# Python Advanced: Exam Preparation

# Stewards

**Link to Judge:** <https://judge.softuni.org/Contests/Practice/Index/3534#0>

*As you know, stewards are needed for every single flight. Today you will be that one steward, and you will be assisting the passengers in finding their seats.*

You will be given a **sequence of 6 seats** - every seat is a **mix** of **a** **number and a letter** in the format **"{number}{letter}"**. You will also be given two more sequences of **numbers only**.

First, you have to take the **first** number of the **first** sequence and the **last** number of the **second** sequence. Next, take the **sum** of those **two numbers** and find its **ASCII character**.

* Compare **each** of the **two taken numbers** and the **found character** with the **seats**. If you find a match, the **passenger is seated**, and the seat is considered taken. **Remove** **both numbers** from their sequences.
* If there is **no equality**, the two numbers should be returned **at the end of their sequences** (first becomes last, last becomes first).
* If you **match an already taken seat**, you should just **remove both numbers** from their sequences.

**Each time** you take numbers from the sequences and try to match them, **you make one rotation**. You should **keep track of all rotations made**.

The **program should** **end** under the following circumstances:

* You have found **3** (three) **seat** **matches**
* You have made a total of **10 rotations**

### Input

* On the **first line,** you will be given a sequence of seats - **strings** separated by comma and space **", "**
* On the **second** and the **third line,** you will be given two more sequences - **integers** separated by a comma and a space **", "**

### Output

When the program ends, print the following on **two** **different lines**:

* **Seat matches: {matches separated by comma and space}**
* **Rotations count: {total rotations made}**

### Constraints

* All integers will be in the range **[1, 100]**
* All letters will be in the range **[A-Z]**
* You will **never** **run out of numbers** in your sequences before the program ends
* You will never have **more than one match** at a time

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 17K, 20B, 3C, 15D, 31Z, 28F  20, 35, 15, 3, 2, 10  1, 15, 64, 53, 45, 46 | Seat matches: 20B, 15D, 3C  Rotations count: 4 |
| **Comment** | |
| 1) Take the first number from the first sequence (20) and the last number from the second sequence (46). Then, sum the two numbers (66) - its ASCII character is "B". Check both combinations "20B" and "46B" with the seats. The seat "20B" matches the first combination, so remove both numbers (20 and 46) from the sequences and take the seat - it's no longer available for matching.  2) Take the next numbers - 35 and 45. Their sum matches the character "P". There are no matches. Return both numbers to the opposite side of the sequences. The sequences look the following way: "15, 3, 2, 10, 35" and "45, 1, 15, 64, 53"  3) Take the following numbers - 15 and 53. Their sum matches the character "D". The seat "15D" is matched. Remove numbers 15 and 53 from their sequences.  4) Take the following numbers - 3 and 64. Their sum matches "C". The seat "3C" is matched. Remove the numbers 3 and 64 from their sequences.  Three seats are matched - print the needed information and end the program. | |
| **Input** | **Output** |
| 25A, 16B, 44T, 49D, 27M, 44F  25, 3, 31, 49, 26, 13  10, 15, 44, 40 | Seat matches: 25A, 44F  Rotations count: 10 |
| **Input** | **Output** |
| 15C, 25C, 36C, 43P, 40E, 38G  15, 25, 80, 40, 15, 99, 52  15, 42, 29 | Seat matches: 25C, 40E, 15C  Rotations count: 7 |

# Exit Founder

**Link to Judge:** <https://judge.softuni.org/Contests/Practice/Index/3515#1>

*Tom and Jerry decided to play a game together. The game is a maze of which they need to find a way out. Monitor their moves closely and find out who the winner will be!*

First, you will be given the names **"Tom"** and **"Jerry"**, separated by a comma and a space **", "**. The order in which they are received determines the order in which they will take turns. The **first player starts first**.

Next, you will be given a matrix with **6 rows and 6 columns** representing the **maze board**. It consists of:

* **Only one Exit** - marked with the **"E"** letter
* **Trap** (one, many, or none) - marked with the **"T"** letter
* **Wall** (one, many, or none) - marked with the **"W"** letter
* **Empty positions** will be marked with **"."**

In the beginning, Tom and Jerry are **outside the board**. On **each line**, after the matrix is given, you will be receiving **coordinates** for **each of the players**. They will be **taking turns and stepping on different positions on the board** until **one of them** **find the Exit** or **falls into a Trap**. Here are the rules:

* If a player hits the letter **"E"**, he escapes the maze and **wins the game**.
  + Print **"{player} found the Exit and wins the game!"** and end the program.
* If the letter **"T"** is hit, the player falls into a Trap, **the game ends**, and **his** **opponent** **wins** automatically.
  + Print **"{player} is out of the game! The winner is {winner}."** and end the program.
* If the letter **"W"** is hit, the player hits a wall, and he needs to rest. The **player's next move is ignored**.
  + Print **"{player} hits a wall and needs to rest."**
* If a player steps on an **empty position** **"."**, nothing happens.
* Both players **can** step in the **same position** at the **same time**.

### Input

* On the first line, you will receive "**Tom"** and "**Jerry"** separated by **", "**. **The first player starts first.**
* **On the following 6 lines,** you will receive the **maze** **board** (**elements** will be **separated by a space**)
* On the following lines, you will be receiving **coordinates** in the format: **"({row}, {column})"**

### Output

* You should print the output as described above.
* The input coordinates will always be valid.

### Еxamples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| Tom, Jerry  . . T . . .  . . . . . .  . . W . . .  . . W . . E  . . . . . .  . T . W . .  (3, 2)  (1, 3)  (5, 1)  (5, 1) | Tom hits a wall and needs to rest.  Jerry is out of the game! The winner is Tom. | First is Tom. He moves to position (3, 2). He hits a wall and needs to rest.  Next is Jerry. He moves to position (1, 3). It is an empty position.  Tom's next move (5, 1) is ignored because he is resting.  Jerry moves to (5, 1). There is a trap, so he is out of the game. The program ends. |
| Jerry, Tom  . T . . . W  . . . . T .  . W . . . T  . T . E . .  . . . . . T  . . T . . .  (1, 1)  (3, 0)  (3, 3) | Jerry found the Exit and wins the game! |  |
| Jerry, Tom  . . . W . .  . . T T . .  . . . . . .  . T . W . .  W . . . E .  . . . W . .  (0, 3)  (3, 3)  (1, 3)  (2, 2)  (3, 5)  (4, 0)  (5, 3)  (3, 1)  (4, 4)  (4, 4) | Jerry hits a wall and needs to rest.  Tom hits a wall and needs to rest.  Tom hits a wall and needs to rest.  Jerry hits a wall and needs to rest.  Tom found the Exit and wins the game! |  |

# Springtime

**Link to Judge:** <https://judge.softuni.org/Contests/Practice/Index/3374#2>

*Spring is the season of new beginnings. Fresh buds bloom, animals awaken and the earth seems to come to life again. Farmers and gardeners plant their seeds and temperatures slowly rise.*

Write a function called **start\_spring** which will **receive a different number of keyword arguments.**

Each **keyword** holds a **key** with a **name of the spring object** (string),andeach **value** holds **its type** (string). For example, **dahlia="flower"**, **shrikes="bird"**, **dogwood="tree"**.

The function should **sort** the given spring objects in collections **by their type**:

* The collections **sorted by their number of elements** in descending order. If two or more **collections** have **the same number of elements** in them, return them in **ascending order** (alphabetically) by the **type's name**.
* **Each** **collection's** **elements** should be sorted in **ascending order** (alphabetically) by the **object's name**.

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

### Input

* There will be **no input**. Just parameters passed to your function.

### Output

* **Return** the result, sorted as **described above** in the **format:**
  + **"{type\_one}:**

**-{spring\_object\_of\_this\_type\_one}**

**-{spring\_object\_of\_this\_type\_two}**

**…**

**-{spring\_object\_of\_this\_type\_N}**

**{type\_two}:**

**…**

**{type\_N}:**

**…**

**-{last\_spring\_object\_of\_typeN}"**

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| example\_objects = {"Water Lilly": "flower",  "Swifts": "bird",  "Callery Pear": "tree",  "Swallows": "bird",  "Dahlia": "flower",  "Tulip": "flower",}  print(start\_spring(\*\*example\_objects)) | flower:  -Dahlia  -Tulip  -Water Lilly  bird:  -Swallows  -Swifts  tree:  -Callery Pear |
| example\_objects = {"Swallow": "bird",  "Thrushes": "bird",  "Woodpeckers": "bird",  "Swallows": "bird",  "Warblers": "bird",  "Shrikes": "bird",}  print(start\_spring(\*\*example\_objects)) | bird:  -Shrikes  -Swallow  -Swallows  -Thrushes  -Warblers  -Woodpeckers |
| example\_objects = {"Magnolia": "tree",  "Swallow": "bird",  "Thrushes": "bird",  "Pear": "tree",  "Cherries": "tree",  "Shrikes": "bird",  "Butterfly": "insect"}  print(start\_spring(\*\*example\_objects)) | bird:  -Shrikes  -Swallow  -Thrushes  tree:  -Cherries  -Magnolia  -Pear  insect:  -Butterfly |