Exercise: Text Processing

This document defines the exercises for the "Python Fundamentals" course at @SoftUni Global

1.Valid Usernames

Write a program that **reads usernames** on a **single** line (separated by **", "**) and **prints** all **valid usernames** on separate lines.

A valid username:

- has **length** between 3 and 16 characters inclusive
- **can contain** only letters, numbers, hyphens, and underscores
- has **no redundant symbols** before, after, or in between

Examples

Input	Output
sh, too_long_username, !lleg@l ch@rs, jeffbutt	jeffbutt
Jeff, john45, ab, cd, peter-ivanov, @smith	Jeff John45 peter-ivanov

2. Character Multiplier

Create a program that receives **two strings** on a **single line** separated by a single **space.** Then, it prints the **sum** of their **multiplied character codes** as follows: multiply **`str1[0]`** with **`str2[0]`** and add the result to the total sum, then continue with the next two characters. If one of the strings is **longer** than the other, **add** the **remaining** character codes to the **total sum** without multiplication.

Examples

Input	Output
George Peter	52114
123 522	7647
a aaaa	9700

3. Extract File

Write a program that reads the path to a file and **subtracts** the **file name** and its **extension**.

Examples

Input	Output
C:\Internal\training-internal\Template.pptx	File name: Template File extension: pptx

C:\Projects\Data-Structures\LinkedList.cs	File name: LinkedList File extension: cs

4. Caesar Cipher

Write a program that returns an **encrypted version** of the same text. Encrypt the text by **replacing each character** with the corresponding character **three** positions **forward in the ASCII table**. For example, **A** would be replaced with **D**, **B** would become **E**, and so on. Print the **encrypted text**.

Examples

Input	Output
Programming is cool!	Surjudpplqj#lv#frro$
One year has 365 days.	Rqh#|hdu#kdv#698#gd|v1

5. Emoticon Finder

Find all emoticons in the text. An emoticon **always starts with ":"** and is followed by a **symbol.**
The input will be provided as a **single string.**

Example

Input	Output
There are so many emoticons nowadays :P. I have many ideas :O what input to place here :)	:P :O :)

6. Replace Repeating Chars

Write a program that reads a string from the console and **replaces** any **sequence of the same letters** with a **single corresponding letter**.

Examples

Input	Output
aaaaabbbbbcdddeeeedssaa	abcdedsa
qqqwerqwecccwd	qwerqwecwd

7. String Explosion

Write a program that reads a string from the console that contains:

- Explosions marked with **'>'**
- Immediately after the mark, there will be an **integer x**, which signifies the **strength** of the explosion
- Any other characters

Your task is to **delete x characters**, starting **after** the exploded **character** (**'>'**). If you find another explosion mark (**'>'**) while deleting characters, you should **add** the **strength** to your **previous explosion**. You should **not delete** the **explosion** character – **'>'**.

When all characters are processed, **print** the final string.

Constraints

- You will **always** receive **strength** for the explosions
- The path will consist only of letters from the **Latin alphabet**, **integers,** and the char `'>'`
- The strength of the punches will be in the interval [0…9]

Examples

Input	Output	Comments
abv>1>1>2>2asdasd	abv>>>>dasd	1st explosion is at index **3,** with a **strength** of **1**. We delete **only** the **digit after** the explosion character. The string will look like this: **abv>>1>2>2asdasd** 2nd explosion is with strength **one,** and the string transforms to this: **abv>>>2>2asdasd** 3rd explosion is now with a strength of 2. We delete the digit, and we find **another** explosion. At this point, the string looks like this: **abv>>>>2asdasd**. 4th explosion is with strength **2**. We have **1** strength **left** from the previous explosion, and we **add** the strength of the **current** explosion to what is **left**, which adds up to a **total** strength of **3**. We **delete** the next **three characters,** and we **receive** the **string abv>>>>dasd** We do **not** have **any more explosions,** and we print the result: **abv>>>>dasd**
pesho>2sis>1a>2akarate>4hexmaster	pesho>is>a>karate>master	

8. *Letters Change Numbers

John invented a game with numbers and letters from the English alphabet. The game was simple.

You receive a string consisting of a **number between two letters**. For the given string, you should perform different mathematical operations to achieve a result:

- **First,** if the letter **in front of** the number is:
  - **Uppercase: divide** the number by the letter's **position** in the alphabet (starting from 1)
  - **Lowercase: multiply** the number with the letter's **position** in the alphabet (starting from 1)
- **Next, if** the **letter after** the number is:
  - **Uppercase: subtract** its position from the resulting number (starting from 1)
  - **Lowercase: add** its position to the resulting number (starting from 1)

The game was too easy for John. He decided to complicate it by doing the same calculations to **multiple** strings keeping track of only the **total sum** of all results. Once he started to solve this with more strings and bigger numbers, it became quite hard to do it only in his mind.

He kindly asks you to write a program that **performs the operations** described above and **sums the final results of each string**.

Input

- The input comes from the console as a **single line, holding a sequence of strings**
- Strings are separated by **one or more white spaces**
- The input data will always be valid. There is no need to check it explicitly

Output

- Print at the console a single number:
  - The **total sum of all processed numbers**, formatted to the second decimal separator

Constraints

- The **count** of the strings will be in the range `[1 ... 10]`
- The numbers between the letters will be integers in the range `[1 ... 2 147 483 647]`
- Time limit: 0.3 sec. Memory limit: 16 MB

Examples

Input	Output	Comments
`A12b s17G`	`330.00`	`12/1=12, 12+2=14, 17*19=323, 323-7=316, 14+316=330`
`P34562Z q2576f H456z`	`46015.12`	
`a1A`	`0.00`	

9. *Rage Quit

Every gamer knows what rage-quitting means. It's when you're just not good enough, and you blame everybody else for losing a game - you press the CAPS LOCK key on the keyboard and flood the chat with gibberish to show your frustration.

Peter is a gamer, a bad one. When he rage-quits, he wants to be the most annoying kid on his team; he wants something truly spectacular. He asks for your help.

He'll give you **a series of strings (containing only non-numerical characters) followed by non-negative numbers (N),** e.g., **`"a3"`**. You need to **convert the letters to uppercase** for each string and **print it repeatedly N times** on the console. In the example, you need to write back **`"AAA"`**.

First, on the output, you should print a statistic of the **number of unique symbols** used (case-insensitive) in the format: **`"Unique symbols used {0}"`**.

Next, **print the rage message** itself.

The **strings and numbers will not be separated by anything**. The input will always start with a **non-numeric symbol**, and for each string, there will be a corresponding number. The input will be given on a **single line**.

Input

- The input data should be read from the console.
- It consists of a single line holding a series of **string-number sequences**.

- The input data will always be valid. There is no need to check it explicitly.

Output

- The output should be printed on the console. It should consist of **exactly two lines**:
  - On the first line, print the **number of unique symbols used** in the message in the format described above.
  - On the second line, print the **rage message**.

Constraints

- The count of **string-number pairs** will be in the range [1 … 20 000].
- Each string will contain any character **except digits**. The **length** of each string will be in the range [1 … 20].
- The **repeat count** for each string will be an integer in the range [0 … 20].
- Allowed working time for your program: 0.3 seconds. Allowed memory: 64 MB.

Examples

Input	Output	Comments
a3	Unique symbols used: 1 AAA	We have just one string-number pair. The symbol is 'a', convert it to uppercase and repeat 3 times: AAA. Only one symbol is used ('A').
aSd2&5s@1	Unique symbols used: 5 ASDASD&&&&&S@	"aSd" is converted to "ASD" and repeated twice; "&" is repeated 5 times; "s@" is converted to "S@" and repeated once. 5 symbols are used: 'A', 'S', 'D', '&' and '@'.

10. *Winning Ticket

The lottery is exciting. However, checking a million tickets for winnings only by hand is not. So, you are given the task of creating a program that automatically checks if a ticket is a winner.

You are given a **collection of tickets separated by commas and spaces (one or many)**. You need to check each ticket to see if it has a winning combination of symbols:

- A **valid ticket** has **exactly 20 characters**.
- A **winning ticket** is a **valid** one, containing **one of the symbols `'@'`, `'#'`, `'$'` or `'^'` uninterruptedly repeated** at least **6 times** in both **halves of the tickets**.
- In order to win a Jackpot, the ticket should contain **the same winning symbol 10 times** on both sides

An example of a **valid winning ticket**:

`"Cash$$$$$$Ca$$$$$$sh"`

An example of a **Jackpot winning valid ticket**:

`"$$$$$$$$$$$$$$$$$$$$"`

Input

The input will be read from the console. The input consists of a **single line** containing all tickets **separated by commas and one or more white spaces** in the format:

- **`"{ticket}, {ticket}, … {ticket}"`**

Output

Print the result for every ticket in the order of their appearance, each on a separate line in the format:

- If the ticket is invalid: **"invalid ticket"**
- If the ticket is valid, but it is not winning: **"ticket "{ticket}" - no match"**
- If the ticket is valid and winning, but not a Jackpot:
  "ticket "{ticket}" - {uninterrupted_match_length}{match_symbol}"
- If the ticket hits the Jackpot:
  "ticket "{ticket}" - {uninterrupted_match_length}{match_symbol} Jackpot!"

Constraints

- Number of tickets will be in range [0 … 100]

Examples

Input	Output
Cash$$$$$$Ca$$$$$$sh	ticket "Cash$$$$$$Ca$$$$$$sh" - 6$
$$$$$$$$$$$$$$$$$$, aabb , th@@@@@@eemo@@@@@@ey	ticket "$$$$$$$$$$$$$$$$$$" - 10$ Jackpot! invalid ticket ticket "th@@@@@@eemo@@@@@@ey" - 6@
validticketnomatch:(, Cas$$$$$$$Ca$$$$$$s$	ticket "validticketnomatch:(" - no match ticket "Cas$$$$$$$Ca$$$$$$s$" - 6$