More Exercises: Lists Advanced

Additional exercises for the Python Fundamentals Course @SoftUni.

Submit your solutions in the SoftUni judge system at https://judge.softuni.org/Contests/1732

Note: All the exercises are excluded from your homework!

1. Social Distribution

A core idea of several left-wing ideologies is that the wealthiest should support the poorest, no matter what, and that is exactly what you are called to do for this problem.

On the first line, you will be given the **population** (numbers separated by comma and space `", "`). On the second line, you will be given the **minimum wealth**. You should **distribute** the wealth so that no part of the population has **less than the minimum** wealth. To do that, you should always take wealth from the **wealthiest part of the population**.

There **will be cases** where the distribution will **not be possible**. In that case, print: **`"No equal distribution possible"`**.

Example

Input	Output

`2, 3, 5, 15, 75`
`5`
	`[5, 5, 5, 15, 70]`

`2, 3, 5, 15, 75`
`20`
	`[20, 20, 20, 20, 20]`

`2, 3, 5, 45, 45`
`30`
	`No equal distribution possible`

2. Take/Skip Rope

Write a program, which reads a **string** and **skips** through it, extracting a **hidden message**. The algorithm you should implement is as follows:

Let us take the string "**`skipTest_String044160`**" as an example.

Take every **digit** from the string and **transfer it** somewhere. After this operation, you should have **two lists of items** - a **numbers list** and a **non-numbers list**:

- Numbers' list: **`[0, 4, 4, 1, 6, 0]`**
- Non-numbers: **`[s, k, i, p, T, e, s, t, _, S, t, r, i, n, g]`**

After that, take every digit in the **numbers list** and split it up into a **take list** and a **skip list**. In the **take** list, you should keep all digits at an **even** index. In the **skip** list, you should keep all digits at an **odd** index.

- Numbers' list: **`[0, 4, 4, 1, 6, 0]`**
- Take list: **`[0, 4, 6]`**
- Skip list: **`[4, 1, 0]`**

Afterward, **iterate over both lists**:

- **First**, **`take`** **m** characters from the **non-numbers list** and store it in a **result string**
- **Then, `skip` n** characters from the **non-numbers list**

Note that the skipped characters are **summed up** as they go. The process would look like this:

1. Current string: **"skipTest_String"**. Take **0** characters and skip **4** characters:
    - Taken string: **""**
    - Skipped string: **"skip"**
2. The remaining string looks like this: **"Test_String"**. Take **4** characters and skip **1** character:
    - Taken string: **"Test"**
    - Skipped string: **"_"**
3. The string looks like this: **"String"**. Take **6** characters and skip **0** characters:
    - Taken string: **"String"**
    - Skipped string: **""**
4. The final string is **"TestString"**.

After that, print the **final string** on the console.

Constraints

- The count of digits in the input string will **always be even**.
- The encrypted message will contain any printable ASCII character.

Examples

Input	Output
T2exs15ti23ng1_3cT1h3e0_Roppe	TestingTheRope
O{1ne1T2021wf312o13Th111xreve!!@!	OneTwoThree!!!
this forbidden mess of an age rating 0127504740	hidden message

3. Kate's Way Out

Kate is stuck in a maze. You should help her to find her way out.

On the **first line,** you will be given how many **rows** there are in the maze. On the **following n lines,** you will be given the **maze itself**. Here is a legend for the maze:

- **"#"** - means a **wall**; Kate cannot go through there
- **" "** - means **empty** space; Kate can go through there
- **"k"** - the initial **position of Kate**; start looking for a way out from there

There are two options: Kate either gets out or not:

- If Kate **can get** out, print the following:
  "Kate got out in {number_of_moves} moves".
  Note: If there are **two or more ways** out, she **always** chooses **the longest one**.
- Otherwise, print: **"Kate cannot get out"**.

Examples

Input	Output
4 ###### ## k# ## ###	Kate got out in 5 moves

`## ###`	

`5`
`######`
`##  k#`
`## ###`
`######`
`## ###`
	`Kate cannot get out`

4. Battle Ships

You will be given a number **n** representing the number of **rows of the field**. On the following **n** lines, you will receive **each field row** as a **string** with **numbers separated by a space**. Each number greater than zero represents a **ship** with **health** equal to the **number value**.

After that, you will receive the **squares** that are being **attacked** in the format: **`"{row}-{col} {row}-{col}"`**. Each time a square is being attacked, if there is a ship (number greater than 0), you should **reduce its value by 1**. If a ship's health **reaches zero**, it is **destroyed**. After the attacks have ended, print **how many ships** were **destroyed**.

Example

Input	Output	Comment

`3`
`1 0 0 1`
`2 0 0 0`
`0 3 0 1`
`0-0 1-0 2-1 2-1 2-1 1-1 2-1`
	`2`	States after each attack:
		First attack -> 1 ship destroyed
		0 0 0 1
		2 0 0 0
		0 3 0 1
		Second attack -> reduce ship health
		0 0 0 1
		1 0 0 0
		0 2 0 1
		Third attack -> reduce ship health
		0 0 0 1
		2 0 0 0
		0 2 0 1
		Fourth attack -> reduce ship health
		0 0 0 1
		2 0 0 0
		0 1 0 1
		Fifth attack -> another ship destroyed
		0 0 0 1
		2 0 0 0
		0 0 0 1
		Sixth and Seventh attack -> no ship destroyed

`5`
`1 0 5 0 1`
`6 3 9 0 0`
`7 9 4 3 2`
`1 0 0 4 9`
`5 6 0 3 5`
`0-1 0-2 0-2 0-2 0-2 0-2 3-0`
	`2`	

5. Dots

You will be given a number **n** representing the number of **rows of a board of dots and dashes**. On the following **n** lines, you will receive **each row** of the board as a **string** with symbols (dots and dashes only), separated by a **single space**.

Your task is to find and print the **largest count of dots** that could be connected **at once**. You could only connect **horizontally or vertically.**

Example

Input	Output
5 . . - - - . . - - - - - - - - - - - . . - - - . .	4
6 . . - . - . - . . . . . - . - - - - - . . - - - - . . . . - - - - . . -	18
4 - . - . . - . - . . - . . - - - - - - - - . - -	4