Exercise: Lists Basics

Problems for exercise and homework for the Python Fundamentals Course @SoftUni. Submit your solutions in the SoftUni judge system at https://judge.softuni.bg/Contests/1725

1. Invert Values

Write a program that receives a **single string** containing **numbers** separated by a **single space**. Print a list containing the **opposite of each number.**

Example

Input	Output
`1 2 -3 -3 5`	`[-1, -2, 3, 3, -5]`
`-4 0 2 57 -101`	`[4, 0, -2, -57, 101]`

2. Multiples List

Write a program that receives **two numbers** (factor and count) and creates a **list** with **length** of the given **count** and contains only **elements** that are **multiples** of the given **factor.**

Example

Input	Output

`2`
`5`
	`[2, 4, 6, 8, 10]`

`1`
`10`
	`[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`

3. Football Cards

Most football fans love it for the goals and excitement. Well, this problem doesn't. You are to handle the referee's little notebook and count the players who were sent off for fouls and misbehavior.

The rules: **Two teams**, named "**A**" and "**B**" have **11 players** each. The players on each team are **numbered** from **1 to 11**. Any player may be **sent off** the field by being given a **red card**. If one of the teams has **less than 7 players** remaining, the game is **stopped** immediately by the referee, and the **team with less than 7 players loses**.

A **card** is a **string** with the **team's letter** (**'A' or 'B'**) followed by a **single dash** and **player's number**. **e.g.** the card **'B-7'** means player **#7** from team **B** received a **card**.

The task: Given a list of cards (could be empty), return the number of **remaining players** on **each team** at the **end of the game** in the format: **`"Team A - {players_count}; Team B - {players_count}".`** If the game was terminated print an additional line: **`"Game was terminated"`**

<u>Note for the random tests</u>: If a player that has already been sent off receives another card - ignore it.

Input

The input (the cards) will come on a **single line** separated by a **single space.**

Output

Print the remaining players as described above and add another line (as shown above) if the game was terminated.

Example

Input	Output
A-1 A-5 A-10 B-2	Team A - 8; Team B - 10
A-1 A-5 A-10 B-2 A-10 A-7 A-3	Team A - 6; Team B - 10 Game was terminated

4. Number Beggars

Your task here is pretty simple: given a **list of numbers** and a **number of beggars**, you are supposed to return a **list with the sum** of what **each beggar** brings home, assuming they all take **regular turns**, from the first to the last.

For example: **[1,2,3,4,5]** for 2 beggars will return a result of **9 and 6**, as the first one takes **[1,3,5]**, the second collects **[2,4]**.

The same list with **3 beggars** would produce a better outcome for the **second** beggar: **5, 7 and 3**, as they will respectively take **[1, 4], [2, 5] and [3]**.

Also note that not all beggars have to take the same amount of "offers", meaning that the length of the list is **not** necessarily a **multiple of n**; length can be even shorter, in which case the last beggars will of course take nothing (0).

Input

You will receive **2 lines** of input: a **single string** containing the numbers separated by a comma and a space **", "**. On the **second line** you will receive the **number of beggars.**

Output

Print a **list** of all the **sums** that each beggar got.

Example

Input	Output
1, 2, 3, 4, 5 2	[9, 6]
3, 4, 5, 1, 29, 4 6	[3, 4, 5, 1, 29, 4]
100, 94, 24, 99 5	[100, 94, 24, 99, 0]

5. Faro Shuffle

A faro shuffle of a deck of playing cards is a shuffle in which the deck is **split exactly in half** and then the cards in the two halves are **perfectly interwoven**, such that the **original bottom card is still on the bottom and the original top card is still on top**.

For example, faro shuffling the list

['ace', 'two', 'three', 'four', 'five', 'six'] once, gives **['ace', 'four', 'two', 'five', 'three', 'six']**

Write a program that receives a **single string** (cards separated by **space**) and on the **second line** receives a **number** of faro **shuffles** that have to be made. Print the **state of the deck after the shuffle.**

***Note*: The length of the deck of cards will always be an even number**

Example

Input	Output

`a b c d e f g h`
`5`
	`['a', 'c', 'e', 'g', 'b', 'd', 'f', 'h']`

`one two three four`
`3`
	`['one', 'three', 'two', 'four']`

6. Survival of the Biggest

Write a program that receives a **list of integer** numbers and a number **n**. The number **n** represents the **amount of numbers to remove** from the list. You should remove the **smallest ones.**

Input

On the first line you will receive a string (numbers separated by a space), on the second line you will receive a number n (count of numbers to remove).

Output

Print all the numbers that are left in the list.

Example

Input	Output

`10 9 8 7 6 5`
`3`
	`[10, 9, 8]`

`1 10 2 9 3 8`
`2`
	`[10, 9, 3, 8]`

7. * Easter Gifts

As a good friend, you decide to buy presents for your friends.

Create a program that helps you plan the gifts for your friends and family. First, you are going to **receive the gifts** you plan on buying on a **single line, separated by space**, in the following **format**:

`"{gift₁} {gift₂} {gift₃}… {giftₙ}"`

Then you will start receiving **commands** until you read the "**No Money**" message. There are **three** possible commands:

- **`"OutOfStock {gift}"`**
  - Find **the gifts** with **this name** in your collection, **if there are any**, and change their values to "**None**".
- **`"Required {gift} {index}"`**
  - **Replace** the value of the **current gift** on the given index **with this gift,** if the **index** is **valid**.
- **`"JustInCase {gift}"`**
  - **Replace** the value of your **last** gift **with this one**.

In the end, print the **gifts** on a **single line**, **except the ones** with value **"None",** separated by a **single space** in the following format:

"{gift$_1$} {gift$_2$} {gift$_3$}… {gift$_n$}"

Input / Constraints

- On the **1st line** you are going to receive the **names of the gifts**, separated by a single space.
- On the next **lines**, until the **"No Money"** command is received, you will be receiving commands.
- The **input** will **always** be **valid**.

Output

- Print the gifts in the **format described above**.

Examples

Input	**Output**
Eggs StuffedAnimal Cozonac Sweets EasterBunny Eggs Clothes OutOfStock Eggs Required Spoon 2 JustInCase ChocolateEgg No Money	StuffedAnimal Spoon Sweets EasterBunny ChocolateEgg

Comments
First, we receive the command "**OutOfStock**" and we need to replace the values of "**Eggs**" with "**None**". After this command the list should look like this: **None StuffedAnimal Cozonac Sweets EasterBunny None Clothes** Afterwards, we receive the "**Required**" command and we need to replace the value on the 2nd index of our list with the value "**Spoon**". The list should looks like this: **None StuffedAnimal Spoon Sweets EasterBunny None Clothes** After, we receive the "**JustInCase**" command, which means we need to replace the last value in our list with "**ChocolateEggs**". The list should look like this: **None StuffedAnimal Spoon Sweets EasterBunny None ChocolateEggs** In the end, we print all of the gifts, except the ones with values "**None**". This is the result list: **StuffedAnimal Spoon Sweets EasterBunny ChocolateEggs**

`Sweets Cozonac Clothes Flowers Wine`
`Clothes Eggs Clothes`
`Required Paper 8`
`OutOfStock Clothes`
`Required Chocolate 2`
`JustInCase Hat`
`OutOfStock Cable`
`No Money`
	`Sweets Cozonac Chocolate Flowers Wine`
	`Eggs Hat`

8. * Seize the Fire

The group of adventurists have gone on their first task. Now they have to walk through fire - literally. They have to use all of the water they have left. Your task is to help them survive.

Create a program that calculates the water that is needed to put out a "fire cell", based on the given information about its "fire level" and how much it gets affected by water.

First, you will be given **the level of fire** inside the cell with the **integer value** of the **cell**, which represents the needed water to put out the fire. They will be given in the following format:

```
"{typeOfFire} = {valueOfCell}#{typeOfFire} = {valueOfCell}#{typeOfFire} = {valueOfCell}……"
```

Afterwards you will receive the **amount of water** you have for putting out the fires. There is a **range** of fire for each of the fire types, and if a cell's value is below or exceeds it, it is invalid and you don't need to put it out.

Type of Fire	**Range**
High	81 - 125
Medium	51 - 80
Low	1 - 50

If a cell is valid, you have to put it out by reducing the water with its value. Putting out fire also takes **effort** and you need to **calculate it**. Its value is equal to **25% of the cell's value**. In the end you will have to print the **total effort**. Keep putting out cells until you run out of water. If you **don't have enough water** to put out a given cell – **skip it** and **try the next one**. In the end, **print the cells you have put out** in the following format:

```
"Cells:
 - {cell1}
 - {cell2}
 - {cell3}
……
 - {cellN}"
"Effort: {effort}"
```

In the end, print the total fire you have put out from all of the cells in the following format: **`"Total Fire: {totalFire}"`**

Input / Constraints

- **On the 1st line** you are going to receive the **fires with their cells** in the format described above **– integer numbers in the range [1…500]**
- **On the 2nd line**, you are going to be given the **water** – **an integer number** in the range **[0….100000]**

Output

- Print the cells, which you have put out in the following format:

```
"Cells:
 - {cell}
 - {cell2}
 - {cell3}
 - {cell5}
……
 - {cellN}"
```

- Print the effort, rounded 2 digits after the decimal separator in the following format:

```
"Effort: {effort}"
```

- Print the total fire put out

```
"Total Fire: {totalFire}"
```

Examples

Input	Output
High = 89#Low = 28#Medium = 77#Low = 23 1250	Cells: - 89 - 28 - 77 - 23 Effort: 54.25 Total Fire: 217

Comments
After reading the output, we start **checking** the **level of the fire** and its validity. The first is valid, so we **subtract the 89** from the amount of **water** – 1250, and the water becomes 1161. We need to calculate the **effort**, which is **25%** of 89. We will **add 89 to the total fire** we have put out. In the end the effort is 54.22 and the total fire: 217

Input	Output

`High = 150#Low = 55#Medium = 86#Low = 40#High = 110#Medium = 77`
`220`
	`Cells:`
	` - 40`
	` - 110`
	`Effort: 37.50`
	`Total Fire: 150`

9. * Hello, France

The budget was enough to get Ali and her friends to Frankfurt and they have some money left, but their final aim is to go to France, which means that they will need more finances. They've decided to make profit by buying items on discount from the Thrift Shop and selling them for a higher price. You must help them.

Create a program that calculates the profit after buying some items and selling them on a higher price. In order to fulfil that, you are going to need certain data - you will receive a **collection of items** and a **budget** in the following format:

`{type->price|type->price|type->price……|type->price}`

`{budget}`

The prices for each of the types **cannot exceed** a certain **price**, which is given bellow:

Type	**Maximum Price**
Clothes	50.00
Shoes	35.00
Accessories	20.50

If a **price** for a certain **item** is **higher than** the **maximum** price, **don't buy it**. Every time you **buy an item**, you have to **reduce the budget** with the value of **its price**. If you don't have enough money for it, you **can't buy it**. Buy **as much** items **as you can**.

You have to **increase** the price of **each of the items you have successfully bought with 40%.** Print the list with **the new prices** and **the profit** you will gain **from selling the items**. They need exactly **150$** for tickets for the train, so if their budget after selling the products is enough – print – **`"Hello, France!"`** and if not – **`"Time to go."`**

Input / Constraints

- **On the 1st line** you are going to receive the **items with their prices** in the format described above **– real numbers in the range [0.00……1000.00]**
- **On the 2nd line**, you are going to be given the **budget** – **a real number** in the range **[0.0….1000.0]**

Output

- Print the list with the bought item's new prices, rounded 2 digits after the decimal separator in the following format:

`"{price1} {price2} {price3} {price5}………{priceN}"`

- Print the profit, **rounded 2 digits** after the decimal separator in the following format:

`"Profit: {profit}"`

- If the money for tickets are enough, print: **`"Hello, France!"`** and if not – **`"Time to go."`**

Examples

Input	Output	Comments

`Clothes->43.30|Shoes->25.25|Clothes->36.52|Clothes->20.90|Accessories->15.60`
`120`
	`60.62 35.35 51.13`
	`Profit: 42.03`
	`Hello, France!`
		We start subtracting the valid prices from the budget:
		120 – 43.40 = **76.7.**
		76.7 – 25.25 = **51.45**
		51.45 – 36.52 = **14.93**
		14.93 is **less** than **20.90** and **15.60**, so we can't buy either of the last two. We must increase **each price** with 40% and the new prices are: **60.62 35.35 51.13.** The profit is **42.03** and their new budget will be – what is left of the budget - **14.93 + {sum of all newPrices}.** It is enough, so we print: **Hello, France!**

`Shoes->41.20|Clothes->20.30|Accessories->40|Shoes->15.60|Shoes->33.30|Clothes->48.60`
`90`
	`28.42 21.84 46.62`
	`Profit: 27.68`
	`Time to go.`	

10. * Bread Factory

As a young baker, you are baking the bread out of the bakery.

You have **initial energy 100 and initial coins 100**. You will be given **a string, representing the working day events**. Each event is separated with **'|'** (vertical bar): **"event1|event2|event3…"**

Each event contains event name or item and a number, separated by dash(**"{event/ingredient}-{number}"**)

- If the event is **"rest":** you gain energy, the number in the second part. But your energy **cannot exceed** your **initial energy (100)**. Print: **"You gained {0} energy."**.

  After that, print your current energy: **"Current energy: {0}."**.
- If the event is **"order"**: You've earned some coins, the number in the second part. Each time you get an order, your **energy decreases with 30 points.**
  - If you have energy to complete the order, print: **"You earned {0} coins."**.

  - If your energy drops below 0, you **skip the order** and **gain 50 energy points**. Print: **"You had to rest!"**.

- In any other case you are having an ingredient, you have to buy. The second part of the event, contains the coins you have to spent and remove from your coins.
  - If you are not bankrupt **(coins <= 0)** you've bought the ingredient successfully, and you should print (**"You bought {ingredient}."**)
  - If you went bankrupt, print **"Closed! Cannot afford {ingredient}."** and your bakery rush is over.

If you managed to handle all events through the day, print on the next three lines:

"Day completed!", **"Coins: {coins}"**, **"Energy: {energy}"**.

Input / Constraints

You will receive a string, representing the working day events, separated with **'|'** (vertical bar): **"event1|event2|event3…"**.

Each event contains event name or ingredient and a number, separated by dash(**"{event/ingredient}-{number}"**)

Output

Print the corresponding messages, described above.

Examples

Input	Output
rest-2|order-10|eggs-100|rest-10	You gained 0 energy. Current energy: 100. You earned 10 coins. You bought eggs. You gained 10 energy. Current energy: 80. Day completed! Coins: 10 Energy: 80
Input	**Output**
order-10|order-10|order-10|flour-100|order-100|oven-100|order-1000	You earned 10 coins. You earned 10 coins. You earned 10 coins. You bought flour.

	`You had to rest!`
	`Closed! Cannot afford oven.`

... 1 package (1/4 ounce) active dry yeast

2-1/4 cups warm water (110° to 115°)

3 tablespoons sugar

1 tablespoon salt

2 tablespoons canola oil

6-1/4 to 6-3/4 cups all-purpose flour ...