# Python Advanced: Exam Preparation

# Christmas Elves

**Link to Judge:** <https://judge.softuni.org/Contests/Practice/Index/3306#0>

*Everything in the Satna Claus' workshop was going well until, on one freezing Sunday, a dangerous storm destroyed almost all toys. Now Santa's elves fear they won't be able to meet their December deadline. It could be a disaster, and some children around the world may not get their Christmas toys. Luckily, you've come up with an idea, and you just need to write a program that manages your plan.*

The Christmas elves have special toy-making skills - еach elf can make a toy from a given number of materials.

First, you will receive a sequence of **integers** representing each **elf's energy**. On the following line, you will be given another sequence of **integers,** each representing a **number of materials** **in a box**.

Your task is to calculate **the total elves' energy used** for making toys and **the total number of successfully made toys**.

You are very clever and have immediately recognized the **pros and cons of the work process** - the **first elf** takes **the last box of materials** and tries to create the toy:

* Usually, the elf **needs energy equal to the number of materials**. If he **has enough** energy, he **makes the toy**. His energy **decreases** by the used energy, and the **toy** **goes straight to Santa's bag**. Then, the elf **eats** a cookie reward which **increases his energy by 1**, and **goes to the end of the line**, preparing for the upcoming boxes.
* Every **third** **time** one of the elves **takes a box**, he tries his best to be creative, and **he will need** **twice as much** **energy** **as usual**. If **he has enough**, he manages to create **2 toys**. Then, his **energy decreases;** he eats a **cookie reward** and **goes to the end of the line,** similar to the first bullet.
* Every **fifth** **time** one of the elves **takes a box**, he is a little clumsy and somehow manages to **break the toy** **when he just made it (if he made it)**. The **toy is** **thrown away,** and the **elf** **doesn't get a cookie reward**. However, his **energy is already spent,** and it needs to be **added** to the total elves' energy.
  + **If an elf creates 2 toys, but he is clumsy, he breaks them.**
* If an elf does **not** have enough energy, he **leaves the box of materials** **to the next elf**. Instead of making the toy, the elf drinks a hot chocolate which **doubles his energy**, and **goes to the end of the line**, preparing for the upcoming boxes.

**Note:** North Pole's social policy is very tolerant of the elves. If the **current elf's energy** is **less than 5 units**, he **does NOT TAKE a box**, but he takes a day off. **Remove the elf** from the collection.

**Stop crafting toys when you are out of materials or elves.**

## Input

* The **first** line of input will represent each elf's energy - **integers**, separated by a **single space**
* On the **second** line, you will be given the **number of materials in each box** - **integers**, separated by a **single space**

## Output

* On the **first line**, print **the number of created toys**: **"Toys: {total\_number\_of\_toys}"**
* On the **second line**, print **the total used energy**: **"Energy: {total\_used\_energy}"**
* On the **next two lines** print the **elves** and **boxes** that are **left**, **if there are any**, **otherwise skip the line:**
  + "**Elves left: {elf1}, {elf2}, … {elfN}**"
  + **"Boxes left: {box1}, {box2}, … {boxN}"**

## Constraints

* All the elves' values will be **integers** in the range **[1, 100]**
* All the boxes' values will be **integers** in the range **[1, 100]**

## Examples

|  |  |  |
| --- | --- | --- |
| ****Input**** | ****Output**** | ****Comment**** |
| **10 16 13 25**  **12 11 8** | **Toys: 3**  **Energy: 31**  **Elves left: 3, 6, 26, 14** | 1) The elf with energy **10** takes the box with **8** materials. He creates **1 gift** and uses **8 units of energy**. He eats a cookie and goes to the end of the line, which now looks like this: **16 13 25 3**.  2) The elf with energy **16** takes the box with 11 materials. He creates **1 gift** and uses **11 units of energy**. Then, he eats a cookie and goes to the end of the line, which now looks like this: **13 25 3 6**.  3) The elf with energy **13** takes the box with **12 materials**. It is the **third** time an elf takes a box. The elf does not have the needed energy: **12 \* 2**, so he drinks a hot chocolate and goes to the end of the line: **25 3 6 26**.  4) The elf with energy **25** **takes the box** with 12 materials. It is the **fourth** time an elf takes a box. He creates **1 gift** and uses **12 units of energy**. He eats a cookie and goes to the end of the line, which now looks like this: **3 6 26 14**.  No boxes are left, so the program ends. Print the desired text. |
| **10 14 22 4 5**  **11 16 17 11 1 8** | **Toys: 7**  **Energy: 75**  **Elves left: 10, 14** |  |
| **5 6 7**  **2 1 5 7 5 3** | **Toys: 3**  **Energy: 20**  **Boxes left: 2, 1** |  |

# Pawn Wars

**Link to Judge:** <https://judge.softuni.org/Contests/Practice/Index/3374#1>

A picture containing text, electronics, keyboard, calculator

Description automatically generated

A chessboard has 8 rows and 8 columns. Rows, also called ranks, are marked from number 1 to 8, and columns are marked from A to H. We have a total of 64 squares. Each square is represented by a combination of letters and a number (a1, b1, c1, etc.). In this problem colors of the board will be ignored.

We will play the game with two pawns, **white (w)** and **black (b)**, where they can:

* **Only** move **forward** in a **straight** **line**:
  + - White (**w**) moves from the 1st rank to the 8th rank direction.
    - Black (**b**) moves from 8th rank to the 1st rank direction.
* Can move only 1 square at a time.
* Can **capture** another pawn **in** **from of them** **only** **diagonally**:

A picture containing text, checker

Description automatically generated

When a pawn reaches the **last rank** (for the **white one - this is the 8th** rank, and **for the black one - this is the 1st** rank), can be **promoted** to a **queen**.

Two pawns (**w** and **b**) will be placed on two random squares of the bord. The **first** **move is always made by the white pawn** (**w**), then black moves (b), then white (w) again, and so on.

Some rules apply when moving paws:

* If the **two pawns** **interact diagonally**, the player, in turn, **must** **capture** the opponent's pawn. When a pawn **captures another pawn**, the **game is ove**r.
* If no capture is possible, the pawns **keep on moving** until **one** of them **reaches the last rank**.

### Input

* On 8 **lines**, you will receive **each row with its 8 columns, each element separated by a single space:**
  + **Empty** **positions** are marked with **"-"**.
  + **White** pawn is marked with **"w"**
  + **Black** pawn is marked with **"b"**

### Output

Print either one of the following:

* **If a pawn captures the other**, print:
  + "**Game over! {White/Black} win, capture on {square}.**"
* **If a pawn reaches the last rank**, print:
  + "**Game over! {White/Black} pawn is promoted to a queen at {square}.**"

## Constraints

* The input will always be valid.
* The matrix will always be 8x8.
* There will be no case where two pawns are placed on the same square.
* There will be no case where two pawns are placed on the same column.
* There will be no case where black/white will be placed on the last rank.

## Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| - - - - - - b -  - - - - - - - -  - - - - - - - -  - - - - - - - -  - - - - - - - -  - w - - - - - -  - - - - - - - -  - - - - - - - - | Game over! White pawn is promoted to a queen at b8. | We start by pushing the white pawn to b4, next, we push the black pawn to g7:  - - - - - - - -  - - - - - - b -  - - - - - - - -  - - - - - - - -  - w - - - - - -  - - - - - - - -  - - - - - - - -  - - - - - - - -  Then white play b5, black play g6:  - - - - - - - -  - - - - - - - -  - - - - - - b -  - w - - - - - -  - - - - - - - -  - - - - - - - -  - - - - - - - -  - - - - - - - -  …  Capturing is not possible here, so after a few more moves, the white pawn is promoted to a queen on b8. |
| - - - - - - - -  - - - - - - - -  - - - - - - - -  - - - - - - - -  - - - - - - - -  b - - - - - - -  - w - - - - - -  - - - - - - - - | Game over! White win, capture on a3. | A white pawn always start first, so it must capture the black one on a3 in the first move:  - - - - - - - -  - - - - - - - -  - - - - - - - -  - - - - - - - -  - - - - - - - -  w - - - - - - -  - - - - - - - -  - - - - - - - - |

# Words Sorting

**Link to Judge:** <https://judge.softuni.org/Contests/Practice/Index/3430#2>

Write a function **words\_sorting** which receives a different number of words.

Create a dictionary, which will have as keys the words that the function received. For each key, create a value that is the sum of all ASCII values of that key.

Then, **sort the dictionary**:

* **By values** in **descending** order, if the **sum of all values** of the dictionary is **odd**
* **By keys** in **ascending** order, if the **sum of all values** of the dictionary is **even**

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

### Input

* There will be **no input**, just any number of words passed to your function

### Output

* The function should **return a string** in the format **"{key} - {value}"** for each key and value on a **separate lines**

### Constraints:

* There will be **no case** with **capital** letters.
* There will be **no case** with a string consisting of **other than letters**.

### Examples

|  |  |  |
| --- | --- | --- |
| **Test Code** | **Output** | **Comment** |
| print(  words\_sorting(  'escape',  'charm',  'mythology'  )) | charm - 523  escape - 625  mythology - 1004 | All of the ascii values of the 'escape' word are:  e = 101, s = 115, c = 99, a = 97, p = 112, e = 101  Their sum is 625.  We add it in the dictionary {'escape': 625}.  The ascii values of the 'charm' are:  c = 99, h = 104, a = 97, r = 117, m = 109  Their sum is 523.  We add it in the dictionary {'escape': 625, 'charm': 625}  The ascii values of the 'mythology' word are:  m = 109, y = 121, t = 116, h = 104, o = 111, l = 108, o = 111, g = 103, y = 121.  Their sum is 1004.  We add it in the dictionary  {'escape': 625, 'charm': 523, 'mythology': 1004}  When we sum 625 + 523 + 1004 = 2152. The result is even, and we sort the dictionary by keys in ascending order. |
| print(  words\_sorting(  'escape',  'charm',  'eye'  )) | escape - 625  charm - 523  eye - 323 |  |
| print(  words\_sorting(  'cacophony',  'accolade'  )) | accolade - 812  cacophony - 964 |  |