# Python Advanced: Exam Preparation

[**Link to Judge**](https://alpha.judge.softuni.org/contests/python-advanced-retake-exam-12-august-2024/4818)

## Wild Survival



*Bees: Nature's diligent pollinators, sustaining life with every buzz and bloom. Bee-eaters: Elegant hunters, who balance ecosystems with precision and grace.*

On the **first line**, you will be given a **sequence** containing **integers** representing **bee groups** that live in a **beehive**.

On the **second line**, you will be given **another sequence of integers** representing **bee-eaters groups** living near the **beehive**.

**Bees** and **bee-eaters** are eternal **enemies** and are always fighting. **Bees** are known as the **defenders** of their **hive** while the **bee-eaters** are known as the **attackers**.

Until **there are bees** **and bee-eaters** available, the program will **continue** running.

You need to **compare** the **first** **group** of **bees** to the **last group** of **bee-eaters** (See the [Examples](#_Examples)):

* They start a **fight until at least one of the groups is defeated**.

**One bee-eater can kill 7** (seven) **bees at once**, **per each fight**, then **dies**.

If **one attacker** needs to fight **fewer defenders** in number (**less than 7**), it **survives** while the **defenders** are considered **defeated**. In the **next battle**, it **can kill 7** (seven) **defenders again**.

* + If the **bee-eaters** from the **current** **fighting** group **win** (there are **0** (zero) **remaining bees** in the **corresponding** group) **return** the **survived** **bee-eaters** to the **sequence** (in their initial position). The **defeated bee group is removed**.
  + If the **bees** from the **current** **fighting** group **win**, (there are**0** (zero) **remaining bee-eaters** of the **corresponding group**) **add** the **bees** **that survived** to the **back** of the **bees collection**. The **defeated group** of **bee-eaters is removed**.
  + If the result is a **draw**, **remove** **both groups** from **their collections** and proceed to the next ones.

### Input / Constraints

* On the **first line**, you will receive **integers** representing the **bee groups**, separated by a **single space**. (See the [Examples](#_Examples))
* On the **second line**, you will receive **integers** representing the **bee-eaters groups**, separated by a **single space**. (See the [Examples](#_Examples))
* The given numbers will be **valid positive** **integers** in the range **[1 - 100]** inclusive.

### Output

The output of your program should be **printed** on the **Console**, on **separate lines**, depending on the following **outcome variations**:

* On the **first** line:

**"The final battle is over!"**

* On the **second** line:
  + If **bees** and **bee-eaters** have **slaughtered** each other, print:

**"But no one made it out alive!"**

* + If there are **bees** that **survived**, print:

**"Bee groups left: {bee\_group1, bee\_group2, …, bee\_groupN}"**

* + - Print the **bee** **groups** in their **current** **order**, **separated by comma** and **space** **", "**.
  + If there are **bee-eater** **groups** that have **survived**, print:

**"Bee-eater groups left: {bee\_eaters\_group1, …, bee\_eaters\_groupN}"**

* + - Print the **bee-eater** **groups** in their current order, **separated by comma** and **space** **", "**.

### Examples

|  |  |  |
| --- | --- | --- |
| Input | Output | Comment |
| 32 42 7 28 3  1 5 6 | The final battle is over!  Bee groups left: 21 | We start with the first group of bees and the last group of bee-eaters: 32 vs 6.  They begin to fight until one of the groups defeats the other. Each bee-eater can kill 7 bees in the current battle before dying:  32 vs 6 -> 25 vs 5 -> 18 vs 4 -> 11 vs 3 -> 4 vs 2 -> 0 vs 2  4 fighting bee-eaters die, killing 28 bees (4x7=28)  The last 2 fighting bee-eaters defeat the remaining 4 bees and survive the battle. The bee group is removed from its collection. Finally, the 2 bee-eaters go back to the initial position of their collection:  42 7 28 3  1 5 2  The next fight continues with the next group of bees:  42 vs 2 -> 35 vs 1 -> 28 vs 0  The fighting bees kill all bee-eaters from the group, part of them survive and go to the back of their collection:  7 28 3 28  1 5  The third battle begins with 7 vs 5.  7 vs 5 -> 0 vs 4  Four bee-eaters survive and go back to their initial position in the sequence. 0 bees left, so their group is removed.  28 3 28  1 4  The next battle results in a draw: 28 vs 4 -> 21 vs 3 -> 14 vs 2 -> 7 vs 1 -> 0 vs 0, and no one survives. Both groups are removed.  3 28  1  Next, we have 3 vs 1. The bee-eater wins over the three bees. They are removed and the bee-eater is returned to its collection.  28  1  28 vs 1 -> 21 vs 0  The remaining 21 bees from the group survive and go to the end of their sequence. As there are no bee-eater groups left the program finishes.  21  Finally, only one group of bees survived, containing 21 bees. |
| 21 14 14 7  1 2 2 3 | The final battle is over**!**  But no one made it out alive! | The first group of bees fights the last group of bee-eaters: 21 vs 3.  21 vs 3 -> 14 vs 2 -> 7 vs 1 -> 0 vs 0 results in a draw.  14 14 7  1 2 2  14 vs 2 -> 7 vs 1 -> 0 vs 0 results in another draw.  14 7  1 2  14 vs 2 -> 7 vs 1 -> 0 vs 0 results in a third draw.  7  1  7 vs 1 -> 0 vs 0 results in a final draw. No groups left. |
| 14 6  1 3 2 | The final battle is over!  Bee-eater groups left: 1, 3 | 14 vs 2 results in a draw. Both groups are removed.  6  1 3  6 vs 3 -> 0 vs 3  The last group of 6 bees fights against a group of 3 bee-eaters. Bees are defeated while the 3 attackers survive and return to their initial position in the collection.  1 3  No bee groups left and the program finishes. |
| 29 41 9  1 6 | The final battle is over!  Bee groups left: 27, 2 |  |

## Collecting Stars Game



*In this thrilling adventure, you are a star collector navigating a mysterious, obstacle-filled field, searching for ten elusive stars. Use your wits and skillful movements to gather stars, avoid obstacles, and achieve victory before your stars run out!*

On the **first line**, you will be given an **integer N**, **representing the size** of the **field** with a **square shape**.

On the **following N lines**, you will be given the **field** containing symbols, separated by a single space. See the [**Examples**](#_Examples_1) section.

Your **goal** is to **collect 10 stars**. **10**

The **field will contain randomly positioned elements** - a **player**, **stars**, and **obstacles**:

* **One player**, **marked** with the **letter "P"**
* **Stars**, **marked** with the **asterisk** **symbol "\*"**
* **Obstacles** **marked** with **hashtag symbol** **"#"**

There are **two possible outcomes** of the **game** and **commands** are received until:

* The player **collects** **10 stars** and **wins** the **game**.
* The player **hits obstacles**, **loses all his stars**, and **loses** the **game**.

After the field state, you will be **given commands** for the **player's movement**. **Commands** can be: **"up"**, **"down"**, **"left"**, or **"right"**.

The **player** **starts** the **game with 2 stars initially** **2** and **moves** in the **given direction** with **one step for each command**, **collecting all the stars he comes across or losing a star each time he hits an obstacle**.

The player **can go through** the **same path many times** but **can collect the stars just once** (the first time), while the **obstacles** are **immovable** and **will remain** there. The player **can hit the same obstacle** **many times** and **lose** a **star** **each time** that happens.

**Game rules**:

* When the **player comes across a star** and **collects** **it**, the cell shall be **marked with a dot "."**.
  + The **total number of collected stars** shall be **increased by one**. **+1**
* If the **player encounters an obstacle**, he **does not move** and **remains** in **his current position**.
  + The **player loses one star each time he hits an obstacle**. **-1**
* If the **player** **steps out of the field**, he will be punished by a **teleport** to the **field's starting position** ( coordinates [**0, 0**]).
* When the **player makes his first move**, mark his **initial position** with a **dot** **"."**.
* The **game** **continues until** the player manages to **collect 10 stars** and **wins** **or** **until** he **loses all his stars** by hitting obstaclesand therefore **loses the game**. See the [**Examples**](#_Examples_1) section.
* **At** the **end of the game**, **print** the **final state** of the **field** and **the player's final position**, **marked with "P"**.

### Input

* On the **first line** you will receive an **integer** **N** representing the **size of the square field** (matrix **NxN**).
* On the **next N lines** you will get the **field** **rows** (**each position separated by a single space**)
* On **each of the following lines**, you will get a **valid** **move command**.

### Output

At the **end** of the program:

* If the player **won the game**, print: **"You won! You have collected 10 stars."**
* If the player **loses the game**, print: **"Game over! You are out of any stars."**
* Next, print the **player's final position**: **"Your final position is [{row\_position}, {column\_position}]"**
* Finally, print the **matrix in its final state**, **each position separated by a single space**. Remember to **mark the player's final position with "P"**.

### Constraints

* There will always be **enough commands** to **either win or lose the game**.
* There will be **no case** inwhich **less than 10 stars will be in the field**.
* There will be **no obstacle** at the **field's** **starting position** (coordinates [0,0])
* All **given symbols** will be **valid** following the **description**.

**See the Examples section below**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  \* # \* \* \*  # \* \* \* \*  \* \* \* \* #  P \* \* \* \*  \* \* \* # \*  right  right  right  up  up  up  right  down  left  left  left  down | You won! You have collected 10 stars.  Your final position is [1, 4]  \* # \* . .  # \* \* . P  \* \* \* . #  . . . . \*  \* \* \* # \* |
| **Comment** | |
| The **program starts** with the **player placed at coordinates [3, 0]** and **having 2 stars initially**.  The commands are processed as follows:  right: Moves from [3, 0] to [3, 1], collects one star and now has **3** stars in total  right: Moves from [3, 1] to [3, 2], collects one star and now has **4** stars in total  right: Moves from [3, 2] to [3, 3], collects one star and now has **5** stars in total  up: Moves from [3, 3] to [2, 3], collects one star, and now has **6** stars in total  up: Moves from [2, 3] to [1, 3], collects one star, and now has **7** stars in total  up: Moves from [1, 3] to [0, 3], collects one star, and now has **8** stars in total  right: Moves from [0, 3] to [0, 4], collects one star and now has **9** stars in total  down: Moves from [0, 4] to [1, 4], collects one star and now has **10** stars in total  **Win**: The **player collected 10 stars** and **finished the game at [1, 4]**. The player's **path** was **marked with dots** as he **collected stars** on **each move**.  The **appropriate messages** are **printed** indicating the **success**. | |
| **Input** | **Output** |
| 4  \* # \* \*  # P # \*  \* \* \* \*  \* \* \* \*  up  down  up  up  up  up  down  left  left  left  down | Game over! You are out of any stars.  Your final position is [1, 1]  \* # \* \*  # P # \*  \* . \* \*  \* \* \* \* |
| **Input** | **Output** |
| 4  \* \* # \*  \* \* # \*  P \* # \*  # # \* \*  right  up  left  left  down  up  right  right  right  right  right  up  right  right  right  right  right | Game over! You are out of any stars.  Your final position is [0, 1]  . P # \*  . . # \*  . . # \*  # # \* \* |

## Guest Accommodation



*Your system should match guests to available rooms, balancing comfort and efficiency. While some guests may not find a room, you must prioritize the seamless and well-organized process!*

Write a function named **"accommodate"** that **receives information** about **guests** waiting for accommodation and the available **rooms**.

The function will receive an **unknown number of arguments and keyword arguments**:

* The **arguments** will be passed as follows:
  + **Guest groups** as integers in the range **[1-10]** inclusive. (See the [**Examples**](#_Examples_2))
* The **keyword arguments** will be passed as follows:
  + **Room** **number as a key** - a unique string in the format **"room\_{number}"**.
    - The number contains **exactly 3** **digits**, in therange **[100-999]** inclusive.
  + **Room capacity as a value** - an integer in the range **[1-6]** inclusive.

**Example: room\_101=5**

The program runs until **all groups of guests** have **tried to complete** the **check-in** process at the reception area.

The receptionist tries to accommodate the **groups of guests** in their **initial order**, **starting** with the **first group**, and **searches for** the **most suitable** **room**.

* **Best fit rule**: A room is considered a **best fit** if it has a **capacity equal to the number of guests**. If there is **no such room**, choose the one with the **smallest capacity** that is **greater than the guests' number**. If multiple rooms have the **same capacity**, pick the one with the **smallest room number**.

**Hint**: Order the available rooms by capacity ascending, then by room number ascending.

* Once a group is **accommodated in a room**, the **room is no longer available**.
* If **no suitable room** is **found** (the number of guests does not fit in any available room), the group remains **unaccommodated**.
  + **Keep track** of the **number of guests** with **unsuccessful accommodations**.

Once **all guests** have **tried to** **complete** the check-in process, the hotel software **displays** the result.

* If there are **any successful accommodations**, **sort** them by the **room number ascending**.
* **Return** the output as described below. (See the [**Output**](#_Output_1) section)

***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

* If there are **accommodations**, **return** them **sorted** by **room number** and in the following format:

**"A total of {total\_number\_of\_accommodations} accommodations were completed!**

**<Room {room\_number1} accommodates {guests1} guests>**

**<Room {room\_number2} accommodates {guests2} guests>**

**…**

**<Room {room\_numberN} accommodates {guestsN} guests>"**

**Note:** If the room is "**room\_101**" the number of the room is **101**.

* Otherwise, **return** the **message**:

**"No accommodations were completed!"**

The **output string** should also contain information for the **guests without accommodation** and the **empty hotel rooms if there are any**:

* If there are **guests** **without** **accommodation**, **return** a **message**:

**"Guests with no accommodation: {total\_number\_of\_unaccommodated\_guests}"**

* If there are **empty rooms**, **return** a **message**:

**"Empty rooms: {total\_number\_of\_empty\_rooms}"**

### Constraints

* The **arguments** will always be **integers** in the **range [1-10]** inclusive.
* The **keyword** **arguments** will always have a **key** as a **string** in the format **"room\_{number}"** and a **value** as an **integer** in the **range [1-6]** inclusive.
* The **room numbers** will be **unique** andcontain **exactly 3 digits** in therange **[100-999]** inclusive.
* There will always be **at least** **one** **argument** and **at least one** **keyword** **argument**.

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| print(accommodate(5, 4, 2, room\_305=6, room\_410=5, room\_204=2)) | A total of 3 accommodations were completed!  <Room 204 accommodates 2 guests>  <Room 305 accommodates 4 guests>  <Room 410 accommodates 5 guests> |
| print(accommodate(10, 9, 8, room\_307=6, room\_802=5)) | No accommodations were completed!  Guests with no accommodation:27  Empty rooms: 2 |
| print(accommodate(1, 2, 4, 8, room\_102=3, room\_101=1, room\_103=2)) | A total of 2 accommodations were completed!  <Room 101 accommodates 1 guests>  <Room 103 accommodates 2 guests>  Guests with no accommodation: 12  Empty rooms: 1 |