Student Grades:")
for name, grade in students.items():
    print(f"{name}: {grade}")

print(f"\nAverage Grade: {average:.2f}")
```

16.Write a program that takes a tuple of numbers and returns a new tuple with the elements multiplied by a given factor.

```
def multiply_tuple_elements(input_tuple, factor):
    multiplied_tuple = tuple(element * factor for element in input_tuple)
    return multiplied_tuple
# Example usage:
input_numbers = (2, 5, 8, 10)
multiplication_factor = 3
result_tuple = multiply_tuple_elements(input_numbers, multiplication_factor)
print(f"Original Tuple: {input_numbers}")
print(f"Factor: {multiplication_factor}")
print(f"Resulting Tuple: {result_tuple}")
```

17.Implement a function to check if a key exists in a dictionary.

```
def key_exists(dictionary, key_to_check):
    """
    Returns:
    - True if the key exists, False otherwise.
    """
    return key_to_check in dictionary

# Example usage:
my_dict = {'name': 'John', 'age': 25, 'city': 'New York'}

key_to_search = 'age'
if key_exists(my_dict, key_to_search):
    print(f"The key '{key_to_search}' exists in the dictionary.")
else:
    print(f"The key '{key_to_search}' does not exist in the dictionary.")
```

18.Write a python program to find the longest words

```
def find_longest_words(text):
    # Split the text into words
    words = text.split()
    # Find the length of the longest word(s)
    max_length = max(len(word) for word in words)
    # Find the longest word(s)
    longest_words = [word for word in words if len(word) == max_length]
    return longest_words

# Example usage:
input_text = "Python is a powerful programming language with a simple syntax."
result = find_longest_words(input_text)

print("Longest word(s):")
print(result)
```

19.Write a program to concatenate two tuples.

```
def concatenate_tuples(tuple1, tuple2):
    concatenated_tuple = tuple1 + tuple2
    return concatenated_tuple
# Example usage:
tuple_a = (1, 2, 3)
tuple_b = ('a', 'b', 'c')
result = concatenate_tuples(tuple_a, tuple_b)

print("Concatenated Tuple:")
```

```
print(result)
```

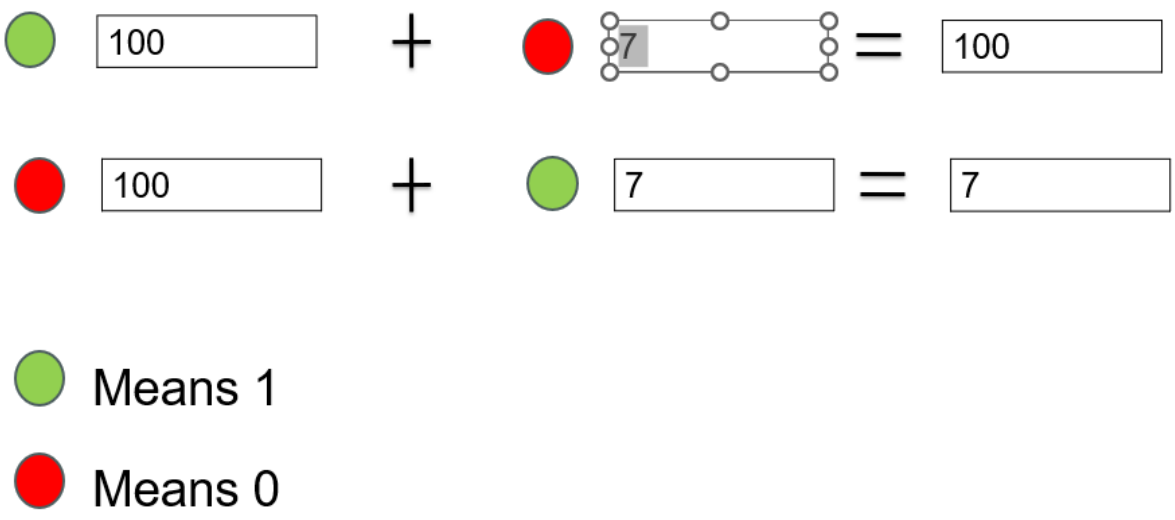
20.Write a Python function to find the intersection of two sets.

```
def find_intersection(set1, set2):
    intersection_set = set1.intersection(set2)
    return intersection_set
# Example usage:
set_a = {1, 2, 3, 4, 5}
set_b = {3, 4, 5, 6, 7}
result_intersection = find_intersection(set_a, set_b)

print("Intersection of Sets:")
print(result_intersection)
```

---

21. Write a program that reads an integer and print 100 if number is even or 7 if number is odd
    - E.g. for input 8 ⇒ 100
    - E.g. for input 133 ⇒ 7
    - Note: if you know if condition, don't use it.



```
3
4
5   number = int(input("Enter Integer Number"))
6   #We use input function to take input from user
7   #input() function converts input into a string
8   #So we need to cast this input into an integer
9   is_even = (number%2 == 0)
10  is_odd = 1 - is_even
11  result = (is_even * 100) + (is_odd * 7)
12  print(result)
13
14
```

---

22. We know N % M computes the remainder of division

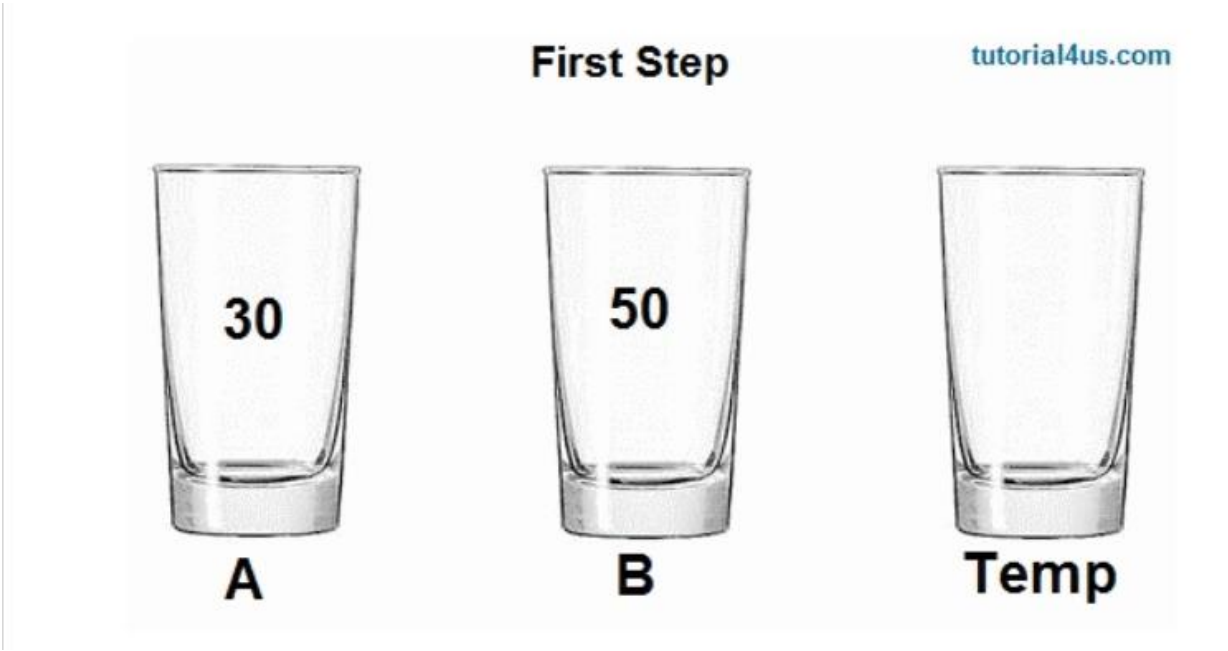
Write a program that reads 2 integers and print such reminder without using the modulus operator %

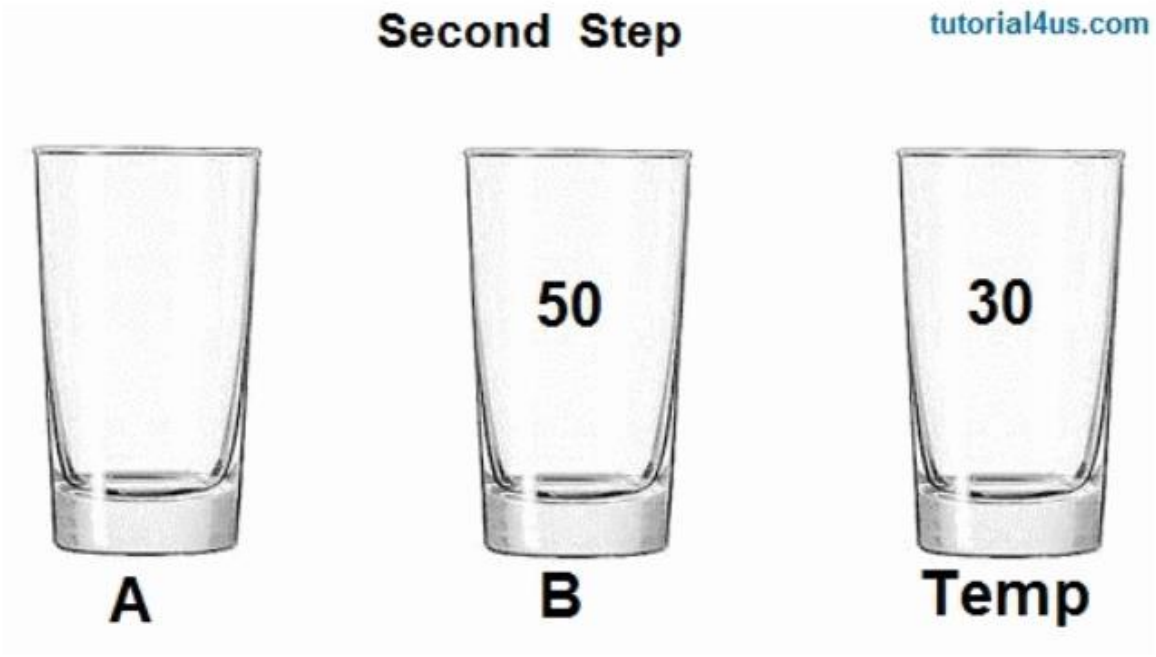
- E.g. for inputs 27 and 12 ⇒ output 3
  - Remember in math: 27 % 12 = 3

```
2
3   n = int(input("Enter n"))
4   m = int(input("Enter m"))
5   int_division = int(n/m)
6   print(n - int_division*m )
7
8
```

23. Write a program that reads 2 variables num1 and num2
    - E.g. say we read num1 = 30 and num2 = 50

Target: we want swap the values of Num1 and Num2?
- Swap means exchange
- So Num1 takes value 50 and Num2 takes value 30





Third Step

tutorial4us.com

Fourth Step

tutorial4us.com

```
2
3   A = 30
4   B = 50
5   Temp = None
6   #You can assign zero or any value, we will overwrite anyway
7   Temp = A
8   A = B
9   B = Temp
10  print(A)
11  print(B)
12
```

24.

A. Elephant

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

An elephant decided to visit his friend. It turned out that the elephant's house is located at point 0 and his friend's house is located at point $x(x > 0)$ of the coordinate line. In one step the elephant can move 1, 2, 3, 4 or 5 positions forward. Determine, what is the minimum number of steps he need to make in order to get to his friend's house.

Input

The first line of the input contains an integer x ($1 \le x \le 1\,000\,000$) — The coordinate of the friend's house.

Output

Print the minimum number of steps that elephant needs to make to get from point 0 to point x.

Examples

input	Copy
5	

output	Copy
1	

input	Copy
12	

output	Copy
3	

Note

In the first sample the elephant needs to make one step of length 5 to reach the point x.

In the second sample the elephant can get to point x if he moves by 3, 5 and 4. There are other ways to get the optimal answer but the elephant cannot reach x in less than three moves.

```
2
3   x = int(input())
4   is_multipleof5 = (x%5==0)
5   isnot = 1 - is_multipleof5
6   # If it's multiple of 5 so the answer is the integral division of x over 5
7   # otherwise it's the integral divisoin plus one of the rest(4,3,2,or1)
8   res = is_multipleof5*(int(x/5)) +  isnot*((int(x/5))+1)
9   print(res)
10
11
12
```

25. Write a program that reads 2 numbers and print their + - * / as following
    - For inputs 12 and 3

```
Enter First Number 12
Enter First Number 3
12 + 3   =   15
12 - 3   =   9
12 * 3   =   36
12 / 3   =   4.0
```

```
3
4    number1 = int(input("Enter First Number "))
5    number2 = int(input("Enter First Number "))
6    print(number1 , "+" , number2 , " = " , (number1+number2))
7    print(number1 , "-" , number2 , " = " , (number1-number2))
8    print(number1 , "*" , number2 , " = " , (number1*number2))
9    print(number1 , "/" , number2 , " = " , (number1/number2))
10
```

26.

A. Word Capitalization

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Capitalization is writing a word with its first letter as a capital letter. Your task is to capitalize the given word.

Note, that during capitalization all the letters except the first one remains unchanged.

Input

A single line contains a non-empty word. This word consists of lowercase and uppercase English letters. The length of the word will not exceed 10^3.

Output

Output the given word after capitalization.

Examples

input	Copy
ApPLe	
output	Copy
ApPLe	

input	Copy
konjac	
output	Copy
Konjac	

Solution:

```
str=input()
ch=str[0].upper()
print(ch+str[1:len(str)])
```

27.

A. Vasya the Hipster

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

One day Vasya the Hipster decided to count how many socks he had. It turned out that he had a red socks and b blue socks.

According to the latest fashion, hipsters should wear the socks of different colors: a red one on the left foot, a blue one on the right foot.

Every day Vasya puts on new socks in the morning and throws them away before going to bed as he doesn't want to wash them.

Vasya wonders, what is the maximum number of days when he can dress fashionable and wear different socks, and after that, for how many days he can then wear the same socks until he either runs out of socks or cannot make a single pair from the socks he's got.

Can you help him?

Input

The single line of the input contains two positive integers a and b ($1 \le a, b \le 100$) — the number of red and blue socks that Vasya's got.

Output

Print two space-separated integers — the maximum number of days when Vasya can wear different socks and the number of days when he can wear the same socks until he either runs out of socks or cannot make a single pair from the socks he's got.

Keep in mind that at the end of the day Vasya throws away the socks that he's been wearing on that day.

Examples

input	Copy
3 1	
output	Copy
1 1	

input	Copy
2 3	
output	Copy
2 0	

input	Copy
7 3	
output	Copy
3 2	

Note

In the first sample Vasya can first put on one pair of different socks, after that he has two red socks left to wear on the second day.

Solution:

```
a=int(input("Enter a: "))
b=int(input("Enter b: "))
if a>=b:
    a -= b
    a /= 2
    print("%2d %2d" %(b, a))
else:
    b -= a
```

```
b /= 2
print("%2d %2d" %(a, b))
```

28.

Y. The last 2 digits

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Given 4 numbers A, B, C and D. Print the **last 2 digits** from their **Multiplication**.

Input
Only one line containing four numbers A, B, C and D ($2 \leq A, B, C, D \leq 10^9$).

Output
Print the **last 2 digits** from their **Multiplication**.

Examples

input	Copy
5 7 2 4	

output	Copy
80	

input	Copy
3 9 9 9	

output	Copy
87	

Note
First Example :

the Multiplication of 4 numbers is **5 * 7 * 2 * 4 = 280** so the answer will be the last 2 digits which are **80**.

Second Example :

the Multiplication of 4 numbers is **3 * 9 * 9 * 9 = 2187** so the answer will be the last 2 digits which are **87**.

Solution:

```
n1=int(input("n1 = "))
n2=int(input("n2 = "))
n3=int(input("n3 = "))
n4=int(input("n4 = "))
n1%=100
n2%=100
n3%=100
n4%=100
t=n1*n2*n3*n4
if t%100<=9:
    print(0)
else:
    print(t%100)
```

29.

X. Two intervals

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Given the boundaries of **2** intervals. Print the boundaries of their **intersection**.

Note: **Boundaries** mean the two ends of an interval which are the starting number and the ending number.

Input

Only one line contains two intervals $[l_1, r_1]$, $[l_2, r_2]$ where $(1 \leq l_1, l_2, r_1, r_2 \leq 10^9)$, $(l_1 \leq r_1, l_2 \leq r_2)$.

It's guaranteed that $l_1 \leq r_1$ and $l_2 \leq r_2$.

Output

If there is an **intersection** between these **2** intervals print its boundaries , otherwise print **-1**.

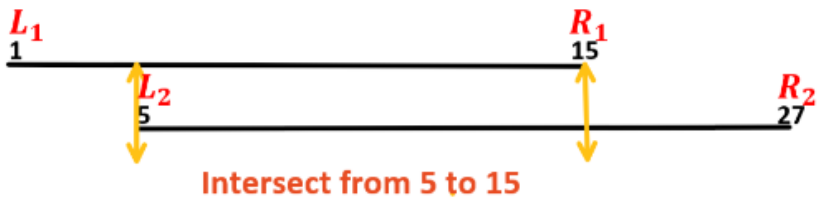
Examples

input	Copy
1 15 5 27	
output	Copy
5 15	

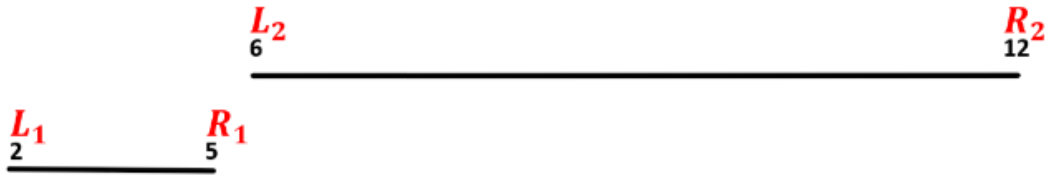
input	Copy
2 5 6 12	
output	Copy
-1	

Note

First Example :



Second Example :



There are No intersections

Solution:

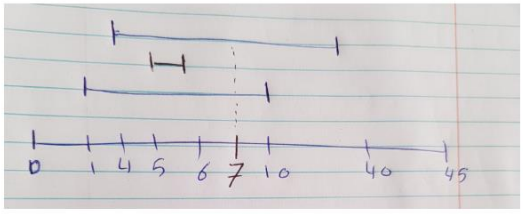
```
s1=int(input("s1 = "))
e1=int(input("e1 = "))
s2=int(input("s1 = "))
e2=int(input("e2 = "))
if ((s2>e1 and e2>s1) or (s2<s1 and e2<s1)):
```

```
    print(-1)
else:
    s = max(s1,s2)
    e = min(e1,e2)
    print("%2d %2d" %(s,e))
```

30.

Homework 7: Intervals

- Read number X then read 6 numbers s1, e1, s2, e2, s3, e3
  - These 6 numbers are for 3 interval
  - Each Interval is a range [start, end]
  - Number X in a range if start <= X <= end
  - E.g 7 in range [5, 12] but not in range [10, 20]
- Print how many digits X is part of it
- Inputs
  - 7 1 10 5 6 4 40 ⇒ 2
    - Number 7 exists in 2 intervals [1, 10] and [4, 40]
  - 10 5 15 6 100 3 30 ⇒ 3
    - 1o exists in the 3 intervals [5 15], [6 100], [3 30]
  - 10 100 200 100 101 120 170 ⇒ 0



Solution:

```
x=int(input("x= "))
s=int(input("s= "))
e=int(input("e= "))
cnt=0
cnt += (s <= x and x <= e)
s=int(input("s= "))
e=int(input("e= "))
cnt += (s <= x and x <= e)
s=int(input("s= "))
e=int(input("e= "))
cnt += (s <= x and x <= e)
print(cnt)
```

31.

Given an array of integers, find the sum of its elements.

For example, if the array $ar = [1, 2, 3]$, $1 + 2 + 3 = 6$, so return 6.

Function Description

Complete the simpleArraySum function in the editor below. It must return the sum of the array elements as an integer.

simpleArraySum has the following parameter(s):

- ar: an array of integers

Input Format

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

The second line contains n space-separated integers representing the array's elements.

Constraints

$0 < n, ar[i] \leq 1000$

Output Format

Print the sum of the array's elements as a single integer.

Sample Input

```
6
1 2 3 4 10 11
```

Sample Output

```
31
```

Explanation

We print the sum of the array's elements: $1 + 2 + 3 + 4 + 10 + 11 = 31$.

Solution:

```python
def simpleArraySum(ar):

    total = 0
    for number in ar:
        total += number
    return total


# Example usage
example_array = [1, 2, 3, 4, 10, 11]
result = simpleArraySum(example_array)
print(result)  # Output should be 31
```

32.

A. Watermelon

time limit per test: 1 second
memory limit per test: 64 megabytes
input: standard input
output: standard output

One hot summer day Pete and his friend Billy decided to buy a watermelon. They chose the biggest and the ripest one, in their opinion. After that the watermelon was weighed, and the scales showed w kilos. They rushed home, dying of thirst, and decided to divide the berry, however they faced a hard problem.

Pete and Billy are great fans of even numbers, that's why they want to divide the watermelon in such a way that each of the two parts weighs even number of kilos, at the same time it is not obligatory that the parts are equal. The boys are extremely tired and want to start their meal as soon as possible, that's why you should help them and find out, if they can divide the watermelon in the way they want. For sure, each of them should get a part of positive weight.

Input

The first (and the only) input line contains integer number w ($1 \le w \le 100$) — the weight of the watermelon bought by the boys.

Output

Print `YES`, if the boys can divide the watermelon into two parts, each of them weighing even number of kilos; and `NO` in the opposite case.

Examples

input	Copy
8	

output	Copy
YES	

Note

For example, the boys can divide the watermelon into two parts of 2 and 6 kilos respectively (another variant — two parts of 4 and 4 kilos).

```python
def divide_watermelon(w):

    if w > 2 and w % 2 == 0:
        return "YES"
    else:
        return "NO"

# Testing the function with an example
example_weight = 6
result = divide_watermelon(example_weight)
print(result)  # Output: 'YES'
```

33.

A. Way Too Long Words

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Sometimes some words like "localization" or "internationalization" are so long that writing them many times in one text is quite tiresome.

Let's consider a word *too long*, if its length is **strictly more** than 10 characters. All too long words should be replaced with a special abbreviation.

This abbreviation is made like this: we write down the first and the last letter of a word and between them we write the number of letters between the first and the last letters. That number is in decimal system and doesn't contain any leading zeroes.

Thus, "localization" will be spelt as "l10n", and "internationalization» will be spelt as "i18n".

You are suggested to automatize the process of changing the words with abbreviations. At that all too long words should be replaced by the abbreviation and the words that are not too long should not undergo any changes.

Input

The first line contains an integer n ($1 \le n \le 100$). Each of the following n lines contains one word. All the words consist of lowercase Latin letters and possess the lengths of from 1 to 100 characters.

Output

Print n lines. The i-th line should contain the result of replacing of the i-th word from the input data.

Examples

input	Copy
4 word localization internationalization pneumonoultramicroscopicsilicovolcanoconiosis	

output	Copy
word l10n i18n p43s	

Solution:

```
def abbreviate_word(word):

    if len(word) > 10:
        return f"{word[0]}{len(word) - 2}{word[-1]}"
    else:
        return word

# Example usage
example_words = ["word", "localization", "internationalization",
"pneumonoultramicroscopicsilicovolcanoconiosis"]
abbreviated_words = [abbreviate_word(word) for word in example_words]
print(abbreviated_words)
```

34.

A. Bit++

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

The classic programming language of Bitland is Bit++. This language is so peculiar and complicated.

The language is that peculiar as it has exactly one variable, called x. Also, there are two operations:

- Operation ++ increases the value of variable x by 1.
- Operation -- decreases the value of variable x by 1.

A statement in language Bit++ is a sequence, consisting of exactly one operation and one variable x. The statement is written without spaces, that is, it can only contain characters "+", "-", "X". Executing a statement means applying the operation it contains.

A programme in Bit++ is a sequence of statements, each of them needs to be executed. Executing a programme means executing all the statements it contains.

You're given a programme in language Bit++. The initial value of x is 0. Execute the programme and find its final value (the value of the variable when this programme is executed).

Input

The first line contains a single integer n ($1 \le n \le 150$) — the number of statements in the programme.

Next n lines contain a statement each. Each statement contains exactly one operation (++ or --) and exactly one variable x (denoted as letter «X»). Thus, there are no empty statements. The operation and the variable can be written in any order.

Output

Print a single integer — the final value of x.

Examples

input	Copy
1 ++X	

output	Copy
1	

input	Copy
2 X++ --X	

output	Copy
0	

Solution:

```
n=int(input())
x=0

for i in range(0,n):
    s = input("st: ")
    if "++" in s:
        x += 1
    elif "--" in s:
```

```
        x -= 1
print(x)
```

35.

A. Football

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Petya loves football very much. One day, as he was watching a football match, he was writing the players' current positions on a piece of paper. To simplify the situation he depicted it as a string consisting of zeroes and ones. A zero corresponds to players of one team; a one corresponds to players of another team. If there are at least 7 players of some team standing one after another, then the situation is considered dangerous. For example, the situation 00100110111111101 is dangerous and 11110111011101 is not. You are given the current situation. Determine whether it is dangerous or not.

Input

The first input line contains a non-empty string consisting of characters "0" and "1", which represents players. The length of the string does not exceed 100 characters. There's at least one player from each team present on the field.

Output

Print "YES" if the situation is dangerous. Otherwise, print "NO".

Examples

input	Copy
001001	
output	Copy
NO	

input	Copy
1000000001	
output	Copy
YES	

Solution:

```
string=raw_input()
if "0000000" in string or "1111111" in string:
        print "YES"
else:
        print "NO"
```

36.

Homework 5: Remove evens inplace

- Read a line of N integers.
- Implement function: **def** remove_evens_inplace(lst):
  - It finds all the even numbers and remove them **in place**
  - Try to do it without creating new memory
- Input ⇒ Output
- 1 2 3 4 5 6 ⇒ 1 3 5
- -6 6 ⇒ Empty output
- Empty input ⇒ Empty output

Solution:

```
def remove_evens_inplace1(lst):
    # iterate backward
    for pos in range(len(lst)-1, -1, -1):
        if lst[pos] % 2 == 0:
            del lst[pos]
    return lst


def remove_evens_inplace2(lst):
    # iterate on revered but get the right index
    sz = len(lst)   # important. take it here as list will be updated
    for pos, item in enumerate(reversed(lst)):
        if item % 2 == 0:
            pos = sz - pos - 1    # idx in original list
            # recall pos will be reassigned in every iteration
            del lst[pos]
    return lst



if _name_ == '_main_':
    lst = list(map(int, input().split()))

    remove_evens_inplace2(lst)

    print(lst)
```

37.

Homework 2: Digits frequency

- Read a line of N integers.
- Compute the digits [0 to 8] frequency of all the N numbers
  - Input 78 307 [compute digits frequency of 7 8 3 0 7]
  - Output:
  - 0 1
  - 1 0 [digit 1 never appeared]
  - 2 0
  - 3 1
  - 4 0
  - 5 0
  - 6 0
  - 7 2 [digit 7 appeared twice]
  - 8 1
  - 9 0

Solution:

```
def digits_frequency(lst):
    freq = [0] * 10     # to compute fromquency from 0 to 9

    for value in lst:
        value = abs(value)  # get rid of the sign

        if value == 0:
            freq[0] += 1    # special case
        else:
            while value > 0:
                digit = value % 10
                value //= 10
                freq[digit] += 1

    return freq


if _name_ == '_main_':
    lst = list(map(int, input().split()))
    freq = digits_frequency(lst)

    for idx in range(10):
        print(idx, freq[idx])
```

38.

Homework 1: Find most frequent number

- Read a line of N integers. Each integer is -500 <= value <= 270
- Find the value that repeated the most number of times.
  - If there are many solutions: find the **smallest** value
- Input ⇒ output
  - -1 2 -1 3 -1 5 5 ⇒ Value -1 repeated 3

Solution:

```
def most_frequent_fast(lst):
    # With simple change we can use the practice code
    # we will shift all the data to start from ZERO (so we can index normally)
    # then later undo the effect
    # to do that: just subtract the minimum
    # e.g. if input is -10 20 -2 9 20
    # the min is -10
    # subtract it from all: 0 30_oop 8 19 30_oop
    # Find max 30_oop. Undo with -10 ==> 20
    mn, mx = min(lst), max(lst)
    freq_lst  = [0] * (mx - mn +1)

    for value in lst:
        print(value - mn)
        freq_lst[value - mn] += 1

    # argmax - Observe: the tie is also handled!
    most_value = freq_lst.index(max(freq_lst))

    return most_value + mn, freq_lst[most_value]


if _name_ == '_main_':
    lst = list(map(int, input().split()))
    most_value, frequency = most_frequent_fast(lst)
    print('Value', most_value, 'repeated', frequency)
```

A. Theatre Square

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Theatre Square in the capital city of Berland has a rectangular shape with the size $n \times m$ meters. On the occasion of the city's anniversary, a decision was taken to pave the Square with square granite flagstones. Each flagstone is of the size $a \times a$.

What is the least number of flagstones needed to pave the Square? It's allowed to cover the surface larger than the Theatre Square, but the Square has to be covered. It's not allowed to break the flagstones. The sides of flagstones should be parallel to the sides of the Square.

Input
The input contains three positive integer numbers in the first line: n, m and a ($1 \le n, m, a \le 10^9$).

Output
Write the needed number of flagstones.

Examples

input	Copy
6 6 4	
output	**Copy**
4	

Solution:

```
from math import ceil
n,m,a = map(int,input().split())
A = ceil(n/a)
B = ceil(m/a)
print(int(A*B))
```

40.

Homework 4: Recamán's sequence

- The first terms of this sequence are 0, 1, 3, 6, 2, **7**, …
  - So last term **value** is 7 and its **index** is 5 (zero based)
  - The next value is either:
    - **LastValue - LastIndex - 1** if the following 2 conditions are satisfied:
      - value > 0 and It did not appear before
      - E.g. 7 (last value) - last index (5) - 1 = 7-5-1 = 1 (> 0 but already exists)
    - Or **LastValue + LastIndex + 1** = 7+5+1 = 13
- Read integer zero-based index ([1, 200]) and print the value of this index
  - E.g. (6 ⇒ 13), (9 ⇒ 21), (17 ⇒ 25)
- Don't use nested loops
- The series is: 0, 1, 3, 6, 2, 7, **13**, 20, 12, **21**, 11, 22, 10, 23, 9, 24, 8, **25**, 43

Solution:

```
def recaman(n):
    if n == 0:
        return 0

    # For N, probably an upper bound value is n * 10
    occurrence = [0] * n * 10   # empty for n = 0
    last_value, occurrence[0] = 0, 1  # first term

    for i in range(1, n+1):
        last_idx = i - 1

        val = last_value - last_idx - 1

        if val < 0 or occurrence[val]:
            val = last_value + last_idx + 1

        occurrence[val], last_value = 1, val

    return last_value

if __name__ == '__main__':
    n = int(input())

    print(recaman(n))
```