# Exercise: Comprehensions

Problems for exercise and homework for the [Python Advanced Course @SoftUni](https://softuni.bg/courses/python-advanced).

Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/1837>.

## Word Filter

Using a **comprehension**, write a program that receives some **text**, separated by **space**, and take only those words, whose length is **even**. Print each word on a new line.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| kiwi orange banana apple | kiwi  orange  banana |
| pizza cake pasta chips | cake |

## Words Lengths

Using a **list** **comprehension**, write a program that receives some **text**, separated by comma and space **", "**, and prints on the console each **string** with its **length** in the following format:

**"{first\_str} -> {first\_str\_len}, {second\_str} -> {second\_str\_len},…"**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Peter, George, Bill, Lilly, Katy | Peter -> 5, George -> 6, Bill -> 4, Lilly -> 5, Katy -> 4 |
| Some, Random, Text | Some -> 4, Random -> 6, Text -> 4 |

## Capitals

Using a **dictionary comprehension**, write a program which receives **country names** on the **first line,** separated by **comma and space** **", "**, and their corresponding **capital cities** on the second line (again separated by **comma and space** **", "**). **Print** **each country** with their **capital** on a **separate line** in the following format:

**"{country} -> {capital}"**

### Hints

* You can use the **zip()** function to zip the two lists into **tuple pairs.**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Bulgaria, Romania, Germany, England  Sofia, Bucharest, Berlin, London | Bulgaria -> Sofia  Romania -> Bucharest  Germany -> Berlin  England -> London |

## Number Classification

Using a **list comprehension**, write a program that receives **numbers**, separated by comma and space **", "**, and prints all the **positive**, **negative**, **even** and **odd** numbers on separate lines as shown below.

***Note: Zero is counted for a positive number***

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1, -2, 0, 5, 3, 4, -100, -20, 12, 19, -33 | Positive: 1, 0, 5, 3, 4, 12, 19  Negative: -2, -100, -20, -33  Even: -2, 0, 4, -100, -20, 12  Odd: 1, 5, 3, 19, -33 |

## Diagonals

Using a **nested list comprehension,** write a program that reads **NxN** matrix, finds its **diagonals**, prints them and their **sum** as shown below.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  1, 2, 3  4, 5, 6  7, 8, 9 | First diagonal: 1, 5, 9. Sum: 15  Second diagonal: 3, 5, 7. Sum: 15 |

## Matrix of Palindromes

Write a program to generate the following **matrix of palindromes** of **3** letters with **r** rows and **c** columns like the one in the examples below.

* **Rows** define the **first** and the **last** letter: row 0 🡪 'a', row 1 🡪 'b', row 2 🡪 'c', …
* **Columns + rows** define the **middle** letter:
  + column 0, row 0 🡪 'a', column 1, row 0 🡪 'b', column 2, row 0 🡪 'c', …
  + column 0, row 1 🡪 'b', column 1, row 1 🡪 'c', column 2, row 1 🡪 'd', …

### Input

* The numbers r and c stay at the first line at the input.
* r and c are integers.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4 6 | aaa aba aca ada aea afa  bbb bcb bdb beb bfb bgb  ccc cdc cec cfc cgc chc  ddd ded dfd dgd dhd did |
| 3 2 | aaa aba  bbb bcb  ccc cdc |

## Flatten Lists

Write a program to flatten **several lists** of numbers, received in the following format:

* String with numbers or empty strings separated by '|'.
* Values are separated by spaces (' ', one or several)
* Order the output list from the **last** to the **first received**, and their values from **left** to **right** as shown below.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 2 3 |4 5 6 | 7 88 | 7 88 4 5 6 1 2 3 |
| 7 | 4 5|1 0| 2 5 |3 | 3 2 5 1 0 4 5 7 |
| 1| 4 5 6 7 | 8 9 | 8 9 4 5 6 7 1 |

## Heroes Inventory

Using a **comprehension**, write a program that receives a **hero's names and** **items** that need to be added in their inventory (item **name** and item **cost**). **Print** the total **amount of items** with their **total cost** for each hero.

### Input

* On the first line, you will receive the **names of the heroes** separated by comma and space **", "**
* On the next lines until the command **"End"**, you will be given **items** with their **cost** in the following format: **"{name}-{item}-{cost}"**. If an item **already exists** in the hero's inventory - **ignore** it.

### Output

* For each hero, print his **name**, the total **items** and the total **cost** of the items in the format: **"{name} -> Items: {items\_count}, Cost: {items\_cost}"**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Peter, George  Peter-Sword-20  Peter-Shield-10  George-Gem-100  Peter-Sword-15  George-Sword-20  End | Peter -> Items: 2, Cost: 30  George -> Items: 2, Cost: 120 |

## Bunker

Using a **comprehension**, write a program that finds the **number** of given **items** in a bunker and their **average quality**.

On the first line, you will be given **all item categories** present in the bunker, then you will be given a number (**n**). On the next **"n"** lines, you will be given different items in the following format:

**"{category} - {item\_name} - quantity:{item\_quantity};quality:{item\_quality}"**

Store that information, you will need it later. After you receive all the inputs, **print** the **total amount** of items (**sum the quantities**) in the format:

**"Count of items: {count}"**

After that, print the **average quality** of all items in the following format, **formatted to the second digit**:

**"Average quality: {quality sum/categories count}"**

Finally, **print** all **categories** with the **items** on **separate lines** in the format:

**"{category} -> {item1}, {item2}, …"**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| food, water, materials, metal  5  food - pizza - quantity:10;quality:5  metal - copper - quantity:3;quality:10  water - mineral - quantity:5;quality:10  food - burgers - quantity:5;quality:2  materials - wood - quantity:2;quality:5 | Count of items: 25  Average quality: 8.00  food -> pizza, burgers  water -> mineral  materials -> wood  metal -> copper |

## Matrix Modification

Write a program that **reads a matrix** from the console. On the first line, you will get the matrix's **rows**. On next **rows** lines, you will get elements for each **column**, separated with **space**. You will be receiving commands in the following format:

* **Add {row} {col} {value}** – **Increase** the number at the given **coordinates** with the **value.**
* **Subtract {row} {col} {value}** – **Decrease** the number at the given **coordinates** by the **value**.

If **the coordinate is invalid**, you should **print** "**Invalid coordinates**". A coordinate **is valid** if both of the given indexes are in range **[0; len() – 1]**.

When you receive "**END**", you should **print the matrix** and **stop the program**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  1 2 3  4 5 6  7 8 9  Add 0 0 5  Subtract 1 1 2  END | 6 2 3  4 3 6  7 8 9 |
| 4  1 2 3 4  5 6 7 8  8 7 6 5  4 3 2 1  Add 4 4 100  Add 3 3 100  Subtract -1 -1 42  Subtract 0 0 42  END | Invalid coordinates  Invalid coordinates  -41 2 3 4  5 6 7 8  8 7 6 5  4 3 2 101 |