# Python Advanced: Exam Preparation

# Pizza Orders

**Link to Judge:** <https://judge.softuni.org/Contests/Practice/Index/2828#0>

On the first line, you will receivea sequence of **pizza** **orders**. **Each order contains a different number of pizzas**, separated by comma and space **", "**. On the **second line**, you will receive a sequence of **employees** **with pizza-making capacities** (how much pizzas an employee could make), separated by comma and space **", "**.

Your task is to check if **all pizza orders are completed**.

To do that, you should take the **first order** and the **last** **employee** and see:

* If the number of pizzas in the order is **less than or equal to** the employee's pizza making capacity, the order is completed. **Remove** **both** the order and the employee.
* If the number of pizzas in the order is **greater than** the employee's pizza making capacity, the **remaining pizzas** from the order are going to be made by **the next employees** until the order is **completed**.
  + If there are **no more employees** to finish the order, consider it **not completed**.
* The restaurant **does not take** orders for more than **10 pizzas** **at once**.
* If an order is **invalid** (less than or equal to 0), you need to **remove it** **before** it is taken by an employee.

You should keep track of the **total pizzas that are being made**.

### Input

* On the **first line** you will be given a sequence of **pizza orders** each represented as a number – **integers** separated by comma and space **", "**
* On the **second line** you will be given a sequence of **employees** with pizza-making capacities – **integers** separated by comma and space **", "**

### Output

* If all orders are **successfully** completed, print:  
  **All orders are successfully completed!**

**Total pizzas made: {total count}**

**Employees: {left employees joined by ", "}**

* Otherwise, if you **ran out** of **employees** and there are still some **orders left** print:  
  **Not all orders are completed.  
  Orders left: {left orders joined by ", "}**

### Constraints

* You will always have **at least one order** and **at least one employee**
* All integers will be in range **[-100, 100]**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 11, 6, 8, 1  3, 1, 9, 10, 5, 9, 1 | All orders are successfully completed!  Total pizzas made: 15  Employees: 3, 1 |
| **Comment** | |
| 1) The restaurant do not take the first order for 11 pizzas.  2) The first employee (1) takes an order for 6 pizzas but could only make 1. 5 pizzas left.  3) The next employee (9) continues the same order for 5 pizzas. The order is completed. Remove both.  4) The next employee (5) takes an order for 8 pizzas but could only make 5. 3 pizzas left.  5) The next employee (10) continues the same order for 3 pizzas. The order is completed. Remove both.  6) The next employee (9) takes an order for 1 pizza. The order is completed. Remove both.  7) All orders are completed. | |
| **Input** | **Output** |
| 10, 9, 8, 7, 5  5, 10, 9, 8, 7 | Not all orders are completed.  Orders left: 2, 5 |
| **Comment** | |
| 1) The last employee (7) takes an order for 10 pizzas but could only make 7. 3 pizzas left.  2) The next employee (8) continues the same order for 3 pizzas. The order is completed. Remove both.  3) The next employee (9) takes an order for 9 pizzas. The order is completed. Remove both.  4) The next employee (10) takes an order for 8 pizzas. The order is completed. Remove both.  5) The next employee (5) takes an order for 7 pizzas but could only make 5. 2 pizzas left.  6) Orders are not completed. | |
| **Input** | **Output** |
| 12, -3, 14, 3, 2, 0  10, 15, 4, 6, 3, 1, 22, 1 | All orders are successfully completed!  Total pizzas made: 5  Employees: 10, 15, 4, 6 |

# Collecting Coins

**Link to Judge:** <https://judge.softuni.org/Contests/Practice/Index/2812#1>

*You are playing a game, and your goal is to collect 100 coins.*

On the first line, you will be given a **number** representing the **size of the field** with a **square** shape. On the following few lines, you will be given the **field** with:

* **One player** - randomly placed in it and marked with the symbol "**P**"
* **Numbers** for coins placed at different positions of the field
* **Walls** marked with "**X**"

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

The player **moves** in the given **direction** with one **step for each command and collects all the coins that come across**. If he goes out of the field, he should **continue to traverse the field** from the **opposite side** in the **same direction**.

**Note:** He can go through the **same path many times**, but he can **collect** **the coins** **just once** (the first time).

There are only **two** **possible outcomes** of the game:

* The player **hits a wall**, **loses the game**, and **his coins are reduced to 50% and rounded down** tothe next-lowest integer number.
* The player collects **at least** **100 coins** and wins the game.

For more clarifications, see the examples below.

### Input

* **A number** representing the size of the field (matrix NxN)
* **A matrix** representing the field (each position **separated by a single space**)
* On each of the following lines, you will get a move command
* The input will always be in the correct format

### Output

* If the player won the game, print: "**You won! You've collected {total\_coins} coins.**"
* If the player loses the game, print: "**Game over! You've collected {total\_coins} coins.**"
* Collected coins have to be **rounded down** tothe next-lowest number.
* The player's path as **cooridnates** **in lists on separate lines:**

"**Your path:**

**[{row\_position1}, {column\_position1}]**

**[{row\_position2}, {column\_position2}]**

**…**

**[{row\_positionN}, {column\_positionN}]**"

### Constrains

* There will be nocase in which less than 100 coins will be in the field
* All given numbers will be valid integers in the range [0, 100]

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  1 X 7 9 11  X 14 46 62 0  15 33 21 95 X  P 14 3 4 18  9 20 33 X 0  left  right  right  up  up  right | You won! You**'**ve collected 125 coins.  Your path:  [3, 0]  [3, 4]  [3, 0]  [3, 1]  [2, 1]  [1, 1]  [1, 2] |
| 8  13 18 9 7 24 41 52 11  54 21 19 X 6 4 75 6  76 5 7 1 76 27 2 37  92 3 25 37 52 X 56 72  15 X 1 45 45 X 7 63  1 63 P 2 X 43 5 1  48 19 35 20 100 27 42 80  73 88 78 33 37 52 X 22  up  down  up  left | Game over! You**'**ve collected 0 coins.  Your path:  [5, 2]  [4, 2]  [5, 2]  [4, 2]  [4, 1] |

# Shopping List

**Link to Judge:** <https://judge.softuni.org/Contests/Practice/Index/3227#2>

Write a function called **shopping\_list** which will receive an integer **number** representing **the budget in leva** and **a different number of keywords.** Each **key** represents the **product** (string),andeach **value** will be a **tuple** with **the product's price** (integer or float number) at the **first** position and **quantity** (integer) at the **second** position.

Your job is to **return which products you bought** with the given budget. You only buy a product if you can buy **all of its quantity**.

You could **only** **start** shopping if you have **at least 100 leva** budget. Otherwise, you should **stop the program** and **return** **"You do not have enough budget."**.

Also, you are **carrying a** **basket** that cannot hold **more than 5 types of products**. You should **stop** **buying** products**:**

* if you **reach** the allowed **amount** **of types of products** (no matter their quantity)
* if you **did not reach** the allowed amount, but you do **not have more products** on the shopping list

You should **always** **buy products** **successively**!

**For each product (all its quantity)** **you succeeded to buy, return a string on a new line in the following format**:

**"You bought {product\_name} for {total\_price} leva."**

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

### Input

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

### Output

* The function should **return strings on separate lines** containing the **bought products** and **their price** in the format described above.
* The **total** **price** should be formatted to the **second decimal point**.

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| print(shopping\_list(100,  microwave=(70, 2),  skirts=(15, 4),  coffee=(1.50, 10),  )) | You bought skirts for 60.00 leva.  You bought coffee for 15.00 leva. |
| print(shopping\_list(20,  jeans=(19.99, 1),  )) | You do not have enough budget. |
| print(shopping\_list(104,  cola=(1.20, 2),  candies=(0.25, 15),  bread=(1.80, 1),  pie=(10.50, 5),  tomatoes=(4.20, 1),  milk=(2.50, 2),  juice=(2, 3),  eggs=(3, 1),  )) | You bought cola for 2.40 leva.  You bought candies for 3.75 leva.  You bought bread for 1.80 leva.  You bought pie for 52.50 leva.  You bought tomatoes for 4.20 leva. |

# Problem 2 - Ball in the Bucket

*You are at the funfair to play different games and test your skills. Now you are playing ball in the bucket game.*

You will be given a **matrix with 6 rows and 6 columns** representing the board. On the board, there will be points (integers) and **buckets** marked with the **letter** "**B**". Rules of the game:

* You can **throw** a ball only **3 times**.
* When you **hit a bucket** (position marked with '**B**'), you score the **sum of the points** in the **same column**.
* You can hit a bucket **only once**. If you hit the **same** **bucket** **again**, you do **not** score any points.
* If you hit **outside a bucket** (hit a number on the board) or **outside the board**, you do **not** score any points.

After the board state, you are going to receive the information for every throw on a **separate line**. The **coordinates’** information of **a hit** will be in the format: **"({row}, {column})"**.

Depending on **how many points you have collected**, you win one of the following:

|  |  |
| --- | --- |
| **Football** | **100 to 199 points** |
| **Teddy Bear** | **200 to 299 points** |
| **Lego Construction Set** | **300 and more points** |

**Your job is to keep track of the scored points and to check if you won a prize.**

For more clarifications, see the examples below.

### Input

* **6 lines** – **matrix** representing the board (each position **separated by a single space**)
* On the next 3 **lines** - the **coordinates** of the throw in the format: **"({row}, {column})"**

### Output

* On the first line:
  + If you won a prize, print:

"**Good job! You scored {points} points, and you've won {prize}."**

* + If you did not win any prize, print the points you need to get at least the first prize:

"**Sorry! You need {points} points more to win a prize.**"

### Constraints

* All of the given **points** will be **integers** in the range **[1, 30]**
* All the given **indexes** will be **integers** in the range **[0, 30]**
* There **always** will be **exactly 6 buckets - 1 on each column**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 10 30 B 4 20 24  7 8 27 23 11 19  13 3 14 B 17 В  12 5 21 22 9 6  B 26 1 28 29 2  25 B 16 15 B 18  (1, 1)  (20, 15)  (4, 0) | Sorry! You need 33 points more to win a prize. |
| B 30 14 23 20 24  29 8 27 18 11 19  13 3 B B 17 6  28 5 21 22 9 B  10 B 26 12 B 2  25 1 16 15 7 4  (0, 0)  (2, 2)  (2, 3) | Good job! You scored 299 points, and you've won Teddy Bear. |