# Problem 1 – Bitcoin "Mining"

*So you have heard that a lot of people are using Bitcoins as alternative currency and mining them can make you rich quickly, without doing anything. Because of this you decided to become a miner at the local mine and start digging Bitcoins out of the ground. Well, after a few days you realized that there are no burried Bitcoins in the ground... But luckily, you started digging up gold and decided to invest it in Bitcoins, because they are cool and gold sucks. So you started exchanging gold for BGN and buying Bitcoins with the money, and keeping track of the whole process. How many Bitcoins did you buy and how much money were you left with it at the end?*

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**. 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 **strings** that must be parsed as **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 **witch day** you **bought** your **first bitcoin**:

"**Day of the first purchased bitcoin: {day}**"

In case you **did not** **purchased 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 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** whichleaves 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 are **left with** **10,531.78 lv.** with **2 bought Bitcoins.** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 50, 100 | Bought bitcoins: 0  Money left: 10126.50 lv. | 3124.15, 504.212, 2511.124 | Bought bitcoins: 30  Day of the first purchased bitcoin: 1  Money left: 5144.11 lv. |

# Problem 2 – Air Pollution

*Because of recent events you have become very conscious of the air quality in Sofia. That’s why you decided to keep track of the air pollution levels by making a map. Each block of the map displays a number that represents the current particle pollution in the air at this moment. There are different forces which affect the air quality in various ways. So how clean is the air in Sofia now?*

Write a JavaScript program that tracks the **pollution** **in the air** above Sofia. You will receive **two arguments** – the **first** is the **map of Sofia** represented by a **matrix of numbers** and the second is an **array of strings** representing the **forces** **affecting** the **air quality**. The **map** will **always** be with **5 rows** and **5 columns** in **total** **of** **25 elements - blocks**. Each block’s particle pollution (PM) is **affected** by **3 forces** receivedin the following formats**:**

* **"breeze {index}" –** index is **the row** where **all column’s value drops** by **15** PM
* **"gale {index}" –** index is **the column in all rows** where **value drops** by **20** PM
* **"smog {value}" – all** **blocks** in the map **increase** equally by **the given value’s** PM

The threshold in each block is **50** PM. If it is **below** **that number**, the block’s air is considered **normal** but if it **reaches or goes over it,** that block’s air is considered **polluted**. Also note, that the **polluted particles** in a block **cannot go below** **zero**.

Finally, your program needs to **find** if there are **any polluted blocks** and **print them** in the format given below.

### Input

You will receive **two** **arguments**:

* The **first** argument is an **array with five strings** – **rows** of the matrix with **columns separated by space** that must be parsed as **numbers**, representing the **map of Sofia**.
* The **second** argument is an **array of strings** – each **string** consists of one of the **words (breeze/gale/smog)** and a **number** **separated by space**, representing the **different forces**.

### Output

Print on the **console a single line**:

* If there are **polluted blocks** in the map, **use** their **coordinates** in the following format:
* **"[{rowIndex}-{columnIndex}]"**

Note that you must **start** from the **top left corner** of the map moving to the **bottom right corner** **horizontally**. Then **separate** each **formatted block’s coordinates** with **comma and space** and print them in a single line in the following format:

* **"Polluted areas: {block1}, {block2}, {block3}, …"**
* If there are **no polluted blocks** in the map print:
* **"No polluted areas"**

### Constraints

* The **number** of **rows** and **columns** for the **matrix** will **always** be **5**
* The **number** in each block will be an **integer** in range **[0..1000] inclusive**
* The **number** of **elements** in the **second input argument** will be in range **[0..100] inclusive**
* Given **smog’s** **value** will be an **integer** in range **[0..100] inclusive**
* Given **indexes** will **always** be **valid**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| [  "5 7 72 14 4",  "41 35 37 27 33",  "23 16 27 42 12",  "2 20 28 39 14",  "16 34 31 10 24",  ],  ["breeze 1", "gale 2", "smog 25"] | Polluted areas: [0-2], [1-0], [2-3], [3-3], [4-1] |

### Explanation

Graphic diagram explaining the **first example’s** program flow:

**input**

**breeze 1 (-15)**

**gale 2 (-20)**

**smog 25 (+25)**

**polluted areas**



|  |  |
| --- | --- |
| **Input** | **Output** |
| [  "5 7 3 28 32",  "41 12 49 30 33",  "3 16 20 42 12",  "2 20 10 39 14",  "7 34 4 27 24",  ],  [  "smog 11", "gale 3",  "breeze 1", "smog 2"  ] | No polluted areas |

|  |  |
| --- | --- |
| **Input** | **Output** |
| [  "5 7 2 14 4",  "21 14 2 5 3",  "3 16 7 42 12",  "2 20 8 39 14",  "7 34 1 10 24",  ],  ["breeze 1", "gale 2", "smog 35"] | Polluted areas: [2-1], [2-3], [3-1], [3-3], [4-1], [4-4] |

# Problem 3 – Survey Parser

Write a JavaScript program that **parses** a given **document** that may contain the results of a **rating survey** and outputs a **summary** of the votes. You will receive a **string** that contains XML-formatted data. From this data, you must extract a **valid label** and **average rating** (sum of ratings, divided by their count). Input, containing valid survey data will follow these rules:

* The document may contain **any symbol** **before and after** the survey data
* The survey data **always** begins with <svg> and ends with </svg>:

*Any text* <svg> *Survey data* </svg> *Any text*

* Each valid survey will contain **exactly two** **sections** beginning with <cat> and ending with </cat>
* There may be **whitespace** **between** the sections

<cat> *Survey heading and label* </cat><cat> *Ratings* </cat>

* The contents of the first cat section must begin with <text> and end with </text>; it may contain **any text**, but needs to have a **valid label**, inside brackets []

<text> *Survey heading* [ *Survey Label* ]</text>

* The second cat section contains all of the **ratings** with each vote beginning with <g> and ending with </g>
* A **valid rating** contains a **value** and **count**, with the **value** surrounded by <val> and </val> and the **count** right **after the value**
* There may be any number of **valid** and **invalid ratings** inside the second cat; you should **only** **process** the valid ones, and **ignore** the invalid ratings

<g><val> *Rating value* </val> *Vote count* </g>

* + **Valid** rating: <g><val>1</val>0</g>
  + **Invalid** rating (**ignore** and continue): <g><val>Seafood</val>1</g>
* The **value** must be a number between **1** and **10**
* The **count** must be a number **0** or larger

If the document **does not** contain survey data (no opening and closing svg tags), print on the console "**No survey found**". If there is survey data, but the rest of the rules aren’t followed, print on the console "**Invalid format**".

At the **end** of the program, print on the **console** the **label** of the survey and the **average rating**, **rounded** to two decimal places.

### Input

You will receive a single **string**, containing a document to be parsed.

### Output

Depending on outcome, print on the **console a single line**:

* **No survey found**
* **Invalid format**
* **{label}: {average rating}**

### Constraints

* There will never be more than one valid survey
* Rating **value** will be in range **[1…10]**
* Rating **count** will be in range **[1…1 000 000]**

### Examples

|  |
| --- |
| **Input** |
| <p>Some random text</p><svg><cat><text>How do you rate our food? [Food - General]</text></cat><cat><g><val>1</val>0</g><g><val>2</val>1</g><g><val>3</val>3</g><g><val>4</val>10</g><g><val>5</val>7</g></cat></svg><p>Some more random text</p> |
| **Output** |
| Food - General: 4.1 |
| **Explanation** |
| The survey data is surrounded by <svg> and </svg>, the remaing data is discarded. The first cat contains our label, surrounded by brackets: Food – General.  The second cat contains five ratings, all surrounded with <g> and </g>, and when parsed we get the following:  0 votes with value 1 = 0  1 votes with value 2 = 2  3 votes with value 3 = 9  10 votes with value 4 = 40  7 votes with value 5 = 35  The sum of all ratings is 86, divided by their count 21 gives us the average rating **4.095…**, which we round to the second decimal – **4.1**. |

|  |
| --- |
| **Input** |
| <svg><cat><text>How do you rate the special menu? [Food - Special]</text></cat> <cat><g><val>1</val>5</g><g><val>5</val>13</g><g><val>10</val>22</g></cat></svg> |
| **Output** |
| Food - Special: 7.25 |

|  |
| --- |
| **Input** |
| <p>How do you suggest we improve our service?</p><p>More tacos.</p><p>It's great, don't mess with it!</p><p>I'd like to have the option for delivery</p> |
| **Output** |
| No survey found |

|  |
| --- |
| **Input** |
| <svg><cat><text>Which is your favourite meal from our selection?</text></cat><cat><g><val>Fish</val>15</g><g><val>Prawns</val>31</g><g><val>Crab Langoon</val>12</g><g><val>Calamari</val>17</g></cat></svg> |
| **Output** |
| Invalid format |

# Problem 4 – Game of Epicness

*Welcome to the Game of Epicness where different kingdoms are fighting for the grant price of a bucket of Bitcoins… EPIC! In this amazing game there are many kingdoms with many generals and every general have their own army. To decide who is the winner for this totally amazing price they fight battles between them. But they are not so awesome at math, so they need you to help them record their battle results.*

Write a JavaScript program that **determines** the **winner** from **all battles**. You will receive **two** arguments:

The **first** argument is an **array of kingdoms with generals and their army** in the form of an **object** with format:

{ kingdom: String, general: String, army: Number }

Every **general** has his own **army** that fights for a certain **kingdom**. Note that, every **kingdom’s name** is **unique,** and every general’s **name** is **unique** in **this kingdom**. If the **general** already **exists** **in** this **kingdom** **add** the **army** to his current one. After you go through all the kingdoms with their generals with armies and store the information about them, it’s time to start the battles.

The **second** argument is **matrix of strings** showing which **kingdom’s generals** are **fighting** in this format:

**[**

**[ "{AttackingKingdom} ", "{AttackingGeneral}", "{DefendingKingdom} ", "{DefendingGeneral}" ],**

**…**

**]**

The **first two elements** are the **names** of the **attacking general from** certain **kingdom** and the **second two** are the **names** of the **defending general from** certain **kingdom**. **Compare** the two general’s **armies to determine** who **wins** and who **losses** based on who have the **larger army wins.** The **winner’s army increases** with **10%** and the **loser’s army decreases** with **10%.** Keep in mind to **round** them **down** if there is any excess **army** **after the battle.** If there is a **draw**, **do not do anything**. **Keep track** of the **wins** and **losses** for every general’s battle.

Note that, **generals** from the **same kingdom** **cannot** **attack** **each other**.

After you finish with all battles you need to **find** which **kingdom** **wins** the game. To decide that, **first** **order them** by all their **general’s wins (descending)** then by their **losses (ascending),** and finally by the **kingdom’s name** in **ascending alphabetical** order.

### Input

You will receive **two arguments –** an **array of objects** with properties and a **matrix of strings** as shown above.

### Output

Print on the **console** the winning kingdom and **sort** the generals by their **armies in** **descending** order, **formatted** as seen in the examples.

### Constraints

* The **number** of **elements** in the **first input argument** will be in range **[1..100] inclusive**
* The **number** of **elements** in the **second input argument** will be in range **[0..100] inclusive**
* General’s **army** will be always an **integer** in range **[0..1,000,000] inclusive**
* There **will** be **no invalid** **input**
* There **will** be **no matching number** of **armies** in the **output**

### Examples

|  |
| --- |
| **Input** |
| [ { kingdom: "Maiden Way", general: "Merek", army: 5000 },  { kingdom: "Stonegate", general: "Ulric", army: 4900 },  { kingdom: "Stonegate", general: "Doran", army: 70000 },  { kingdom: "YorkenShire", general: "Quinn", army: 0 },  { kingdom: "YorkenShire", general: "Quinn", army: 2000 },  { kingdom: "Maiden Way", general: "Berinon", army: 100000 } ],  [ ["YorkenShire", "Quinn", "Stonegate", "Ulric"],  ["Stonegate", "Ulric", "Stonegate", "Doran"],  ["Stonegate", "Doran", "Maiden Way", "Merek"],  ["Stonegate", "Ulric", "Maiden Way", "Merek"],  ["Maiden Way", "Berinon", "Stonegate", "Ulric"] ] |
| **Output** |
| Winner: Stonegate  /\general: Doran  ---army: 77000  ---wins: 1  ---losses: 0  /\general: Ulric  ---army: 5336  ---wins: 2  ---losses: 1 |
| **Explanation** |
| After you successfully store the kingdoms information, the first battle’s result is victory for the defender Ulric and a loss for the attacker Quinn. Second battle is ignored because the generals are from the same kingdom. Third battle is a victory for Doran and a loss for Merek. Fourth battle is a win for Ulric and a loss for Merek. Fifth battle is a victory for Berinon and a defeat for Ulric. All winners increase their armies with 10% for each win and all losers decrease their armies with 10% for each loss.  The result from the battles are – Stonegate: 3 wins and 1 loss; Maiden Way: 1 win and 2 losses; YorkenShire: 0 wins and 1 loss. Making Stonegate the winner of the games because they have the most wins from kingdoms. |

|  |
| --- |
| **Input** |
| [ { kingdom: "Stonegate", general: "Ulric", army: 5000 },  { kingdom: "YorkenShire", general: "Quinn", army: 5000 },  { kingdom: "Maiden Way", general: "Berinon", army: 1000 } ],  [ ["YorkenShire", "Quinn", "Stonegate", "Ulric"],  ["Maiden Way", "Berinon", "YorkenShire", "Quinn"] ] |
| **Output** |
| Winner: YorkenShire  /\general: Quinn  ---army: 5500  ---wins: 1  ---losses: 0 |
| **Explanation** |
| The first battle between Quinn and Ulric is a draw because they have even armies because of that it is not recorded and their armies size does not change. The second battle is a win for Quinn and a loss for Berinon making YorkenShire the winner of the game with 1 win and 0 losses. |

|  |
| --- |
| **Input** |
| [ { kingdom: "Maiden Way", general: "Merek", army: 5000 },  { kingdom: "Stonegate", general: "Ulric", army: 4900 },  { kingdom: "Stonegate", general: "Doran", army: 70000 },  { kingdom: "YorkenShire", general: "Quinn", army: 0 },  { kingdom: "YorkenShire", general: "Quinn", army: 2000 } ],  [ ["YorkenShire", "Quinn", "Stonegate", "Doran"],  ["Stonegate", "Ulric", "Maiden Way", "Merek"] ] |
| **Output** |
| Winner: Maiden Way  /\general: Merek  ---army: 5500  ---wins: 1  ---losses: 0 |