JS Syntax Fundamentals – More Exercises

1. *Validity Checker

Write a program that receives a total of 4 parameters in the format **x1, y1, x2, y2.** Check if the distance between each point (**x, y**) and the beginning of the Cartesian coordinate system (**0, 0**) is **valid**. A distance between two points is considered **valid** if it is an **integer value**.

Note: You can use the following formula to help you calculate the distance between the points (**x1, y1**) and (**x2, y2**).

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

The order of comparisons should always be first **{x1, y1}** to **{0, 0}**, then **{x2, y2}** to **{0, 0}** and finally **{x1, y1}** to **{x2, y2}**.

In case a distance is valid, print: `` `{x1, y1} to {x2, y2} is valid` ``

If the distance is invalid, print: `` `{x1, y1} to {x2, y2} is invalid` ``

The **input** consists of two points given as **4 numbers**.

For each comparison print either `` `{x1, y1} to {x2, y2} is valid` `` if the distance is valid, or `` `{x1, y1} to {x2, y2} is invalid` `` if it is invalid.

Examples

Input	Output

`3, 0, 0, 4`	`{3, 0} to {0, 0} is valid`
	`{0, 4} to {0, 0} is valid`
	`{3, 0} to {0, 4} is valid`

`2, 1, 1, 1`	`{2, 1} to {0, 0} is invalid`
	`{1, 1} to {0, 0} is invalid`
	`{2, 1} to {1, 1} is valid`

2. *Words Uppercase

Write a program that **extracts all words** from a passed-in string and converts them to **upper case**. The extracted words in the upper case must be printed on a single line separated by **", "**.

The **input** comes as a single string argument - the text to extract and convert words from.

The **output** should be a single line containing the converted string.

Examples

Input	Output
`'Hi, how are you?'`	`HI, HOW, ARE, YOU`

Input	Output
`'hello'`	`HELLO`

Hints

- You may need to use a Regular Expression or check for all delimiters that can be found in a sentence (ex. ",", " ", "!", "?" and so on).

3. *Calculator

Write a **function** that receives 3 parameters: a **`number`**, an **`operator`** (string), and **`another number`**.

The **operator** can be: **`'+', '-', '/', '*'.`** Print the result of the calculation on the console formatted to the **second decimal** point.

Examples

Input	Output
5, '+', 10	15.00
25.5, '-', 3	22.50

4. *Gladiator Expenses

As a gladiator, Peter has to repair his broken equipment when he loses a fight. His equipment consists of a helmet, sword, shield, and armor. You will receive Peter`s **lost fights count**.

- Every **second** lost game, his helmet is broken.
- Every **third** lost game, his sword is broken.
- When both **his sword and helmet are broken** in the same lost fight, his **shield also breaks**.
- **Every second time**, when his shield brakes, his **armor** also needs to be repaired.

You will receive the price of each item in his equipment. Calculate his expenses for the year for renewing his equipment.

Input / Constraints

You will receive 5 parameters to your function:

- The first parameter - **lost fights count** - is an integer in the range **[0, 1000]**.
- The second parameter - **helmet price** - is the floating-point number in the range **[0, 1000]**.
- The third parameter - **sword price** - is the floating-point number in the range **[0, 1000]**.
- The fourth parameter - **shield price** - is the floating-point number in the range **[0, 1000]**.
- The fifth parameter - **armor price** - is the floating-point number in the range **[0, 1000]**.

Output

- As output you must print Peter`s total expenses for new equipment rounded to the second decimal point: **`"Gladiator expenses: {expenses} aureus"`**
- Allowed working **time** / **memory**: **100ms** / **16MB**.

Examples

Input	Output	Comment
7, 2, 3, 4, 5	Gladiator expenses: 16.00 aureus	Trashed helmet -> 3 times Trashed sword -> 2 times Trashed shield -> 1 time Total: 6 + 6 + 4 = 16.00 aureus;
23, 12.50, 21.50, 40, 200	Gladiator expenses: 608.00 aureus	

5. *Spice Must Flow

Spice is Love, Spice is Life. And most importantly, Spice must flow. It must be extracted from the scorching sands of Arrakis, under the constant threat of giant sandworms. To make the work as efficient as possible, the Duke has tasked you with the creation of management software.

Write a program that calculates the **total amount** of spice that can be extracted from a source. The source has a **starting yield**, which indicates how much spice can be mined on the **first day**. After it has been mined for a day, the **yield drops** by 10, meaning on the second day it'll produce 10 less spice than on the first, on the third day 10 less than on the second, and so on (see examples). A source is considered profitable only while its yield is **at least** 100 – when less than 100 spices are expected in a day, abandon the source.

The mining crew **consumes** 26 spices **every day** at the end of their shift and **an additional** 26 after the mine has been exhausted. Note that the workers **cannot** consume more spice than there is in storage.

When the operation is complete, print on the console on two separate lines how many **days** the mine has operated and the **total** amount of spice extracted.

Input

You will receive a number, representing the **starting yield** of the source.

Output

Print on the console on two separate lines how many **days** the mine has operated and the **total amount** of spice extracted.

Constraints

- The starting yield will be a **number** within range [0…228].

Examples

Input	Output	Explanation
111	2 134	**Day 1** we extract 111 spices and at the end of the shift, the workers consume 26, leaving 85. The yield drops by 10 to 101. **On day 2** we extract 101 spices, the workers consume 26, leaving 75. The total is 160 and the yield has dropped to 91. **Since** the expected yield is less than 100, we abandon the source. The workers take another 26, leaving 134. The mine has operated for 2 days.
450	36 8938	

6. *Login

You will be given a string representing a username. The correct **password** will be that username **reversed**. Until you receive the correct password print on the console: "**`Incorrect password. Try again.`**". When you receive the correct password print: "**`User {username} logged in.`**"

However, on the fourth try if the password is still not correct print: "**`User {username} blocked!`**" and end the program.

The input comes as an **array of strings** - the first string represents username and **each** subsequent string is a password.

Examples

Input	Output

`['Acer','login','go','let me in','recA']`	`Incorrect password. Try again.`
	`Incorrect password. Try again.`
	`Incorrect password. Try again.`
	`User Acer logged in.`

`['momo','omom']`	`User momo logged in.`

`['sunny','rainy','cloudy','sunny','not sunny']`	`Incorrect password. Try again.`
	`Incorrect password. Try again.`
	`Incorrect password. Try again.`
	`User sunny blocked!`

7. * Bitcoin "Mining"

Write a JavaScript program that calculates the **total amount** of **bitcoins** you purchased with the gold you mined during your **shift** at the mine. Your shift consists of a certain number of days where you mine an amount of **gold** in **grams**. Your program will receive an **array with the amount of gold** you mined **each day**, where the **first day** of your **shift** is the **first index of the array**. Also, someone was stealing **every third day** from the start of your shift **30%** from the mined **gold** for **this day**. You need to check, which day you have enough money to buy your **first bitcoin.** For the different exchanges use these **prices**:

1 Bitcoin	11949.16 lv.
1 g of gold	67.51 lv.

Input

You will receive an array of **numbers**, representing your **shift** at the mine.

Output

Print on the **console these lines in the following formats**:

- **First-line** prints the **total amount** of bought **bitcoins**:

  ```
  `Bought bitcoins: {count}`
  ```

- **Second-line** prints **which day** you **bought** your **first bitcoin**:

  ```
  `Day of the first purchased bitcoin: {day}`
  ```

  In case you **did not purchase any bitcoins,** do not print the second line.

- **Third-line** prints the **amount** of **money** that's left after the bitcoin purchases **rounded by the second digit** after the decimal point:

  ```
  `Left money: {money} lv.`
  ```

Constraints

- The **input** array may contain up to **1,000** elements
- The numbers in the array are in the range **[0.01..5,000.00] inclusive**
- Allowed time/memory: 100ms/16MB

Examples

Input	Output

`[100, 200, 300]`	`Bought bitcoins: 2`
	`Day of the first purchased bitcoin: 2`
	`Left money: 10531.78 lv.`

Scroll down to see the explanation for the first example and more examples.

Explanation

Day 1 – you dig up **100 g** of gold then exchange it for **6751.00 lv.**

Day 2 – you dig up **200 g** of gold then exchange it for **13,502.00 lv.** and the total amount of money is **20,253.00 lv.** Then you buy **1 Bitcoin** which leaves you with **8,303.84 lv.** Also, this purchase is the **first day you bought bitcoin**.

Day 3 – you dig up **300 g** of gold but then **30%** of it is stolen and your gold drops to **210 g** which you exchange for **14,177.10 lv.** making your total amount of money **22,480.94 lv.** Then you buy **1 Bitcoin** making the final amount of money that you have **left with 10,531.78 lv.** with **2 bought Bitcoins.**

Input	Output
`[50, 100]`	`Bought bitcoins: 0` `Money left: 10126.50 lv.`

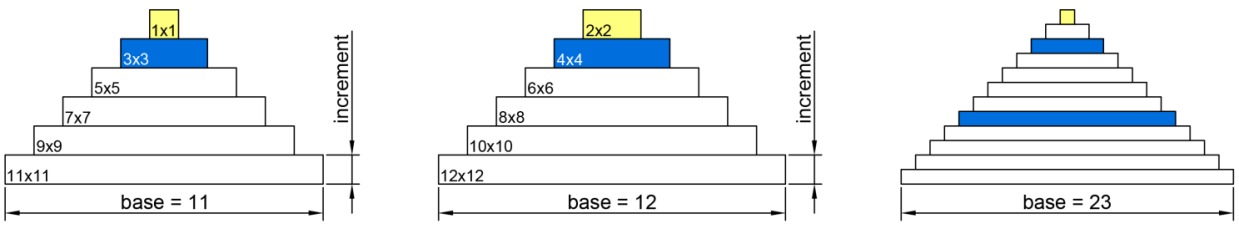
Input	Output

`[3124.15, 504.212, 2511.124]`	`Bought bitcoins: 30`
	`Day of the first purchased bitcoin: 1`
	`Money left: 5144.11 lv.`

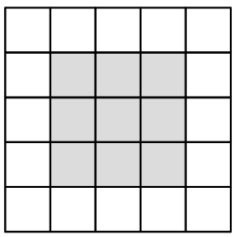
8. * The Pyramid of King Djoser

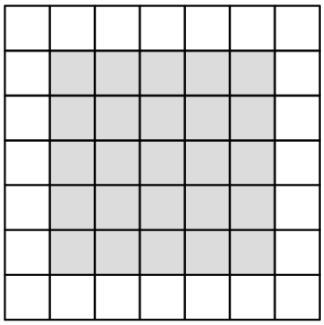
Write a JS program that calculates how much resources will be required for the construction of a pyramid. It is made out of **stone**, **marble**, **lapis lazuli,** and **gold**. Your program will receive an integer that will be the **base** width and length of the pyramid and an **increment** that is the height of each step. The bulk is made out of stone, while the **outer layer** is made out of marble. **Every fifth step's** outer layer is made out of lapis lazuli **instead** of marble. The **final step** is made out of gold.

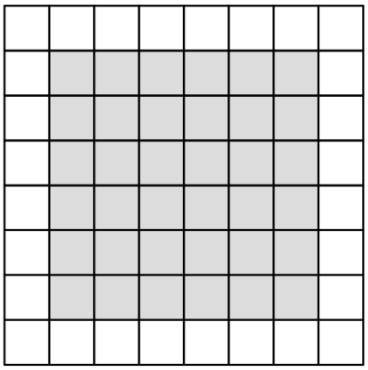
The pyramid is built with **1x1 blocks** with a **height** equal to the given **increment**. The first step of the pyramid has a **width** and **length** equal to the given **base** and every next step is **reduced by 2 blocks** (1 from each side). The height of every step equals the given **increment**. See the drawing for an example. White steps are covered in marble, blue steps are covered in lapis lazuli (**every fifth layer from the bottom**), and yellow steps are made **entirely** out of gold (**top-most step**).



Since the **outer layer** of each step is made of decorative material, to calculate the required stone for one step, reduce the width and length by 2 blocks (one from each side), find its area, and multiply it by the increment. The rest of the step is made out of lapis lazuli for every fifth step from the bottom and marble for all other steps. To find the amount needed, you may, for example, find its perimeter and reduce it by 4 (to compensate for the overlapping corners), and multiply the result by the increment. See the drawing for details (grey is stone, white is decoration).







5x5 step	7x7 step	8x8 step
Stone required – 9 x increment	Stone required – 25 x increment	Stone required – 36 x increment
Marble required– 16 x increment	Marble required – 24 x increment	Marble required – 28 x increment

Note the top-most layer is made entirely out of gold, with a height equal to the given increment. See the examples for complete calculations.

Input

You will receive two **number** parameters **base** and **increment**.

Output

Print on the **console** on separate lines the **total** required **amounts** of each material **rounded up** and the **final height** of the pyramid **rounded down**, as shown in the examples.

Constraints

- The **base** will always be an integer greater than zero
- The **increment** will always be a number greater than zero
- **Number.MAX_SAFE_INTEGER** will **never be exceeded** for any of the calculations

Examples

<table>
<tr><th>Input</th><th>Output</th><th colspan="6">Explanation</th></tr>
<tr><td rowspan="5">11,
1</td><td rowspan="5">Stone required: 165
Marble required: 112
Lapis Lazuli required: 8
Gold required: 1
Final pyramid height: 6</td><td>Step</td><td>Size</td><td>Stone</td><td>Marble</td><td>Lapis</td><td>Gold</td></tr>
<tr><td>1st</td><td>11x11</td><td>81</td><td>40</td><td>-</td><td>-</td></tr>
<tr><td>2nd</td><td>9x9</td><td>49</td><td>32</td><td>-</td><td>-</td></tr>
<tr><td>3rd</td><td>7x7</td><td>25</td><td>24</td><td>-</td><td>-</td></tr>
<tr><td>4th</td><td>5x5</td><td>9</td><td>16</td><td>-</td><td>-</td></tr>
</table>

		5th	3x3	1	-	8	-
		6th	1x1	-	-	-	1
		total	Height=6	165	112	8	1

Input	Output	Explanation
11, 0.75	Stone required: 124 Marble required: 84 Lapis Lazuli required: 6 Gold required: 1 Final pyramid height: 4	Total **stone** is 81*0.75+49*0.75+25*0.75+9*0.75+1*0.75 = 123.75, we round up to 124. Total **marble** is 40*0.75+32*0.75+24*0.75+16*0.75=84. Total **lapis lazuli** is 8*0.75=6. Total **gold** is 1*0.75=0.75, we round up to 1. Total **height** is 4.5 (6 **steps** times 0.75), we round down to 4.

Input	Output
12, 1	Stone required: 220 Marble required: 128 Lapis Lazuli required: 12 Gold required: 4 Final pyramid height: 6

Input	Output
23, 0.5	Stone required: 886 Marble required: 228 Lapis Lazuli required: 36 Gold required: 1 Final pyramid height: 6