# Python Advanced: Exam Preparation

## Monster Extermination

[**Link to Judge**](https://judge.softuni.org/Contests/Practice/Index/4089#0)

*You are tasked with simulating a battle between a brave soldier and a group of terrifying monsters. The soldier has a striking impact, and the monsters have different armor levels. Your task is to write a program that takes two sequences of integers as input from the console and performs the battle simulation.*

There will be given **two sequences of integers**.

The **first sequence** represents the **armor of the monsters**. **Each integer value** represents the **armor of a different monster**.

The **second sequence** represents the **soldier's striking impact**. **Each integer value** represents the **strength of a strike** performed by the soldier.

**Battle Rules:**

* The monster at the front will be the first to face the soldier. Take the **first armor value** and the **last strike strength value** and compare the values.
* If the **soldier's striking impact** **is greater than or equal to the monster's armor**, the monster is killed, and its **armor is removed from the sequence**. The soldier's strike **impact is then decreased by the value of the monster's armor**. The **remaining striking impact value** is **added to the next strike element** in the sequence (**if any**) or is considered to be the **last and only element**. **Zero values** should **not** be pushed back to the sequence.
* If the **soldier's striking impact is less than the monster's armor**, the strike is performed, but the monster survives with **reduced armor**. The **soldier's striking impact value is removed from the sequence**, and the original strike value decreases the monster's armor value. The **monster is then moved to the back of the sequence**.
* The battle goes on **until one of the sequences becomes empty**.

**Your Task:**

Write a **console application** to simulate the battle as described above. **Implement the battle logic using appropriate data structures** to manage the soldier's striking impact and the monsters' armor values. The program should then **display the appropriate outcome of the battle** based on the rules.

## Input

* The first line will represent the **armor values - integers**, **comma-separated** values**.**
* The second line will represent the **soldier's striking impact values** - **integers**, **comma-separated** values.

## Output

* If **all the monsters are killed**, the program should print on the Console a success message:
  + **"All monsters have been killed!"**
* If **the soldier's striking impact stack becomes empty**, the program should print on the Console a message indicating that the soldier has been defeated:
  + **"The soldier has been defeated."**
* The program should **print on the Console the total number of monsters killed** by the soldier, **on a new line**:
  + **"Total monsters killed: {killed\_monsters}"**

## Constraints

* All the given numbers will be valid **integers** in the range **[1, 100].**
* There will be **no negative inputs**.

## Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 20,15,10  5,15,10,25 | All monsters have been killed!  Total monsters killed: 3 |
| **Comment** | |
| 1. The soldier’s first strike (last in the sequence) (25) is greater than the first monster's armor (20). **Monster 1 is killed**. Its **armor value is removed** from the sequence. The soldiers’ striking impact value is decreased by the monster’s armor value and it remains (5). It is **added to the next element** in the sequence (10) 2. The soldier’s second strike (15 {10 + 5}) is equal to the second monster's armor (15). **Monster 2 is killed**. Its **armor value is removed** from the sequence. The monster’s armor value decreases the soldiers’ striking impact value and it remains (0). It is not necessary to be added to the next element. It just can be **removed from the sequence**. 3. The soldier’s third strike (15) is greater than the third monster's armor (10). Monster 3 is killed. Its **armor value is removed** from the sequence. The monster’s armor sequence is empty, so the program ends, and the **output is printed on the Console.** All monsters have been killed. The soldier wins the battle, having killed 3 monsters. | |
| **Input** | **Output** |
| 30,25,40,35  15,20,10,30 | The soldier has been defeated.  Total monsters killed: 1 |
| **Comment** | |
| 1. The soldier’s first strike (30) is equal to the first monster's armor (30). **Monster 1 is killed**. Its **armor value is removed** from the sequence. The monster’s armor value decreases the soldiers’ striking impact value and it remains (0). It is not necessary to be added to the next element. It just can be **removed from the sequence**. 2. The soldier’s second strike (10) is less than the second monster's armor (25). Monster 2 survives with 15 armors remaining. **Monster 2 moves to the back of the sequence**. 3. The soldier’s third strike (20) is less than the third monster's armor (40). Monster 3 survives with 20 armors remaining. 4. The soldier’s fourth strike (15) is less than the fourth monster's armor (35). Monster 4 survives with 20 armors remaining. 5. The soldier has no more strikes left, but there are still three monsters remaining. The soldier is defeated, having killed 1 monster. | |

## Mouse In The Kitchen

[**Link to Judge**](https://judge.softuni.org/Contests/Practice/Index/4081#1)

*A hungry little mouse is living in an old suburbs house. It walks around the kitchen cupboard every night and eats all the cheese. A lazy plump cat is guarding the kitchen, so the mouse should not walk out of the cupboard area.*

In the beginning, you will be given **N** and **M** – **integers, separated by a comma - ","**, indicating the **cupboard’s area dimensions**. On the next **N** lines, you will receive strings, representing the **rows** of the area, with **M** columns.

After that, on each line, **until** the command **"danger"** appears as an input string, you will receive the **possible directions** for the mouse to move - **"up", "down", "right",** and **"left"**.

If the mouse **steps outside the cupboard area**, the cat will attack, and the **cheese hunt is over**. In that case, **the program ends**, keep the **last known position** of the mouse, **before** it steps outside the cupboard area and the following **message** is printed on the Console: **"No more cheese for tonight!"**

**Possible characters in the matrix** are:

* **M** - represents the mouse's position.
* **C** – represents a piece of cheese.
* **\*** – represents an empty position, nothing happens if the mouse steps on it.
* **@** – represents a wall in the cupboard, cannot step on or go through it.
* **T** – represents a trap.

The mouse starts from the **M - position**.

* If the mouse steps on the **C – position**, it **eats the cheese** from the field, and the **position** **is marked** with **"**\***".** 
  + If this is **the last cheese** in the cupboard area, the mouse goes to sleep. Remember that this will be **the last known position of the mouse**. **The program ends** and the following **message** is printed on the Console: **"Happy mouse! All the cheese is eaten, good night!"**
* If the mouse steps into a **trap (T -position)**, it will be **trapped**. Remember that this will be **the last known position of the mouse**. In that case, the **program ends**, and the following **message** is printed on the Console: **"Mouse is trapped!"**
* If the **given direction** **leads** the mouse towardsthe **@ - position,** this is a wall in the cupboard area. **Do not make the move** and **skip the command**.
* If the **"danger"** command is received **before** the mouse manages to **eat all the cheese**, the mouse disappears. Remember that this will be **the last known position** **of the mouse** and **you will need it for the final state of the matrix**. In that case, the **program ends**, and the following **message** is printed on the Console: **"Mouse will come back later!"**

In the end, **print the final state of the matrix** (cupboard area) with **the last known position of the mouse** in it. **Each row is on a new line**.

### Input

* **On the first line** you will get the **number of rows** and **columns of the matrix, separated by a comma**.
* On the next **N** lines, you will receive strings, representing each **row** of the matrix.
* On each of the following lines, until the command **"danger"** appears as an input string, you will receive the possible directions for the mouse to move - **"up", "down", "right",** and **"left"**.
* **"danger"** command – The mouse spots danger and disappears… for now!

### Output

* **On the first line:**
  + If the mouse **steps outside the cupboard**

**"No more cheese for tonight!"**

* + If the mouse manages to eat all the cheese

**"Happy mouse! All the cheese is eaten, good night!"**

* + If the mouse steps into a **trap (T -position)**

"**Mouse is trapped!**"

* + If the **"danger"** command is received before the mouse manages to eat all the cheese –

**"Mouse will come back later!"**

* On the next lines, print the **final state of the matrix** with the **last known position of the mouse** in it. **Each row - on a new line**, **each row position with NO separator**.

### Constraints

* There will always be **at least one trap** in the cupboard.
* There **will always be some cheese** in the cupboard.
* There will always be а **"danger"** command in the end, but **it is not necessary to reach it**. **The program may end earlier**.
* **Each row** of the matrix will have the **same length**.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 5,5  \*\*M\*\*  T@@\*\*  CC@\*\*  \*\*@@\*  \*\*CC\*  left  down  left  down  down  down  right  danger | Mouse is trapped!  \*\*\*\*\*  M@@\*\*  CC@\*\*  \*\*@@\*  \*\*CC\* | The mouse moves to the left and the position is marked with **"\*"**, so nothing happens.  The next command is **"down"**, but the position is marked with **"@"**, so we skip the command.  Next command - **"left"**, the mouse moves to the left. The position is marked with **"\*"**, so nothing happens.  The next command is **"down"**, the position is marked with **"T"**, so the mouse is trapped. Remember to mark the last known position of the mouse with **"M"**. |
| 4,8  CC@\*\*C\*M  T\*@\*\*CT\*  \*\*@@@@\*\*  T\*\*\*C\*\*\*  down  right  left  down  left  danger | No more cheese for tonight!  CC@\*\*C\*\*  T\*@\*\*CTM  \*\*@@@@\*\*  T\*\*\*C\*\*\* |  |
| 6,3  @CC  @TC  @C\*  @M\*  @\*\*  @\*\*  left  up  left  right  up  up  left  left  danger | Happy mouse! All the cheese is eaten, good night!  @M\*  @T\*  @\*\*  @\*\*  @\*\*  @\*\* |  |

## Movie Organizer

[**Link to Judge**](https://judge.softuni.org/Contests/Practice/Index/3893#2)

*Mary has a large collection of films, from old black-and-white classics to the newest blockbusters. But her collection is an unorganized jumble. She spends hours searching for a particular movie, only to find it in the most unexpected place. Mary needs your help to organize her movies.*

Write a function called "**movie\_organizer**" that **groups movies by genre**. The function will receive a **different number of arguments**, passed as **tuples containing two elements**: the **first** one is the **movie's name**, and the **second** is the **genre** for example ("**Movie Name**", "**Genre**").

The function should **sort the movies by their genre**. Arrange Mary's collection **by the number of movies** in eachgenrein **descending order**. If two or more genres have the **same number of movies**, return them in **ascending order** (alphabetically) **by genre.**

Each **genre group** should be **sorted** in **ascending order** (alphabetically) **by the movie's name**.

To help Mary keep track of her movies, add next to each genre the **number of movies in the group**.

**In the end, return** the output as described below.

***Note: Submit only the function in the judge system***

### Input

* There will be **no input from the console**, just parameters passed to your function

### Output

* The **output** should look like this**:**

**"{genre\_1} - {number\_of\_movies\_in\_the\_genre\_group}**

**\* {movie\_name\_1}**

**\* {movie\_name\_2}**

**\* {movie\_name\_3}**

**…**

**\* {movie\_name\_n}**

**{genre\_2} - {number\_of\_movies\_in\_the\_genre\_group}**

**\* {movie\_name\_1}**

**\* {movie\_name\_2}**

**…**

**\* {movie\_name\_n}**

**{genre\_n} - {number\_of\_movies\_in\_the\_genre\_group}**

**\* {movie\_name\_1}**

**…**

**\* {movie\_name\_n}"**

### Constraints

* Each **tuple** given will always contain a **movie** with its **genre.**
* You will **never** receive the **same movie** twice or more times.

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| print(movie\_organizer(  ("The Matrix", "Sci-fi"))) | Sci-fi - 1  \* The Matrix |
| print(movie\_organizer(  ("The Godfather", "Drama"),  ("The Hangover", "Comedy"),  ("The Shawshank Redemption", "Drama"),  ("The Pursuit of Happiness", "Drama"),  ("The Hangover Part II", "Comedy"))) | Drama - 3  \* The Godfather  \* The Pursuit of Happiness  \* The Shawshank Redemption  Comedy - 2  \* The Hangover  \* The Hangover Part II |
| print(movie\_organizer(  ("Avatar: The Way of Water", "Action"),  ("House Of Gucci", "Drama"),  ("Top Gun", "Action"),  ("Ted", "Comedy"),  ("Duck Soup", "Comedy"),  ("The Dark Knight", "Action"),  ("A Star Is Born", "Musicals"),  ("The Warrior", "Action"),  ("Like A Boss", "Comedy"),  ("The Green Mile", "Drama"),  ("21 Jump Street", "Comedy"))) | Action - 4  \* Avatar: The Way of Water  \* The Dark Knight  \* The Warrior  \* Top Gun  Comedy - 4  \* 21 Jump Street  \* Duck Soup  \* Like A Boss  \* Ted  Drama - 2  \* House Of Gucci  \* The Green Mile  Musicals - 1  \* A Star Is Born |