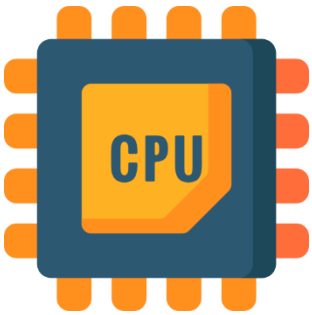
Scheduling



You are hired to create a program that implements SJF (Shortest Job First). It works by letting the shortest jobs to take the CPU, so jobs won't get frozen.

On the **first line** you will be given the **jobs** as **integers** (clock-cycles needed to finish the job) separated by **comma and space ", "**. On the **second line** you will be given the **index** of the job that we are interested in and want to know **how many cycles** will pass until the job is done.

The tasks that need the **least amount of clock-cycles** will be completed **first**.

For the jobs that need the **same amount** of clock-cycles, the order is **FIFO** (First In First Out).

You have to **print** how many **clock-cycles** will pass until the task you are interested in is **completed**. For more clarifications, see the examples below.

Input

- On the first line you will receive **numbers** separated by **", "**
- On the second line you will receive the **index** of the task you are interested in

Output

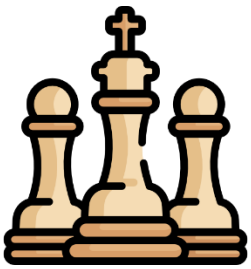
- Single line: the **clock-cycles** that will **pass** until the task you are interested in is **finished**

Examples

Input	Output	Comment
3, 1, 10, 1, 2 0	7	The first task will be 1 at index 1 (1 clock-cycle) Next is 1 at index 3 (total 2 clock-cycles) Next is 2 at index 4 (total 4 clock-cycles) Next, we arrive at 3 on index 0 (total 7 clock-cycles) which is the one we need, and we end the program
4, 10, 10, 6, 2, 99 2	32	2 at index 4 -> total 2 clock-cycles 4 at index 0 -> total 6 clock-cycles 6 at index 3 -> total 12 clock-cycles 10 at index 1 -> total 22 clock-cycles 10 at index 2 -> total 32 clock-cycles

I burned my finger on my computer processor…it MHz!

Checkmate



You will be given a **chess board (8x8)**. On the board there will be **3** types of **symbols**:

- **"."** – empty square
- **"Q"** – a queen
- **"K"** – the king

Your job is to find which **queens** can **capture** the king and **print them**. The moves that the queen can do is to move **diagonally**, **horizontally** and **vertically** (basically all the moves that all the **other figures** can do **except** from the **knight**). Beware that there might be queens that **stand in the way** of other queens and can **stop** them from **capturing** the king. For more clarification see the examples.

Input

- **8 lines** – the state of the board (each square **separated by single space**)

Output

- The **positions** of the **queens** that can **capture** the king as **lists**
- If the king **cannot be captured**, print: **`"The king is safe!"`**
- The **order** of output **does not matter**

Constrains

- There will always be **exactly 8 lines**
- There will always be **exactly one King**
- Only the **3 symbols** described above **will be present** in the input

Examples

Input	Output	Comment
. . . . Q . . . . K . . . Q . . . . . Q . . . . Q . . . Q . . . . Q . . . . Q . . Q . Q	[2, 5] [5, 1] [1, 0]	The queens marked with green can capture the king. The queen marked with blue cannot capture the king, since the queen at [5, 1] stands in the way
. . . . Q . . . . Q . . . Q . . . . . K	The king is safe!	

. . . . . Q . . . . Q		

A happy chess ending is where the King gets mated…

List Pureness

Write function called **`best_list_pureness`** which will receive a **list of numbers** and a number **K**. You have to **rotate** the list **K times (last becomes first)** to find the variation of the list with the **best pureness** (pureness is calculated by **summing** all the **elements** in the list **multiplied** by their **indices**). For example, in the list **`[4, 3, 2, 6]`** with the best pureness is **(3 * 0) + (2 * 1) + (6 * 2) + (4 * 3) = 26**. At the end the function should **return** a **string** containing the **highest pureness** and the **amount of rotations** that were made to find this pureness in the following format: **`"Best pureness {pureness_value} after {count_rotations} rotations"`**. If there is **more than one** highest pureness, take the **first one.**

Note: Submit only the function in the judge system

Input

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

Output

- There is **no expected** output
- The function should **return a string** in the following format: **`"Best pureness {pureness_value} after {count_rotations} rotations"`**

Examples

Test Code	Output	Comment

`test = ([4, 3, 2, 6], 4)`
`result = best_list_pureness(*test)`
`print(result)`
	`Best pureness 26 after 3 rotations`	`Rotation 0 -> Pureness 25`
		`Rotation 1 -> Pureness 16`
		`Rotation 2 -> Pureness 23`
		`Rotation 3 -> Pureness 26`
		`Rotation 4 -> Pureness 25`

`test = ([7, 9, 2, 5, 3, 4], 3)`
`result = best_list_pureness(*test)`
`print(result)`
	`Best pureness 78 after 2 rotations`	`Rotation 0 -> Pureness 60`
		`Rotation 1 -> Pureness 66`
		`Rotation 2 -> Pureness 78`
		`Rotation 3 -> Pureness 78`

`test = ([1, 2, 3, 4, 5], 10)`
`result = best_list_pureness(*test)`
`print(result)`
	`Best pureness 40 after 0 rotations`	

I love the way Earth rotates...
It really makes my day.