# Python Advanced: Exam Preparation

# Bombs

**Link to Judge:** <https://judge.softuni.org/Contests/Practice/Index/2456#0>

*Ezio is still learning how to make bombs. With their help, he will save civilization. We should help Ezio to make his perfect bombs.*

You will be given **two sequences of integers, representing bomb effects** and **bomb casings**.

You need to start from the **first bomb effect** and try to mix it with the **last bomb casing**. If the **sum** of their values is **equal** to **any of the materials in the table below** – **create the bomb** **corresponding** to the **value** and **remove** **both** bomb materials. Otherwise, just **decrease** the value of the **bomb casing** **by 5**. You need to **stop** combining when you have **no more bomb effects** or **bomb casings,** or you successfully filled the bombs pouch.

**Bombs:**

* **Datura Bombs: 40**
* **Cherry Bombs: 60**
* **Smoke Decoy Bombs: 120**

To fill the bomb pouch, Ezio needs **three of each** of the **bomb types**.

### Input

* On the **first line**, you will receive the integers representing the **bomb effects**, **separated** by **", "**.
* On the **second line**, you will receive the integers representing the **bomb casings**, **separated** by ", ".

### Output

* On the **first** line, print:
  + if Ezio **succeeded** to fulfill the bomb pouch: "**Bene! You have successfully filled the bomb pouch!**"
  + if Ezio **didn't succeed** to fulfill the bomb pouch: "You don't have enough materials to fill the bomb pouch."
* On the **second** line, print all bomb effects left:
  + If there are no bomb effects: "**Bomb Effects: empty**"
  + If there are effects: "**Bomb Effects: {bombEffect1}, {bombEffect2},** **(…)**"
* On the **third** line, print all bomb casings left:
  + If there are no bomb casings: "**Bomb Casings: empty**"
  + If there are casings: "**Bomb Casings: {bombCasing1}, {bombCasing2},** **(…)"**
* Then**,** you need to print **all** bombs and the **count you have of them**, ordered **alphabetically**:
  + **"Cherry Bombs: {count}"**
  + **"Datura Bombs: {count}"**
  + **"Smoke Decoy Bombs: {count}"**

### Constraints

* All of the given numbers will be valid integers in the range **[0, 120]**.
* There will be no cases with negative material.

### Examples

|  |  |
| --- | --- |
| ****Input**** | ****Output**** |
| **5, 25, 25, 115**  **5, 15, 25, 35** | **You don't have enough materials to fill the bomb pouch.**  **Bomb Effects: empty**  **Bomb Casings: empty**  **Cherry Bombs: 0**  **Datura Bombs: 3**  **Smoke Decoy Bombs: 1** |
| ****Comment**** | |
| 1) 5 + 35 = 40 -> Datura Bomb. Remove both.  2) 25 + 25 = 50 -> can't create bomb. Bomb casing should be decreased with 5 -> 20  3) 25 + 20 = 45 -> can't create bomb. Bomb casing should be decreased with 5 -> 15  4) 25 + 15 = 40 -> Datura Bomb. Remove both  … | |

|  |  |
| --- | --- |
| ****Input**** | ****Output**** |
| **30, 40, 5, 55, 50, 100, 110, 35, 40, 35, 100, 80**  **20, 25, 20, 5, 20, 20, 70, 5, 35, 0, 10** | **Bene! You have successfully filled the bomb pouch!**  **Bomb Effects: 100, 80**  **Bomb Casings: 20**  **Cherry Bombs: 3**  **Datura Bombs: 4**  **Smoke Decoy Bombs: 3** |
| ****Comment**** | |
| **…**  After creating a bomb with bomb effect 35 and bomb casing 25, have created 3 Cherry bombs, 4 Datura bombs, and 3 Smoke Decoy bombs. From all of the bomb types we have 3 bombs, so the program ends. | |

# Minesweeper Generator

**Link to Judge:** <https://judge.softuni.org/Contests/Practice/Index/2463#1>

*Everybody remembers the old mines game. Now it is time to create your own.*

You will be given an integer **n** for the **size** of the mines field with **square** shape and another one for the number of bombs that you have to place in the field. On the next **n** lines, you will receive the **position for each bomb**. Your task is to **create the game field** placing the **bombs** at the correct positions and mark them with "**\***", and **calculate the numbers** in each cell of the field. Each **cell** represents a **number** of all **bombs** directly near it (**up, down, left, right and the 4 diagonals**).

Table

Description automatically generated

### Input

* On the first line, you are given the integer **n** – the size of the **square** matrix.
* On the second line – the **number** of the **bombs**.
* The **next n lines** holds the position of each **bomb**.

### Output

* Print the matrix you've created.

### Constraints

* The size of the **square** matrix will be between **[2…15].**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4  4  (0, 3)  (1, 1)  (2, 2)  (3, 0) | 1 1 2 \*  1 \* 3 2  2 3 \* 1  \* 2 1 1 |
| 5  3  (1, 1)  (2, 4)  (4, 1) | 1 1 1 0 0  1 \* 1 1 1  1 1 1 1 \*  1 1 1 1 1  1 \* 1 0 0 |

# Flights

**Link to Judge:** <https://judge.softuni.org/Contests/Practice/Index/2828#2>

Create a function named **flights** that receives a different number of arguments representing the information **about the flights for a day**:

* the **destination** of **each flight**
* the count of **passengers** that are boarding the plane
* a string **"Finish"**

You need to take **each argument** and make a **dictionary** with the plane’s **destination as a** **key** and the **passengers** **as a** **value** of the corresponding key.

If there are **more than one** flight to the **same destination**, you should count **all the passengers** that flew to the destination.

**You should modify the dictionary until the current argument is equal to "Finish".**

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

### Input

* There will be **no input**, just parameters passed to your function

### Output

* Thefunction should **return the final dictionary**

### Constrains

* All numbers will be valid integers in the range **[0, 300]**
* There will be no flight without given number of passengers

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| print(flights('Vienna', 256, 'Vienna', 26, 'Morocco', 98, 'Paris', 115, 'Finish', 'Paris', 15)) | {'Vienna': 282, 'Morocco': 98, 'Paris': 115} |
| print(flights('London', 0, 'New York', 9, 'Aberdeen', 215, 'Sydney', 2, 'New York', 300, 'Nice', 0, 'Finish')) | {'London': 0, 'New York': 309, 'Aberdeen': 215, 'Sydney': 2, 'Nice': 0} |
| print(flights('Finish', 'New York', 90, 'Aberdeen', 300, 'Sydney', 0)) | {} |

# 03. List Manipulator

Write a function called **list\_manipulator** which receives a **list of numbers** as **first parameter** and **different amount** of other parameters. The **second** parameter might be **"add"** or **"remove"**. The **third** parameter might be **"beginning"** or **"end"**. There **might** or **might not** be any **other parameters** (numbers):

* In case of **"add"** and **"beginning"**, **add** the given **numbers** to the **beginning** of the given **list of numbers** and **return the new list**
* In case of **"add"** and **"end"**, **add** the given **numbers** to the **end** of the given **list of numbers** and **return the new list**
* In case of **"remove"** and **"beginning"**
  + If there is **another parameter** (number), **remove** that **amount** of numbers from the **beginning** of the **list of numbers**.
  + If there are **no other parameters**, **remove** only the **first element** of the list.
  + Finaly, **return the new list**
* In case of **"remove"** and **"end"**
  + If there is **another parameter** (number), **remove** that **amount** of numbers from the **end** of the **list of numbers**.
  + Otherwise if there are **no other parameters**, **remove** only the **last element** of the list.
  + Finaly, **return the new list**

For more clarifications, see the examples below.

### Input

* There will be **no input**
* **Parameters** will be passed to your function

### Output

* The function should **return the new list of numbers**

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| print(list\_manipulator([1,2,3], "remove", "end"))  print(list\_manipulator([1,2,3], "remove", "beginning"))  print(list\_manipulator([1,2,3], "add", "beginning", 20))  print(list\_manipulator([1,2,3], "add", "end", 30))  print(list\_manipulator([1,2,3], "remove", "end", 2))  print(list\_manipulator([1,2,3], "remove", "beginning", 2))  print(list\_manipulator([1,2,3], "add", "beginning", 20, 30, 40))  print(list\_manipulator([1,2,3], "add", "end", 30, 40, 50)) | [1, 2]  [2, 3]  [20, 1, 2, 3]  [1, 2, 3, 30]  [1]  [3]  [20, 30, 40, 1, 2, 3]  [1, 2, 3, 30, 40, 50] |