# Python Advanced: Exam Preparation

## Offroad Challenge

**[Link to Judge](https://judge.softuni.org/Contests/Practice/Index/4193" \l "0)**

*John is quite an avid off-road fan. He bought a new jeep and made the necessary improvements to it. John is ready for new off-road adventures and can't wait to get started. In this challenge, he must save his fuel very carefully…*

There will be **two sequences of integers**. The first sequence will represent the **initial fuel** and the second - **additional consumption index** due to thin air at high altitudes, hence higher fuel consumption. There will also be a third **sequence of integers**, representing values equal to the necessary **amount of fuel needed** to reach the corresponding altitude in the challenge.

Your task is to take the last **fuel quantity** from the **fuel sequence** and the firstindexfrom **the additional consumption index sequence**. **Subtract the values** and **check the result**.

* The corresponding altitude is **reached if the calculated result is bigger or equal to the first element** from the **needed amount of fuel** sequence. You need to **remove both the fuel and the consumption** indexfrom their sequences as well as the **needed amount of fuel** index from their sequence.
* If the calculated result **is smaller or not equal** to the first element from **the needed amount of fuel** sequence, the corresponding altitude is not reached, movement cannot continue, and the program should end.

### Input

* The first line will represent the initial **fuel** – **integers**, separated by a **single space.**
* The second line will represent the **consumption indexes** that decrease initial **fuel** – **integers**, separated by a **single space**.
* The third line will represent the **quantities** needed to reach the corresponding altitude – **integers**, separated by a **single space**.

### Output

* On the **first** or the next **n** lines, output the corresponding message on the console from the following options:
* If John **reaches the altitude**, print the message:
* **"John has reached: Altitude 1"**
* **…**
* **"John has reached: Altitude {n}"**
* If John **fails to reach the altitude**, print the message:
* **"John did not reach: Altitude {n}"**
* On the **next** lines, based on whether he reached the top or not, print the following on the console in the specified format:
* If John **doesn't have enough fuel to reach the top** but **has reached some altitude**, display the following messages:
* **"John failed to reach the top.**

**Reached altitudes: Altitude 1, … Altitude {n}"**

* If John **does not have enough fuel to reach the top** and **has not reached any altitude**, print:
  + **"John failed to reach the top.**

**John didn't reach any altitude."**

* If John manages to **reach all the altitudes**, he will reach the top. End the program and **print on the console** the following message:
  + **"John has reached all the altitudes and managed to reach the top!"**

### Constraints

* All the given numbers will be valid **integers** in the range **[1, 200].**
* All sequences always consist of **four** elements.
* There will be **no negative input**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| **200 90 40** 100  20 **40 30 50**  **50 60 80 90** | John has reached: Altitude 1  John did not reach: Altitude 2  John failed to reach the top.  Reached altitudes: Altitude 1 |
| **Comment** | |
| We start by taking the last fuel quantity from the fuel sequence (**100**) and the first **additional consumption index from the consumption index fuel sequence** (**20**). The result from subtraction is **100 - 20 = 80**. After that, we check if the sum equals or exceeds the **first amount of needed fuel**. The **result (80)** is more than the **needed fuel (50)** for this altitude, so the **altitude is reached.** As the altitude is reached, we **remove an element from every sequence**. We continue with the next altitude to do the same and as a result, we have **40 – 40 = 0**. The needed fuel is **60**, we do not have enough fuel to reach the current altitude, so the challenge for John ends here. | |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 40 66 123 100  10 30 70 33  40 55 77 100 | John has reached: Altitude 1  John has reached: Altitude 2  John did not reach: Altitude 3  John failed to reach the top.  Reached altitudes: Altitude 1, Altitude 2 |
| **Comment** | |
| Here we take the last **fuel** quantity and like in the previous case subtract the **consumption index** from the fuel and continue forward until the result is equal to or greater than the required fuel otherwise the program stops. | |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 199 190 100 100  20 40 30 50  50 60 70 80 | John has reached: Altitude 1  John has reached: Altitude 2  John has reached: Altitude 3  John has reached: Altitude 4  John has reached all the altitudes and managed to reach the top! |
| **Comment** | |
| Here all altitudes are conquered, and John successfully reaches the top. | |
|  | |

## The Gambler

**[Link to Judge](https://judge.softuni.org/Contests/Practice/Index/4226" \l "1)**

You will be given an integer **n** for the size of the **game board** (square shape). On the next **n** lines, you will receive the rows of the **board**. The gambler will start at a **random** position, marked with the letter '**G**' and have an **initial** **'entering the game'** amount of **100$**.

On each turn, **until command 'end'** is received, you will receive commands for the **direction**, in which the gambler should move. The commands will be "**up**", "**down**", "**left**" and "**right**".

* If a position with **'-' (dash)** is reached, it means that the field is empty and the gambler awaits its next direction.
* If the **position marked** with the letter **'W'** is reached the gambler takes it and adds **100$** to his amount
* If the **position marked** with the letter **'P'** (penalty) is reached **decrease** the gambler's total amount by **200$**
* If the **position marked** with the letter **'J'** is reached the gambler **wins** the jackpot and adds **100000$** to his amount and the game ends.
* If **the gambler leaves** the **boundaries of the board or** his **total amount** becomes **equal to or drops** below **0** (zero)**, he loses everything** and you should stop the program.

The current gambler position should be marked with **'G'**

When the gambler leaves a position marked with a letter it should be replaced by **'-' (dash)**

The program **ends** when one of **these four events** occurs:

* the gambler **leaves** the board boundaries
* command **'end'** is received
* the gambler's total winning amount is **equal to or drops below 0(zero)**
* the position **marked** with **'J'** is reached

### Input

* On the first line, you are given the integer **n** – the size of the matrix (board).
* The **next n lines** hold the values for every **row**.
* On each of the next lines, you will get a direction command.

### Output

* If you win the jackpot on the first and second lines print:
* **"You win the Jackpot!**

**End of the game. Total amount: {amount}$"**

* If you do not win the jackpot print:
* **"End of the game. Total amount: {amount}$"**
* If you leave the boundaries of the matrix or the gambler's amount becomes 0(zero) or below print:
* **"Game over! You lost everything!"**

### Constraints

* The **square matrix** (board) size will be between **[4…10].**
* Gambler's starting position will always be marked with '**G**'.
* There will always be a field marked with **'W'** and it may appear more than once.
* There will be always one field marked with '**J**'.
* There will always be one or two fields marked with **'P'**.
* You will always receive enough commands to end the game.
* Finally if **you** **have any amount print** the **matrix.**

### Examples

|  |  |  |
| --- | --- | --- |
| ****Input**** | ****Output**** | ****Comment**** |
| 4  W-GW  W--W  --P-  ----  down  down  end | **Game over! You lost everything!** | The movement starts from position **[0,2]** after receiving the command "**down**" the gambler moves to position **[1,2]** where there is a **'-'** (dash) field - nothing is happening. The next command is "**down**" again, the gambler lands on a **'P'** (penalty) field and since he has to pay **200$** his sum becomes negative (100 – 200 = -100) and therefore loses it. The game ends. |
| 4  G---  WWWW  P---  PJ--  right  right  right  down  left  left  end | **End of the game. Total amount: 400$**  **----**  **WG--**  **P---**  **PJ--** |  |
| 4  ---G  W-W-  ---P  --JW  left  down  down  down  right  end | **You win the Jackpot!**  **End of the game. Total amount: 100200$**  **----**  **W---**  **---P**  **--GW** |  |
|  |  |  |

## Team Lineup

**[Link to Judge](https://judge.softuni.org/Contests/Practice/Index/4193" \l "2)**

*Uniting nations through teamwork and excellence.  
Embracing diversity, forging victory.*

Write a function called **team\_lineup** that **receives information** about certain **football** **players** and their **countries** and **returns a sorted result**. The function will receive a **tuple of key-value pairs** as **arguments**. The arguments will be passed as follows:

* The following arguments will be the **tuples with two elements** - the **first** one is the **player’s name (string)**, and the **second** one is the **county’s name (string)**.

First, you need to sort the given information as stated below:

* **Sort** the data by the **number** **of** **players** per country (**descending**);
* If the player count is the same in two or more countries, **sort** the data by **country name (alphabetically)**.

**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:

**"{country\_name}:"**

**" -{player1}"**

**" -{player2}"**

**" -{playerN}"**

**\* Please note that there are exactly two intervals before the player’s name.**

### Constraints

* Each **tuple** given will always contain the **player’s name** with its **national country**.
* You **will** **NOT** receive the **same** **player** in two or more **different** **countries**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| print(team\_lineup(  ("Harry Kane", "England"),   ("Manuel Neuer", "Germany"),   ("Raheem Sterling", "England"),   ("Toni Kroos", "Germany"),  ("Cristiano Ronaldo", "Portugal"),  ("Thomas Muller", "Germany"))) | Germany:  -Manuel Neuer  -Toni Kroos  -Thomas Muller  England:  -Harry Kane  -Raheem Sterling  Portugal:  -Cristiano Ronaldo |
| print(team\_lineup(  ("Lionel Messi", "Argentina"),  ("Neymar", "Brazil"),  ("Cristiano Ronaldo", "Portugal"),  ("Harry Kane", "England"),  ("Kylian Mbappe", "France"),  ("Raheem Sterling", "England"))) | England:  -Harry Kane  -Raheem Sterling  Argentina:  -Lionel Messi  Brazil:  -Neymar  France:  -Kylian Mbappe  Portugal:  -Cristiano Ronaldo |
| print(team\_lineup(  ("Harry Kane", "England"),  ("Manuel Neuer", "Germany"),  ("Raheem Sterling", "England"),  ("Toni Kroos", "Germany"),  ("Cristiano Ronaldo", "Portugal"),  ("Thomas Muller", "Germany"),  ("Bruno Fernandes", "Portugal"),  ("Bernardo Silva", "Portugal"),  ("Harry Maguire", "England"))) | England:  -Harry Kane  -Raheem Sterling  -Harry Maguire  Germany:  -Manuel Neuer  -Toni Kroos  -Thomas Muller  Portugal:  -Cristiano Ronaldo  -Bruno Fernandes  -Bernardo Silva |