More Exercises: Regular Expressions

Additional exercises for the Python Fundamentals Course @SoftUni.
Submit your solutions in the SoftUni judge system at https://judge.softuni.org/Contests/1744.
Note: All the exercises are excluded from your homework!

1. Race

Write a program that processes information about a race. On the **first line** you will be given **list of participants separated by ", "**. On the next few lines until you receive a line **"end of race"** you will be given some info which will be some **alphanumeric characters**. In between them you could have some **extra characters which you should ignore**. For example: **"G!32e%o7r#32g$235@!2e"**. The **letters are the name** of the person and the **sum of the digits is the distance** he ran. So here we have **George** who ran **29 km**. Store the information about the person only **if the list of racers contains the name of the person**. If you receive the **same person more than once just add the distance to his old distance**. At the end **print the top 3 racers** ordered by **distance in descending** in the format:

"1st place: {first racer}

2nd place: {second racer}

3rd place: {third racer}"

Examples

Input	Output	Comment
George, Peter, Bill, Tom G4e@55or%6g6!68e!!@ R1@!3a$y4456@ B5@i@#123ll G@e54o$r6ge# 7P%et^#e5346r T$o553m&6 end of race	1st place: George 2nd place: Peter 3rd place: Tom	On the 3rd input line we have Ray. He is not in the list, so we do not count his result. The other ones are valid. George has total of 55 km, Peter has 25 and Tom has 19. We do not print Bill because he is on 4th place.

2. SoftUni Bar Income

Let`s take a break and visit the game bar at SoftUni. It is about time for the people behind the bar to go home and you are the person who has to draw the line and calculate the money from the products that were sold throughout the day. Until you receive a line with text **"end of shift"** you will be given lines of input. But before processing that line you should do some validations first.

Each valid order should have a **customer, product, count and a price:**

- Valid customer's name should be **surrounded by '%'** and must **start with a capital letter**, followed by **lower-case letters**
- Valid product **contains any word character (not only letters)** and must be **surrounded by '<' and '>'**
- Valid count is an **integer**, **surrounded by '|'**
- Valid price is any **real number followed by '$'**

The parts of a valid order should appear in the order given: **customer, product, count and a price**.

Between each part there can be other symbols, except (**'|', '$', '%' and '.'**)

For each valid line print on the console: **"{customerName}: {product} - {totalPrice}"**

When you receive **"end of shift"** print the total amount of money for the day **rounded to 2 decimal places** in the following format: **"Total income: {income}"**.

Input / Constraints

- Strings that you have to process until you receive text **"end of shift".**

Output

- Print all of the valid lines in the format **"{customerName}: {product} - {totalPrice}"**
- After receiving **"end of shift"** print the total amount of money for the day rounded to 2 decimal places in the following format: **"Total income: {income}"**
- Allowed working **time** / **memory**: **100ms** / **16MB**.

Examples

Input	Output	Comment
%George%<Croissant>|2|10.3$ %Peter%<Gum>|1|1.3$ %Maria%<Cola>|1|2.4$ end of shift	George: Croissant - 20.60 Peter: Gum - 1.30 Maria: Cola - 2.40 Total income: 24.30	Each line is valid, so we print each order, calculating the total price of the product bought. At the end we print the total income for the day
%InvalidName%<Croissant>|2|10.3$ %Peter%<Gum>1.3$ %Maria%<Cola>|1|2.4 %Valid%<Valid>valid|10|valid20$ end of shift	Valid: Valid - 200.00 Total income: 200.00	On the first line, the customer name isn`t valid, so we skip that line. The second line is missing product count. The third line don`t have a valid price. And only the forth line is valid

3. Star Enigma

The war is in its peak, but you, young Padawan, can turn the tides with your programming skills. You are tasked to create a program to **decrypt** the messages of The Order and prevent the death of hundreds of lives.

You will receive several messages, which are **encrypted** using the legendary star enigma. You should **decrypt the messages**, following these rules:

To properly decrypt a message, you should **count all the letters [s, t, a, r]** – **case insensitive** and **remove** the count from the **current ASCII value of each symbol** of the encrypted message.

After decryption:

Each message should have a **planet name, population, attack type ('A', as attack or 'D', as destruction) and soldier count.**

The planet name **starts after '@'** and contains **only letters from the Latin alphabet**.

The planet population **starts after ':'** and is an **Integer**;

The attack type may be **"A"(attack) or "D"(destruction)** and must be **surrounded by "!"** (exclamation mark).

The **soldier count** starts after **"->"** and should be an Integer.

The order in the message should be: **planet name -> planet population -> attack type -> soldier count.** Each part can be separated from the others by **any character except: '@', '-', '!', ':' and '>'.**

Input / Constraints

- The **first line holds n** – the number of **messages**– integer in range [1…100];
- On the next **n** lines, you will be receiving encrypted messages.

Output

After decrypting all messages, you should print the decrypted information in the following format:

First print the attacked planets, then the destroyed planets.
"Attacked planets: {attackedPlanetsCount}"
"-> {planetName}"
"Destroyed planets: {destroyedPlanetsCount}"
"-> {planetName}"

The planets should be **ordered by name alphabetically.**

Examples

Input	Output	Comments
2 STCDoghudd4=63333D0A53333 EHfsytsnhf?8555&I&2C9555SR	Attacked planets: 1 -> Alderaa Destroyed planets: 1 -> Cantonica	We receive two messages, to decrypt them we calculate the key: First message has decryption key 3. So we substract from each characters code 3. PQ@Alderaa1:30000!A!->20000 The second message has key 5. @Cantonica:3000!D!->4000NM **Both messages are valid** and they contain planet, population, attack type and soldiers count. After decrypting all messages we print each planet according the format given.

Input	Output	Comments
3 tt(''DGsvywgerx>6444444444%H%1B9444 GQhrr|A977777(H(TTTT EHfsytsnhf?8555&I&2C9555SR	Attacked planets: 0 Destroyed planets: 2 -> Cantonica -> Coruscant	We receive three messages. Message one is decrypted with key 4: pp$##@Coruscant:2000000000!D!->5000 Message two is decrypted with key 7: @Jakku:200000!A!MMMM This is **invalid message**, missing soldier count, so we continue. The third message has key 5. @Cantonica:3000!D!->4000NM

"It's a trap!" – Admiral Ackbar

4. Nether Realms

Mighty battle is coming. In the stormy nether realms, demons are fighting against each other for supremacy in a duel from which only one will survive.

Your job, however is not so exciting. You are assigned to **sign in all the participants** in the nether realm's mighty battle's demon book, which of course is **sorted alphabetically**.

A demon's **name contains his health and his damage**.

The **sum of the asci codes** of **all characters** (excluding numbers (0-9), arithmetic symbols (**'+', '-', '*', '/'**) and delimiter dot (**'.'**)) gives a **demon's total health**.

The sum of all numbers in his name forms his base damage. Note that you should consider the plus **'+'** and minus **'-'** signs (e.g. **+10 is 10** and **-10 is -10**). However, there are some symbols (**'*'** and **'/'**) that can further **alter the base damage by multiplying or dividing it by 2** (e.g. in the name "m**15***/c-**5.0**", the base damage is **15 + (-5.0) = 10** and then you need to multiply it by 2 (e.g. 10 * 2 = 20) and then divide it by 2 (e.g. 20 / 2 = 10)).

So, **multiplication and division** are applied **only after all numbers are included** in the calculation and **in the order they appear in the name**.

You will get all demons **on a single line**, separated by commas and zero or more blank spaces. Sort them in **alphabetical order** and print their names **along their health and damage**.

Input

The input will be read from the console. The input consists of a **single line** containing all demon names **separated by commas and zero or more spaces** in the format: **"{demon name}, {demon name}, ... {demon name}"**

Output

Print all demons **sorted by their name in ascending order**, each on a separate line in the format:
- **"{demon name} - {health points} health, {damage points} damage"**

Constraints

- A demon's name will contain **at least one character**
- A demon's name **cannot contain** blank spaces ' ' or commas ','
- A **floating point number will always have digits before and after its decimal separator**
- **Number** in a demon's name **is considere**d everything that is a valid integer or floating point number (with dot '.' used as separator). For example, all these are valid numbers: '4', '+4', '-4', '3.5', '+3.5', '-3.5'

Examples

Input	Output	Comments
M3ph-0.5s-0.5t0.0**	M3ph-0.5s-0.5t0.0** - 524 health, 8.00 damage	M3ph-0.5s-0.5t0.0**: Health = 'M' + 'p' + 'h' + 's' + 't' = 524 health. Damage = (3 + (-0.5) + (-0.5) + 0.0) * 2 * 2 = 8 damage.

Input	Output	Comments
M3ph1st0**, Azazel	Azazel - 615 health, 0.00 damage M3ph1st0** - 524 health, 16.00 damage	Azazel: Health - 'A' + 'z' + 'a' + 'z' + 'e' + 'l' = 615 health. Damage - no digits = 0 damage. M3ph1st0**:

Follow us:

		Health - 'M' + 'p' + 'h' + 's' + 't' = 524 health.
		Damage - (3 + 1 + 0) * 2 * 2 = 16 damage.
Gos/ho	Gos/ho - 512 health, 0.00 damage	

5. HTML Parser

Write a program that extracts a **title** and all the **content** of a HTML file:

- The **title** should be between the **HTML** tags **<title>** and **<\title>**.
- The **content** should be between the **HTML** tags **<body>** and **<\body>**.

There might be different **HTML tags, which you should ignore**:

- Each HTML tag is surrounded by the symbols **"<"** and **">"**. For example: **<body>, <p>, <\html>**
- You also should ignore the HTML tag **"\n"**

Example:

"**<html>\n<head><title>News</title></head>\n<body><p>SoftUniaims to provide free real-world practical\n training for young people who want to turn into\n skillful Python software engineers.</p></body>\n</html>"**

In this example the title is **"News"** and the content is **"SoftUni aims to provide free real-world practical training for young people who want to turn into skillful Python software engineers."**

Input

- The input will be read from the console.
- The input consists of a **single line.**

Output

- The content should be a single string.
- You should extract only the text without the tags.
- When you extract the title and the content, you should print the result in the following format:
  - **"Title: {extracted title}"**
  - **"Content: {extracted content}"**

Input	Output	Comment
<html>\n<head><title>Some title</title></head>\n<body>Here is some content <a href="www.somesite.com">\nclick</body>\n</html>	Title: Some title Content: Here is some content click	We take the title and ignore all the tags to get the content