# Python Advanced: Exam Preparation

# Climb the Peaks

**Link to Judge:** <https://judge.softuni.org/Contests/Practice/Index/3744#0>

*Alex is a vlogger and he wants to make videos of climbing the five highest peaks in Pirin mountain in just* ***one week****. He will take his video set, a tent, and his* ***backpack*** *to the mountain. The backpack fits* ***food portions*** *for one week, exactly. His* ***daily stamina*** *is also limited. Your task is to trace his adventure and create a post for his profile @alaroundtheworld, at the end of the journey.*

You will have to keep information for **all the conquered peaks** if any.

Every day, Alex will **use one portion** of his **daily food supplies** and will **exhaust one** of his **daily stamina**.

First, you will be given **a sequence of numbers, representing the quantities of the daily portions** of food supplies in his backpack.

Afterward, you will be given another **sequence of numbers, representing the quantities of the daily stamina** he will have at his disposal for the next **seven days**.

You have to **sum** the **quantity of the** **last daily food portion** with the **quantity of the first daily stamina**. He will start climbing **from the first** peak in the table below **to the last** one.

* If the **sum is equal to or greater** than the corresponding **Mountain Peak’s Difficulty level from the table below**, it means that the **peak is conquered**. In this case, you should **remove both quantities from the sequences** and **continue** with the **next ones** towardsthe **next peak**.
* If the **sum** is less than the corresponding **Mountain Peak’s Difficulty level** from the table below, the **peak remains unconquered**. You should **remove both quantities from the sequences.** Alex will have to sleep in his tent. On the next day, he will try **the same peak once again**.

|  |  |
| --- | --- |
| **Mountain Peaks** | **Difficulty level** |
| Vihren | 80 |
| Kutelo | 90 |
| Banski Suhodol | 100 |
| Polezhan | 60 |
| Kamenitza | 70 |

Alex will try to conquer as many peaks as he can in seven days. If he manages to climb **all the peaks**, the journey ends and the output is printed on the Console.

Finally, **print** on the Console **all the conquered peaks**(in the order of climbing).

### Input

* On the **first line**, you will receive the **food portions**, **separated** by **a comma and a** **single space (', ')**.
* On the **second line**, you will receive the **stamina**, **separated** by **a comma and a** **single space (', ')**.

### Output

* On the first line – print whether Alex managed to reach his goal and climb all the peaks in his list:
  + If he managed to conquer all: "**Alex did it! He climbed all top five Pirin peaks in one week -> @FIVEinAWEEK**"
  + If he didn't manage to climb all of the peaks: "**Alex failed! He has to organize his journey better next time -> @PIRINWINS**"
* Then, in either case,you need to print **all the conquered peaks** (in the order of climbing) **if any**:

**"Conquered peaks:**

**{peak1}**

**{peak2}**

**…**

**{peakn}"**

* + If there are **no concurred peaks**, do **NOT** print this message.

### Constraints

* All of the given numbers will be valid integers in the range **[0…100]**.

### Examples

|  |  |  |
| --- | --- | --- |
| ****Input**** | ****Output**** | ****Comment**** |
| **40, 40, 40, 40, 40, 40, 40**  **40, 50, 60, 20, 30, 5, 2** | Alex did it! He climbed all top five Pirin peaks in one week -> @FIVEinAWEEK  Conquered peaks:  Vihren  Kutelo  Banski Suhodol  Polezhan  Kamenitza | We start by taking the last daily portion quantity (**40**) and the first stamina quantity (**40**). Their **sum** is **40 + 40 = 80**. After that, we check if the sum is equal to or greater than the **first peak’s difficulty level**. The **sum** of the food portion and the stamina for the day is equal to **the peak’s difficulty level**, so the **peak is conquered**. We **remove both quantities** from the sequences.  We continue with the next peak, where we have also enough food and stamina to conquer it. After we conquer all five peaks, the program ends, and we print the final post on the Console. After that, we print all conquered peaks, every peak on a new line. |
| **10, 20, 34, 26, 12, 10, 45**  **30, 28, 17, 17, 13, 10, 10** | Alex failed! He has to organize his journey better next time -> @PIRINWINS | We start by taking the last daily portion quantity (**45**) and the first stamina quantity (**30**). Their **sum** is **45 + 30 = 75**. After that, we check if the **sum** is equal to or greater than the **first peak’s difficulty level**. The **sum** is not enough for the peak to be conquered. Alex will have to sleep in the tent. We **remove both quantities** from the sequences.  On the **next day,** he will try the **same peak** once again, so he will need a **sum of 80**, from the food portion and the daily stamina. The sum is 10 + 28 = 40 and it is not enough. Alex will have to sleep in his tent once again. We **remove both quantities** from the sequences.  He will not be able to climb **any of the peaks**, so we should print on the Console a message for an **unsuccessful adventure**. There won’t be **any conquered peaks**, so we will print only the **status message on the Console.** |

# Rally Racing

**Link to Judge:** <https://judge.softuni.org/Contests/Practice/Index/3596#1>

*It's time for one of the biggest races in the world, Paris-Dakar. The organizers of the event want you to do a program that helps them track the cars through the separate stages in the event.*

On the first line, you will be given an **integer N**, which represents the **size of a square matrix**. On the second line you will receive the **racing number** of the tracked race car.

On the **next N lines** you will be given the rows of the matrix (**string sequences**, separated by whitespace), whichwill be representing the **race route**. The tracked race **car** **always** starts with **coordinates [0, 0].** Thеre will be a **tunnel** somewhere across the race route. If the race car runs into the **tunnel** , the **car goes through** it and **exits** at **the other end**. There will be **always two positions** marked with **"T"(tunnel)**. The **finish line** will be marked with **"F".** All other positions will be marked with **"."**.

Keep track of the **kilometers passed**. Every time the race car receives a direction and moves to the **next position** of the race route, it **covers 10 kilometers**. If the car **goes through the tunnel**, it **covers** NOT 10, but **30 kilometers**.

On **each line**, after the matrix is given, you will be receiving **the directions** for the race car.

* left
* right
* up
* down

The race car starts moving across the race route:

* If you receive **"End"** command, before the race car manages to reach the finish line, the car is disqualified and the following output should be printed on the Console: **"Racing car {racing number} DNF."**
* If the race car comes across a position marked with **"."**. The car **passes 10 kilometers** for the current move and waits for the next direction.
* If the race car comes across a position marked with **"T"** this is the **tunnel**. The race car goes through it and **moves to the other position marked with "T"** (the other end of the tunnel).The car **passes 30 kilometers** for the current move. The tunnel stays behind the car, so **the race route is clear**, and **both** the positions marked with **"T"**, **should be marked with "."**.
* If the car **reaches the finish line - "F"** position, the race is over. The tracked race car manages to finish the stage and the following output should be printed on the Console: **"Racing car {racing number} finished the stage!".** Don’t forget that the car has covered another 10 km with the **last** move.

### Input

* On the first line you will receive N - the size of the square matrix (race route)
* On the second line you will receive the racing number of the tracked car
* **On the next N lines,** you will receive the **race route** (**elements** will be **separated by a space**).
* On the following lines, you will receive directions **(left, right, up, down).**
* On the last line, you will receive the command **"End"**.

### Output

* If the racing car has reached the finish line before the **"End"** command is given, print on the Console: **"Racing car {racing number} finished the stage!"**
* If the **"End"** command is given and the racing car **has not reached the finish line yet**, the **race ends** and the following message is printed on the Console: **"Racing car {racing number} DNF."**
* On the second line, print the distance that the tracked race car has covered: **"Distance covered {kilometers passed} km."**
* At the end, mark the **last known position of the race car with** **"C" and** print the **final state of the matrix** (race route). The **row elements** in the output matrix **should NOT be separated** by a whitespace.

### Constraints

* The directions will always lead to coordinates in the matrix.
* There will always be two positions marked with **"T"** , representing the tunnel in the race route.
* The size of the **square** matrix (**race route**) will be between **[4…10].**

### Еxamples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| 5  01  . . . . .  . . . T .  . . . . .  . T . . .  . . F . .  down  right  right  right  down  right  up  down  right  up  End | Racing car 01 finished the stage!  Distance covered 80 km.  .....  .....  .....  .....  ..C.. | The race car starts moving from position[0,0].  The first command is down, so the moving direction is down. The race car is in position[1,0].  Next three commands are right, so the race car comes across the tunnel – "T". The current car position is [1,3]. Swap the "T" with "." The race car goes through the tunnel, so its next position is [3,1]. Swap the "T" with "."  Next direction is down, so the race car position is [4,1].  Next direction is right, so the race car position is [4,2].  The race car reaches the finish line before the "End" command. So it manages to finish the stage. The remaining directions will be ignored and no more moves are going to be executed. |
| 10  45  . . . . . . . . . .  . . T . . . . . . .  . . . . . . . . . .  . . . . . . . . . .  . . . . . . . . . .  . . . . . . . . . .  . . . . . . F . . .  . . . . . . . . . .  . . . . . . . . . .  . . . . . . . T . .  right  down  down  right  up  left  up  up  End | Racing car 45 DNF.  Distance covered 100 km.  ..........  ..........  ..........  ..........  ..........  ..........  ......F...  ......C...  ..........  .......... |  |

# Shopping Cart

**Link to Judge:** <https://judge.softuni.org/Contests/Practice/Index/3515#2>

*Peter has decided to invite some guests. He should go shopping, but he will need help because   
there are too many things he needs to remember. Would you assist him?*

Write a function called **shopping\_cart** that **adds products to a shopping cart** for the following three types of meals: **"Soup"**, **"Pizza"**, and **"Dessert"**. Every meal has a **limit of products** that can be added to it:

* Soup: **3**
* Pizza: **4**
* Dessert: **2**

Once you **reach the limit of a meal**, you should **stop adding products** to that meal.

The function will receive a **different number of arguments**. The arguments will be passed as **tuples with two elements** - the **first** one is the **type of meal**, and **the second** is the **product for the meal**. You need to take **each argument** and make a **dictionary** with the **meal's** **name** as a key and the **products** **as a** **value** of the corresponding key.

There are some additional requirements:

* If you receive the **same product** for the **same meal** **more than once**, you **must not add it** again.
* If you run into the **word** "**Stop**" **(not tuple) as an argument**, you must immediately **stop** **adding products** to the cart - just **sort** and **return** the desired result as described below.

In the end, sort the **meals** by the **number of bought products in descending order**. If **there are meals** with an **equal number of products**, sort them (the meals) by **their name** in **ascending order** (alphabetically). For **each meal** sort **its** **products** in **ascending** **order** (alphabetically).

**Return** an output as described below.

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

### Input

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

### Output

* **Return a string** for **each of the 3 types of a meal** of the **sorted result** in the format**:**
  + **"{meal\_type}:"**

**" - {first\_product\_added}"**

**" - {second\_product\_added}"**

**…**

**" - {Nth\_product\_added}"**

* If there are **no products** given for a meal, return **just** **its name in the format shown above**.
* If there are **NO products** in the cart (at all), **return** the message: "**No products in the cart!"**

### Constrains

* Each **tuple** given will always contain the **type of meal** in the first position and a **product** in the second.
* There will be **no other meals** passed than **"Soup"**, **"Pizza"**, and **"Dessert"**.

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| print(shopping\_cart(  ('Pizza', 'ham'),  ('Soup', 'carrots'),  ('Pizza', 'cheese'),  ('Pizza', 'flour'),  ('Dessert', 'milk'),  ('Pizza', 'mushrooms'),  ('Pizza', 'tomatoes'),  'Stop',  )) | Pizza:  - cheese  - flour  - ham  - mushrooms  Dessert:  - milk  Soup:  - carrots |
| print(shopping\_cart(  ('Pizza', 'ham'),  ('Dessert', 'milk'),  ('Pizza', 'ham'),  'Stop',  )) | Dessert:  - milk  Pizza:  - ham  Soup: |
| print(shopping\_cart(  'Stop',  ('Pizza', 'ham'),  ('Pizza', 'mushrooms'),  )) | No products in the cart! |