More Exercise: Text Processing

Problems for exercise and homework for the ["JS Fundamentals" Course @ SoftUni.](#)

Submit your solutions in the SoftUni judge system at: https://judge.softuni.org/Contests/1707

1. Value of a String

Write a function, which finds the **sum** of the **ASCII codes** of the **letters** in a given **string**. Your tasks will be a bit harder because you will have to find the **sum** of **either** the **lowercase** or the **uppercase** letters.

On the **first** line, you will receive the **string**.

On the **second** line, you will receive **one of two possible inputs**:

- If you receive "**UPPERCASE**" ➔ find the **sum** of **all uppercase English letters** in the previously received string
- If you receive "**LOWERCASE**" ➔ find the **sum** of **all lowercase English letters** in the previously received string

You should **not** sum the **ASCII** codes of any characters, which are **not** letters.

At the end print the sum in the following format:

- **`The total sum is: {sum}`**

Examples

Input	Output
`['HelloFromMyAwesomePROGRAM', 'LOWERCASE']`	`The total sum is: 1539`
`['AC/DC', 'UPPERCASE']`	`The total sum is: 267`

2. Serialize String

You have been tasked to serialize a string. The serialization is done specially, in which a character from that string is saved with the indexes at which it is found.

The input will consist array, containing a single string, which may consist of **ANY ASCII** characters. Your task is to serialize the string in the following way:

`{char}:{index1}/{index2}/{index3}`

The **`char`** will be the **character**, and the **`indexes`** will be the **indexes** it is **found** at in the **string**.

Examples

Input	Output
`["abababa"]`	`a:0/2/4/6` `b:1/3/5`

`["avjavamsdmcalsdm"]`	`a:0/3/5/11`
	`v:1/4`
	`j:2`
	`m:6/9/15`
	`s:7/13`
	`d:8/14`

	`c:10` `l:12`

3. Deserialize String

Write a function, which takes the **output** from the **previous task** and turns it back into a **string**.

Until you receive the line "**end**", you will receive several lines of input on the console, in the following format:

- `"{letter}:{index1}/{index2}/{index…}/{indexN}"`

Your task is to take every **letter** and its **index** and **form a string** out of them.

Examples

Input	Output

`['a:0/2/4/6',`
`'b:1/3/5',`
`'end']`
	`abababa`

`['a:0/3/5/11',`
`'v:1/4',`
`'j:2',`
`'m:6/9/15',`
`'s:7/13',`
`'d:8/14',`
`'c:10',`
`'l:12',`
`'end']`
	`avjavamsdmcalsdm`

4. Ascii Sumator

Write a function that prints a **sum of all characters between two given characters** (their **ASCII code**). On the **first line,** you will get a **character**. On the **second line,** you get **another character**. On the **last line**, you get a **random string**. Find all the characters **between the two given** and **print their ASCII sum**.

Example

Input	Output

`['.',`
`'@',`
`'dsg12gr5653feee5']`
	`363`

`['?',`
`'E',`
`'@ABCEF']`
	`262`

`['a',`
`'1',`
`'jfe392$#@j24ui9ne#@$']`
	`445`

5. Treasure Finder

Write a function that **decrypts a message** by a given **key** and gathers information about the hidden **treasure type** and its **coordinates.** On the **first line,** you will receive a **key (sequence of numbers).**

On the **next few lines until you receive "find"** you will get lines of **strings**. You have to **loop through every string** and **decrease the ASCII code of each character** with a **corresponding number of the key** sequence. The way you choose a key number from the sequence is by just **looping through it**. If the **length of the key** sequence is **less than the string** sequence, you start **looping from the beginning of the key.** For more clarification see the example below. **After decrypting** the message, you will **get a type of treasure and its coordinates.** The **type** will be **between** the symbol **'&'** and the coordinates will be between the symbols **'<'** and **'>'**. For each line, **print the type and the coordinates** in the format:

```
`Found {type} at {coordinates}`
```

Example

Input	Output	Comment
["1 2 1 3", "ikegfp'jpne)bv=41P83X@", "ujfufKt)Tkmyft'duEprsfjqbvfv=53V55XA", "find"]	**Found gold at 10N70W Found Silver at 32S43W**	We start looping through the first string and the key. When we reach the end of the key we start looping from the beginning of the key, but we continue looping through the string. (until the string is over) The first message is: **"hidden&gold&at<10N70W>"** so we print we found gold at the given coordinates We do the same for the second string **"thereIs&Silver&atCoordinates<32S43W>"**(starting from the beginning of the key and the beginning of the string)
["1 4 2 5 3 2 1", \`Ulgwh"jt$ozfj!'kqqg(!bx"A3U237GC\`, "tsojPqsf$(lrne'$CYfqpshksdvfT$>634057YC", "'stj)>34W68Z@", "find"]	**Found gold at 0S123E Found gold at 102N43W Found ore at 23S43W**	

6. Melrah Shake

You are given a **string** of random characters and a **pattern** of random characters. You need to “shake off” (**remove**) all of the **border** occurrences of that pattern, in other words, the **very first match** and the **very last match** of the pattern you find in the string.

When you successfully shake off a match, you **remove** from the pattern the character which corresponds to the **index** equal to **the pattern’s length / 2**. Then you continue to shake off the border occurrences of the new pattern until the pattern becomes **empty** or until there is **less** than the - needed for a shake, matches in the remaining string.

In case you have found at least **two** matches, and you have successfully shaken them off, you print "**`Shaked it.`**" on the console. Otherwise, you print "**`No shake.`**", the remains of the main string, and you end the program. See the examples for more info.

Input

- The input will consist only of two lines
- On the first line, you will get a string of random characters
- On the second line, you will receive the pattern and that ends the input sequence

Output

- You must print "**`Shaked it.`**" for every time you successfully do the melrah shake
- If the melrah shake fails, you print "**`No shake.`**", and on the next line you print what has remained of the main string

Constraints

- The two strings may contain **ANY** ASCII character
- Allowed time/memory: 250ms/16MB

Examples

Input	Output

`['astalavista baby',`
`'sta']`
	`Shaked it.`
	`No shake.`
	`alavi baby`

`['##mtm!!mm.mm*mtm.#',`
`'mtm']`
	`Shaked it.`
	`Shaked it.`
	`No shake.`
	`##!!.*.#`