# Programming Fundamentals with Python: Exam Preparation

## Bonus Scoring System

**Submit your solutions in the SoftUni judge system at [https://judge.softuni.org/Contests/Practice/Index/2028#0](https://judge.softuni.org/Contests/Practice/Index/2028" \l "0).**

Create a program that calculates **bonus points** for each **student** enrolled in a course. On the **first** line, you are going to receive **the number of students**. On the **second** line, you will receive **the total number of lectures** in the course. The course has **an additional bonus**, which you will receive **on the third line**. On the following lines, you will be receiving the **count of attendances** **for each student**.

The bonus is calculated with the following **formula**:

**{total bonus} = {student attendances} / {course lectures} \* (5 + {additional bonus})**

Find the student with the **maximum bonus** and print them, along with **his attendance,** in the following format:

**"Max Bonus: {max bonus points}."**

**"The student has attended {student attendances} lectures."**

Round the bonus points at the end to **the nearest larger number**.

### Input / Constraints

* On the **first line,** you are going to receive the **number of the students** – an integer in the range [0…50]
* On the **second line,** you will receive the **number of the lectures** – an integer number in the range [0...50].
* On the **third line**, you will receive **the additional bonus** – an integer number in the range [0….100].
* **On the following lines**, you will be receiving the **attendance of each student**.
* There will **never** be **students with equal bonuses**.

### Output

* Print the **maximum bonus points** and the **attendances** of the given student, **rounded** to the nearest **larger** number, scored by a student in this course in the format described above.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  25  30  12  19  24  16  20 | Max Bonus: 34.  The student has attended 24 lectures. |
| **Comments** | |
| First, we receive the **number of students** enrolled in the course – **5**. The total count of the lectures is **25,** and the additional bonus is **30**. Then we calculate the bonus of the student with 12 attendances, which is **16.8**. We continue calculating **each of the student's bonuses**. The one **with 24 attendances** has the **highest bonus – 33.6 (34 rounded)**, so we print the appropriate message on the console. | |
| 10  30  14  8  23  27  28  15  17  25  26  5  18 | Max Bonus: 18.  The student has attended 28 lectures. |

## Shopping List

**Submit your solutions in the SoftUni judge system at [https://judge.softuni.org/Contests/Practice/Index/2031#1](https://judge.softuni.org/Contests/Practice/Index/2031" \l "1).**

*It's the end of the week, and it is time for you to go shopping, so you need to create a shopping list first.*

### Input

You will receive an **initial list** with groceries separated by an exclamation mark **"!"**.

After that, you will be receiving **4 types** of commands until you receive **"Go Shopping!"**.

* **"Urgent {item}"** - **add** the item at the **start** of the list. If the item **already exists,** skip this command.
* **"Unnecessary {item}"** - **remove** the item with the given name, only **if it exists** in the list. Otherwise, skip this command.
* **"Correct {oldItem} {newItem}"** - if the item with the given **old name** exists, **change** its name with the **new** one. Otherwise, skip this command.
* **"Rearrange {item}"** - if the grocery exists in the list, **remove** it from its **current position** and **add** it at the **end** of the list. Otherwise, skip this command.

### Constraints

* There won't be any duplicate items in the initial list

### Output

* Print the **list** with all the groceries, joined by **", "**:

**"{firstGrocery}, {secondGrocery}, … {nthGrocery}"**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Tomatoes!Potatoes!Bread  Unnecessary Milk  Urgent Tomatoes  Go Shopping! | Tomatoes, Potatoes, Bread |
| **Input** | **Output** |
| Milk!Pepper!Salt!Water!Banana  Urgent Salt  Unnecessary Grapes  Correct Pepper Onion  Rearrange Grapes  Correct Tomatoes Potatoes  Go Shopping! | Milk, Onion, Salt, Water, Banana |

## Man O War

**Submit your solutions in the SoftUni judge system at [https://judge.softuni.org/Contests/Practice/Index/1773#2](https://judge.softuni.org/Contests/Practice/Index/1773" \l "2).**

*The pirates encounter a huge Man-O-War at sea.*

Create a program that **tracks** the **battle** and either chooses a **winner** or prints a **stalemate**. On the **first line,** you will receive the **status** of the **pirate ship**, which is a **string** representing **integer sections** separated by **">"**. On **the second line,** you will receive the **same** type of status, but for the **warship**:

**"{section1}>{section2}>{section3}… {sectionn}"**

On the **third line,** you will receive the **maximum health capacity** a section of the ship can reach.

The following lines represent commands **until** **"Retire"**:

* **"Fire {index} {damage}"** - the pirate ship **attacks** the warship with the **given damage** at that section. Check if the **index is valid** and if not, **skip** the command. If the section **breaks** (health <= 0) the warship **sinks**, print the following and **stop** the program: **"You won! The enemy ship has sunken."**
* **"Defend {startIndex} {endIndex} {damage}"** - the warship **attacks** the pirate ship with the **given damage** at that **range** (**indexes are inclusive)**. Check if both **indexes are valid** and if not, **skip** the command. If the section **breaks** (health <= 0) the pirate ship **sinks**, print the following and **stop** the program:

**"You lost! The pirate ship has sunken."**

* **"Repair {index} {health}"** - the crew **repairs** a section of the **pirate ship** with the **given health**. Check if the **index is valid** and if not, **skip** the command. The health of the section **cannot** exceed the **maximum health capacity**.
* **"Status"** - prints the **count** of all sections of the **pirate ship** that need repair soon, which are all sections that are **lower than 20%** of the **maximum** **health capacity**. Print the following:

**"{count} sections need repair."**

In the end, if a **stalemate** occurs, print the **status** of **both** ships, which is the **sum** of their individual sections, in the following format:

**"Pirate ship status: {pirateShipSum}**

**Warship status: {warshipSum}"**

## Input

* On the **1st line,** you are going to receive the **status** of the **pirate ship** (**integers** separated by **'>'**)
* On the **2nd line,** you are going to receive the **status** of the **warship**
* On the **3rd line,** you will receive the **maximum health** a section of a ship can reach.
* On the following **lines**, until **"Retire"**, you will be receiving commands.

## Output

* Print the output in the **format** **described** **above**.

## Constraints

* The **section numbers** will be integers in the range [**1**….**1000**]
* The **indexes** will be integers [**-200**….**200**]
* The **damage** will be an integer in the range [**1**….**1000**]
* The **health** will be an integer in the range [**1**….**1000**]

## Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | |
| 12>13>11>20>66  12>22>33>44>55>32>18  70  Fire 2 11  Fire 8 100  Defend 3 6 11  Defend 0 3 5  Repair 1 33  Status  Retire | 2 sections need repair.  Pirate ship status: 135  Warship status: 205 | |
| **Comments** | | |
| First, we receive the command "**Fire 2 11**", and damage the warship at section index 2, which is currently 33, and after reduction, the status of the warship is the following:  **12 22 22 44 55 32 18**  The **second** and **third** commands have **invalid indexes**, so we skip them.  The **fourth** command, **"Defend 0 3 5"** damages **4 sections** of the pirate ship with **5,** which results in the following states:  **7 8 6 15 66**  The **fifth** command, **"Repair 1 33"** repairs the pirate ship section and adds **33 health** to the current **8,** which results in **41**  Only **2 sections** of the pirate ship (**7** and **6**) need repair soon.  In the end, there is a **stalemate,** so we print both ship statuses (**sum** of all sections). | | |
| **Input** | | **Output** |
| 2>3>4>5>2  6>7>8>9>10>11  20  Status  Fire 2 3  Defend 0 4 11  Repair 3 18  Retire | 3 sections need repair.  You lost! The pirate ship has sunken. | |