# Exam Problems

All problems below are given from the JavaScript Basics exam from **27-July-2014**. You can submit your solutions [here](http://judge.softuni.bg/Contests/19/JavaScript-Basics-Exam-27-July-2014). **You are not obligated** to submit any of them in your homework, but it is highly recommend that you solve some or all of them so you can be well prepared for the upcoming exam. You may read [this post](https://softuni.bg/forum/questions/details/1627) to see how to submit JS code in the Judge system.

## \*Build a Table

Write a JavaScript function that takes as input an array of two numbers (**start** and **end**) and prints at the console a HTML table of 3 columns. The first column should hold a number **num**, changing from **start** to **end**. The second column should hold **num\*num**. The third column should hold "**yes**" if **num** is Fibonacci number or "**no**" otherwise. The table should have header cells titled "**Num**", "**Square**" and "**Fib**". See the below examples.

### Input

The input data comes as **array of two numbers**: **start** and **end**. The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

Print at the console the above described **table** in the same format like the examples below. Don't add additional spaces. **Whitespace** and character **casing** are important, so please use the same as in the below examples.

### Constraints

* The input is passed to the first JavaScript function found in your code as array of 2 elements.
* The numbers **start** and **end** are positive integers in the range [1…1 000 000] and **start** ≤ **end**.
* Allowed working time for your program: 0.2 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  6 | <table>  <tr><th>Num</th><th>Square</th><th>Fib</th></tr>  <tr><td>2</td><td>4</td><td>yes</td></tr>  <tr><td>3</td><td>9</td><td>yes</td></tr>  <tr><td>4</td><td>16</td><td>no</td></tr>  <tr><td>5</td><td>25</td><td>yes</td></tr>  <tr><td>6</td><td>36</td><td>no</td></tr>  </table> |
| **Input** | **Output** |
| 55  56 | <table>  <tr><th>Num</th><th>Square</th><th>Fib</th></tr>  <tr><td>55</td><td>3025</td><td>yes</td></tr>  <tr><td>56</td><td>3136</td><td>no</td></tr>  </table> |

## \*Reveal Triangles

You are given a sequence of **text lines**, holding small Latin letters. Your task is to **reveal all triangles** in the text by changing their letters with '**\***'. Triangles consist of equal letters in the form of triangle:

|  |  |  |  |
| --- | --- | --- | --- |
| a  aaa | x  xxx  xxxxx | p  ppp  ppppp  ppppppp | etc. |

Triangles can span **different sizes**: 2 lines, 3 lines, 4 lines, etc. Triangles can **overlap** (some letters can take part in several triangles).

### Input

The input data comes as **array of strings**, holding the text lines.

### Output

Print at the console the input data after replacing all triangles by '**\***'.

### Constraints

* The input will be passed to the first JavaScript function found in your code as **array of strings**.
* Each input line will hold 1…100 Latin letters.
* The number of input lines will be in the range [1..100].
* Allowed working time: 0.2 seconds. Allowed memory: 16 MB.

### Examples

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| a**b**cde**x**gh  **bbb**d**xxx**h  abc**xxxxx** | a\*cde\*gh  \*\*\*d\*\*\*h  abc\*\*\*\*\* | a**a**  **aaa**  **aaaa**  **aaaaa** | a\*  \*\*\*  \*\*\*\*  \*\*\*\*\* | a**x**  **xxx**  b  b**bb**  **bbbb** | a\*  \*\*\*  b  b\*\*  \*\*\*\* | d**ff**dsgy**e**fg  **ffff**ey**eee**  jb**fff**ays  dag**fff**dsss  dfdf**a**  dad**aaa**dddf  sd**aaaaa**  d**aaaaaaa**sf | d\*\*dsgy\*fg  \*\*\*\*ey\*\*\*  jb\*\*\*ays  dag\*\*\*dsss  dfdf\*  dad\*\*\*dddf  sd\*\*\*\*\*  d\*\*\*\*\*\*\*sf |

Hint: to simplify your work, you can reveal only triangles of size "2 lines", because all bigger triangles consist of several overlapping triangles of size "2 lines".

## \*Extract Hyperlinks

Write a JavaScript function to **extract all hyperlinks** (**<href=…>**) from given text. The text comes as **array of strings**, passed as parameter to your function. Print at the console the **href** values in the text.

The input text is **standard HTML code**. It may hold many tags and can be formatted in many different forms (with or without whitespace). The **<a>** elements may have many attributes, not only **href**. You should extract only the values of the **href** attributes of all **<a>** elements.

### Input

The input data comes as **array of strings**, holding the text lines.

### Output

Print at the console the **href** values in the text, each at a separate line, in the order they come from the input.

### Constraints

* The input will be passed to the first JavaScript function found in your code as **array of strings**.
* The input will be **well formed HTML fragment** (all tags and attributes will be correctly closed).
* Attribute values will never hold tags and hyperlinks, e.g. "**<img alt='<a href="hello">' />**" is invalid.
* Commented links are also extracted.
* The number of input lines will be in the range [1..100].
* Allowed working time: 0.2 seconds. Allowed memory: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| <a href="http://softuni.bg" class="new"></a> | http://softuni.bg |
| <p>This text has no links</p> |  |
| <!DOCTYPE html>  <html>  <head>  <title>Hyperlinks</title>  <link href="theme.css" rel="stylesheet" />  </head>  <body>  <ul><li><a **href="/"** id="home">Home</a></li><li><a  class="selected" **href=/courses**>Courses</a>  </li><li><a **href =**  **'/forum'** >Forum</a></li><li><a class="href"  onclick="go()" **href= "#"**>Forum</a></li>  <li><a id="js" **href =**  **"javascript:alert('hi yo')"** class="new">click</a></li>  <li><a id='nakov' **href =**  **http://www.nakov.com** class='new'>nak</a></li></ul>  <a **href="#empty"**></a>  <a id="href">href='fake'<img src='http://abv.bg/i.gif'  alt='abv'/></a><a **href="#"**>&lt;a href='hello'&gt;</a>  <!-- This code is commented:  <a href="#commented">commentex hyperlink</a> -->  </body> | /  /courses  /forum  #  javascript:alert('hi')  http://www.nakov.com  #empty  #  #commented |

## \*Concerts

You are given a **timetable** for the upcoming rock concerts. It consists of **bands**, **towns**, **dates** and **venues**, separated by '**|**'. Your task is to write a JavaScript function that prints at the console a **JSON string** that holds the **towns**, **venues** for each town and **list of bands** for each venue (see the example below).

### Input

The input data is passed to the first JavaScript function found in your code as **array of strings**. Each input line holds a concert description: **band**, **town**, **date** (in format **dd-MMM-YYYY**) and **venue**, separated by '**|**'. The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

Print at the console a **JSON string** that holds the **towns** (in alphabetical order), the **venues** for each town (in alphabetical order) and a **list of bands** for each venue (in alphabetical order). **Duplicates** should be removed (all strings are **case-sensitive**). Please follow exactly the **JSON format** from the example below.

### Constraints

* The numbers of **input lines** is between 1 and 10 000.
* The values **band**, **town** and **venue** consist of Latin letters, spaces and punctuation marks. Their length is between 1 and 300 characters. Leading and trailing **whitespace** should be removed.
* Allowed working time for your program: 0.2 seconds.
* Allowed memory: 16 MB.

### Examples

|  |
| --- |
| **Input** |
| ZZ Top | London | 2-Aug-2014 | Wembley Stadium  Iron Maiden | London | 28-Jul-2014 | Wembley Stadium  Metallica | Sofia | 11-Aug-2014 | Lokomotiv Stadium  Helloween | Sofia | 1-Nov-2014 | Vassil Levski Stadium  Iron Maiden | Sofia | 20-June-2015 | Vassil Levski Stadium  Helloween | Sofia | 30-July-2015 | Vassil Levski Stadium  Iron Maiden | Sofia | 26-Sep-2014 | Lokomotiv Stadium  Helloween | London | 28-Jul-2014 | Wembley Stadium  Twisted Sister | London | 30-Sep-2014 | Wembley Stadium  Metallica | London | 03-Oct-2014 | Olympic Stadium  Iron Maiden | Sofia | 11-Apr-2016 | Lokomotiv Stadium  Iron Maiden | Buenos Aires | 03-Mar-2014 | River Plate Stadium |
| **Output** |
| {"Buenos Aires":{"River Plate Stadium":["Iron Maiden"]},"London":{"Olympic Stadium":["Metallica"],"Wembley Stadium":["Helloween","Iron Maiden","Twisted Sister","ZZ Top"]},"Sofia":{"Lokomotiv Stadium":["Iron Maiden","Metallica"],"Vassil Levski Stadium":["Helloween","Iron Maiden"]}} |
| **Comments** |
| Concerts in Buenos Aires @ River Plate Stadium: Iron Maiden  Concerts in London @ Olympic Stadium: Metallica  Concerts in London @ Wembley Stadium: Helloween, Iron Maiden, Twisted Sister, ZZ Top  Concerts in Sofia @ Lokomotiv Stadium: Iron Maiden, Metallica  Concerts in Sofia @ Vassil Levski Stadium: Helloween, Iron Maiden |

# Exam Problems

All problems below are given from the JavaScript Basics exam from **28-July-2014**. You can submit your solutions [here](http://judge.softuni.bg/Contests/20/JavaScript-Basics-Exam-28-July-2014). **You are not obligated** to submit any of them in your homework, but it is highly recommend that you solve some or all of them so you can be well prepared for the upcoming exam. You may read [this post](https://softuni.bg/forum/questions/details/1627) to see how to submit JS code in the Judge system.

## – \*Double Rakiya Numbers

A "**double rakiya number**" is an integer that **contains a sequence of 2 digits twice** (without overlapping). For example "2**31**56**31**2" is a "double rakiya number" because it contains "**31**" twice. Other examples of "double rakiya numbers" are: **1212**, **3333**, 2**03**1**03**, 5**21**0**21**7, **21**2121**21**, and **5555**5. Examples of non-"double rakiya numbers" are: 333, 5, 111222, 1234567131, and 12213114.

Write a JavaScript function that takes as input two numbers (**start** and **end**) and prints at the console a HTML list holding all numbers in the range [**start**…**end**], along with a link to view details about all "double-rakiya numbers" in that range. Please use the format from the below examples.

### Input

The input is passed to the first JavaScript function found in your code as **array of two strings**: the numbers **start** and **end**. The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

Print at the console the above described **HTML** **list** in the same format like the examples below. Don't add additional spaces. **Whitespace** and character **casing** are important, so please use the same as in the below examples.

### Constraints

* The numbers **start** and **end** are positive integers in the range [1…1 000 000 000] and **start** ≤ **end**.
* Allowed working time for your program: 0.2 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  8 | <ul>  <li><span class='num'>5</span></li>  <li><span class='num'>6</span></li>  <li><span class='num'>7</span></li>  <li><span class='num'>8</span></li>  </ul> |
| 11210  11215 | <ul>  <li><span class='num'>11210</span></li>  <li><span class='rakiya'>11211</span><a href="view.php?id=11211">View</a></li>  <li><span class='rakiya'>11212</span><a href="view.php?id=11212">View</a></li>  <li><span class='num'>11213</span></li>  <li><span class='num'>11214</span></li>  <li><span class='num'>11215</span></li>  </ul> |
| 55555  55560 | <ul>  <li><span class='rakiya'>55555</span><a href="view.php?id=55555">View</a></li>  <li><span class='rakiya'>55556</span><a href="view.php?id=55556">View</a></li>  <li><span class='rakiya'>55557</span><a href="view.php?id=55557">View</a></li>  <li><span class='rakiya'>55558</span><a href="view.php?id=55558">View</a></li>  <li><span class='rakiya'>55559</span><a href="view.php?id=55559">View</a></li>  <li><span class='num'>55560</span></li>  </ul> |

## \*String Matrix Rotation

You are given a **sequence of text lines**. Assume these text lines form a **matrix of characters** (pad the missing positions with spaces to build a rectangular matrix). Write a program to **rotate the matrix** by 90, 180, 270, 360, … degrees. Print the result at the console as sequence of strings. Examples:

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Rotate(90)** | **Rotate(180)** | **Rotate(270)** |
| hello  softuni  exam |  |  |  |
|  |

### Input

The input is passed to the first JavaScript function found in your code as **array of strings**:

* The first line holds a command in format "**Rotate(X)**" where **X** are the degrees of the requested rotation.
* The next lines to the end contain the **lines of the matrix** for rotation.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

Print at the console the **rotated matrix** as a sequence of text lines.

### Constraints

* The rotation **degrees** is positive integer in the range [0…90000], where **degrees** is **multiple of 90**.
* The number of matrix lines is in the range [1…**1 000**].
* The matrix lines are **strings** of length 1 … 1 000.
* Allowed working time: 0.2 seconds. Allowed memory: 16 MB.

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| Rotate(90)  hello  softuni  exam | esh  xoe  afl  mtl  uo  n  i | Rotate(180)  hello  softuni  exam | maxe  inutfos  olleh | Rotate(270)  hello  softuni  exam | i  n  ou  ltm  lfa  eox  hse |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| Rotate(720)  js  exam | js  exam | Rotate(810)  js  exam | ej  xs  a  m | Rotate(0)  js  exam | js  exam |

## \*Sort Table

You are given a **HTML table** with 3 columns: **product**, **price** and **votes**. Write a JavaScript function to sort the table rows by **price** (as number, increasingly).

### Input

The input is passed to the first JavaScript function found in your code as **array of strings** in the format of the examples below. The HTML table will always have a header row and 3 columns: product, price and votes. **No whitespace** will be found between the tags and between the tags and the tags values.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

Print at the console the **HTML table** sorted by column "**Price**" (as number, increasingly). Please don't change anything in the table, just rearrange its data rows. When several rows hold **equal prices**, use the **product name as** **second sort criteria** (sort by product name alphabetically).

### Constraints

* The **number of rows** in the table is in the range [1…10 000].
* All **prices** are number in the range [0…100 000].
* Allowed working time: 0.2 seconds. Allowed memory: 16 MB.

### Examples

|  |
| --- |
| **Input** |
| <table>  <tr><th>Product</th><th>Price</th><th>Votes</th></tr>  <tr><td>Vodka Finlandia 1 l</td><td>**19.35**</td><td>+12</td></tr>  <tr><td>Ariana Radler 0.5 l</td><td>**1.19**</td><td>+33</td></tr>  <tr><td>Laptop HP 250 G2</td><td>**629**</td><td>+1</td></tr>  <tr><td>Kamenitza Grapefruit 1 l</td><td>**1.85**</td><td>+7</td></tr>  <tr><td>Ariana Grapefruit 1.5 l</td><td>**1.85**</td><td>+7</td></tr>  <tr><td>Coffee Davidoff 250 gr.</td><td>**11.99**</td><td>+11</td></tr>  </table> |
| **Output** |
| <table>  <tr><th>Product</th><th>Price</th><th>Votes</th></tr>  <tr><td>Ariana Radler 0.5 l</td><td>**1.19**</td><td>+33</td></tr>  <tr><td>Ariana Grapefruit 1.5 l</td><td>**1.85**</td><td>+7</td></tr>  <tr><td>Kamenitza Grapefruit 1 l</td><td>**1.85**</td><td>+7</td></tr>  <tr><td>Coffee Davidoff 250 gr.</td><td>**11.99**</td><td>+11</td></tr>  <tr><td>Vodka Finlandia 1 l</td><td>**19.35**</td><td>+12</td></tr>  <tr><td>Laptop HP 250 G2</td><td>**629**</td><td>+1</td></tr>  </table> |

## \*Soccer Results

You are given a sequence of soccer results in format "**homeTeam / awayTeam: homeGoals-awayGoals**". Your task is to write a JavaScript function that prints at the console a **JSON string** that holds the **teams**, and for each team **goals scored**, **goals conceded** and a list of teams **that had a match with this team** in the same format like at the below examples.

## Input

The input data is passed to the first JavaScript function found in your code as **array of strings**. Each input line holds a match description in format "**homeTeam / awayTeam: homeGoals-awayGoals**". The input data will always be valid and in the format described. There is no need to check it explicitly.

Note that any two teams may have played multiple matches.

## Output

Print at the console a **JSON string** that holds the **teams** (in alphabetical order) and for each team **goals scored**, **goals conceded** and a list of teams **that had a match with this team** (in alphabetical order). **Duplicated teams** should be printed once only (all strings are **case-sensitive**). Please follow exactly the **JSON format** from the example below.

## Constraints

* The numbers of **input lines** is between 1 and 10 000.
* The values **homeTeam** and **awayTeam** consist of Latin letters and spaces. Their **length** is between 1 and 50 characters. Leading and trailing **whitespace** should be removed.
* The values **homeGoals** and **awayGoals** are integer numbers in the range [0…99].
* **Whitespace** may be found or missing around the separators "**/**', "**:**" and "**-**".
* Allowed working time: 0.2 seconds. Allowed memory: 16 MB.

## Examples

|  |
| --- |
| **Input** |
| Germany / Argentina: 1-0  Brazil / Netherlands: 0-3  Netherlands / Argentina: 0-0  Brazil / Germany: 1-7  Argentina / Belgium: 1-0  Netherlands / Costa Rica: 0-0  France / Germany: 0-1  Brazil / Colombia: 2-1 |
| **Output** |
| {"Argentina":{"goalsScored":1,"goalsConceded":1,"matchesPlayedWith":["Belgium","Germany","Netherlands"]},"Belgium":{"goalsScored":0,"goalsConceded":1,"matchesPlayedWith":["Argentina"]},"Brazil":{"goalsScored":3,"goalsConceded":11,"matchesPlayedWith":["Colombia","Germany","Netherlands"]},"Colombia":{"goalsScored":1,"goalsConceded":2,"matchesPlayedWith":["Brazil"]},"Costa Rica":{"goalsScored":0,"goalsConceded":0,"matchesPlayedWith":["Netherlands"]},"France":{"goalsScored":0,"goalsConceded":1,"matchesPlayedWith":["Germany"]},"Germany":{"goalsScored":9,"goalsConceded":1,"matchesPlayedWith":["Argentina","Brazil","France"]},"Netherlands":{"goalsScored":3,"goalsConceded":0,"matchesPlayedWith":["Argentina","Brazil","Costa Rica"]}} |

# Exam Problems

All problems below are given from the JavaScript Basics exam from **29-July-2014**. You can submit your solutions [here](http://judge.softuni.bg/Contests/21/JavaScript-Basics-Exam-29-July-2014). **You are not obligated** to submit any of them in your homework, but it is highly recommend that you solve some or all of them so you can be well prepared for the upcoming exam. You may read [this post](https://softuni.bg/forum/questions/details/1627) to see how to submit JS code in the Judge system.

## \*Prices Trends

You are given a **list of prices**. Your task is to **print them in a HTML table**: the first column holds a **price**; the second column holds a **trend**. The trend is either fixed (no change) or moving up or moving down. **Fixed** is the trend of the first price and when the previous price is the same as the current price (after rounding). **Moving up** is when the current price is greater than the previous price (after rounding). **Moving down** is when the current price is less than the previous price (after rounding). All numbers are **rounded to 2 digits after the decimal point**. See the examples below for better understanding.

### Input

The input is passed to the first JavaScript function found in your code as **array of strings** holding the input numbers. The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

Print at the console the prices / trends HTML table following the examples below. The table has a fixed header defining 2 columns: **Price** and **Trend**. The prices column should hold the price, **rounded to 2 decimal places**. The trend is calculated **after rounding** (with 2 decimal places) and can be "**fixed**", "**up**" or "**down**". **Whitespace** and character **casing** are important, so please use the same as in the below examples.

### Constraints

* The **count** of input numbers is in the range [0…1 000].
* All **input numbers** are in the range [0…1 000 000].
* Allowed working time: 0.2 seconds. Allowed memory: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 50  60 | <table>  <tr><th>Price</th><th>Trend</th></tr>  <tr><td>50.00</td><td><img src="fixed.png"/></td></tr>  <tr><td>60.00</td><td><img src="up.png"/></td></tr>  </table> |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 36.333  36.5  37.019  35.4  35  35.001  36.225 | <table>  <tr><th>Price</th><th>Trend</th></tr>  <tr><td>36.33</td><td><img src="fixed.png"/></td></tr>  <tr><td>36.50</td><td><img src="up.png"/></td></tr>  <tr><td>37.02</td><td><img src="up.png"/></td></tr>  <tr><td>35.40</td><td><img src="down.png"/></td></tr>  <tr><td>35.00</td><td><img src="down.png"/></td></tr>  <tr><td>35.00</td><td><img src="fixed.png"/></td></tr>  <tr><td>36.23</td><td><img src="up.png"/></td></tr>  </table> |

## http://tetrismania.net/media/info/tetriminos.png \*Tetris Figures

In the classical Tetris game we have 7 Tetris figures (also called "**tetriminos**"), shown at the figure on the right: **I**, **L**, **J**, **O**, **Z**, **S** and **T**. You are given a rectangular Tetris **game field** consisting of full end empty cells. Your task is to write a JavaScript function to **count the number of each of these 7 tetriminos** (with overlapping, without rotations). For example, on the figure below we have a game field with 2 "**I**", 1 "**L**", 5 "**J**", 3 "**O**", 3 "**Z**", 4 "**S**" and 3 "**T**" figures on it.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| - | - | o | - | - | o | - |  | - | - | **o** | - | - | **o** | - |  | - | - | o | - | - | o | - |  | - | - | **o** | - | - | **o** | - |  | - | - | o | - | - | o | - |
| - | - | o | o | - | o | o | - | - | **o** | o | - | **o** | o | - | - | **o** | o | - | o | o | - | - | **o** | o | - | **o** | o | - | - | **o** | o | - | **o** | o |
| o | o | o | - | o | o | - | o | o | **o** | - | o | **o** | - | o | o | **o** | - | o | o | - | o | **o** | **o** | - | **o** | **o** | - | o | o | **o** | - | o | **o** | - |
| - | o | o | o | o | o | - | - | o | **o** | o | o | **o** | - | - | o | **o** | **o** | o | o | - | - | o | o | o | **o** | o | - | - | **o** | **o** | o | **o** | **o** | - |
| - | - | - | o | o | - | - | - | - | - | o | o | - | - | - | - | - | o | o | - | - | - | - | - | **o** | **o** | - | - | - | - | - | o | o | - | - |
| The game field | | | | | | | 2 pieces "**I**" | | | | | | | 1 piece "**L**" | | | | | | | 3 pieces "**J**" | | | | | | | +2 more pieces "**J**" | | | | | | |
| - | - | o | - | - | o | - |  | - | - | o | - | - | o | - |  | - | - | o | - | - | o | - |  | - | - | o | - | - | o | - |  | - | - | o | - | - | o | - |
| - | - | o | o | - | o | o | - | - | o | o | - | o | o | - | - | **o** | **o** | - | **o** | **o** | - | - | o | o | - | o | o | - | - | o | o | - | o | o |
| o | **o** | **o** | - | **o** | **o** | - | **o** | **o** | **o** | - | o | o | - | o | **o** | **o** | - | **o** | **o** | - | o | o | o | - | **o** | **o** | - | **o** | **o** | **o** | - | o | o | - |
| - | **o** | **o** | **o** | **o** | **o** | - | - | **o** | **o** | **o** | o | o | - | - | o | o | o | **o** | **o** | - | - | o | o | **o** | **o** | o | - | - | **o** | **o** | **o** | **o** | **o** | - |
| - | - | - | **o** | **o** | - | - | - | - | - | **o** | **o** | - | - | - | - | - | **o** | **o** | - | - | - | - | - | o | o | - | - | - | - | - | **o** | **o** | - | - |
| 3 pieces "**O**" | | | | | | | 3 pieces "**Z**" | | | | | | | 3 pieces "**S**" | | | | | | | + 1 more piece "**S**" | | | | | | | 3 pieces "**T**" | | | | | | |

### Input

The input is passed to the first JavaScript function found in your code as **array of strings** holding the game field lines. Each game field line holds only two letters: '**-**' and '**o**' (empty and full cells). All game field lines have the same length. The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

Print at the console the number of **I**, **L**, **J**, **O**, **Z**, **S** and **T** tetriminos found in the game field (with overlapping and without rotations) as **JSON string**, in the same format like in the sample output below.

### Constraints

* The **size of the game field** is in the range [2…100].
* All **input lines** have the same length and consist only of '**-**' and '**o**' (empty and full cells).
* Allowed working time: 0.2 seconds. Allowed memory: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| --o--o-  --oo-oo  ooo-oo-  -ooooo-  ---oo-- | {"I":2,"L":1,"J":5,"O":3,"Z":3,"S":4,"T":3} |

|  |  |
| --- | --- |
| **Input** | **Output** |
| -oo  ooo  ooo | {"I":0,"L":1,"J":2,"O":3,"Z":1,"S":2,"T":1} |

## \*Biggest Table Row

You are given a **HTML table** of 4 columns: **Town**, **Store1**, **Store2** and **Store3**. It consists of sequence of text lines: the "**<table>**" tag, the header row, several data rows, and **"</table>**" tag (see the examples below). The **Store1**, **Store2**, and **Store3** columns hold either numbers or "**-**" (which means "**no data**"). Your task is to write a JavaScript function which parses the table data rows and finds the row with a **maximal sum** of its values.

### Input

The input is passed to the first JavaScript function found in your code as **array of strings** holding the table lines. The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

Print at the console a single line, holding the data row values with a **maximal sum** in format: "**sum = value1 + values2 + …**". Print the values exactly as they were found in the input (no rounding, no reformatting). If all rows contain no data, print "**no data**". If two rows have the **same maximal sum**, print the first of them.

### Constraints

* The **count** of input numbers is in the range [0…1 000].
* The columns **Store1**, **Store2** and **Store3** hold numbers in the range [-100 0000…100 000].
* There is **no whitespace** anywhere in the data rows.
* Allowed working time: 0.2 seconds. Allowed memory: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| <table>  <tr><th>Town</th><th>Store1</th><th>Store2</th><th>Store3</th></tr>  <tr><td>Sofia</td><td>26.2</td><td>8.20</td><td>-</td></tr>  **<tr><td>Varna</td><td>11.2</td><td>18.00</td><td>36.10</td></tr>**  <tr><td>Plovdiv</td><td>17.2</td><td>12.3</td><td>6.4</td></tr>  <tr><td>Bourgas</td><td>-</td><td>24.3</td><td>-</td></tr>  </table> | 65.3 = 11.2 + 18.00 + 36.10 |

|  |  |
| --- | --- |
| **Input** | **Output** |
| <table>  <tr><th>Town</th><th>Store1</th><th>Store2</th><th>Store3</th></tr>  <tr><td>Sofia</td><td>-</td><td>-</td><td>-</td></tr>  </table> | no data |

|  |  |
| --- | --- |
| **Input** | **Output** |
| <table>  <tr><th>Town</th><th>Store1</th><th>Store2</th><th>Store3</th></tr>  <tr><td>Sofia</td><td>12850</td><td>-560</td><td>20833</td></tr>  **<tr><td>Rousse</td><td>-</td><td>50000.0</td><td>-</td></tr>**  <tr><td>Bourgas</td><td>25000</td><td>25000</td><td>-</td></tr>  </table> | 50000 = 50000.0 |

## \*Students, Courses, Grades, Visits

You are given a **list of students score** given as text table with the following columns: **student** name, **course**, **grade**, number of **visits**. A student can have several grades and visits for the same course. Write a JavaScript function to **aggregate the results** and print then as **JSON string** as shown in the examples below.

### Input

The input is passed to the first JavaScript function found in your code as **array of strings** holding the table lines. The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

Print at the console a **JSON string** that holds the **courses** (in alphabetical order), the **average grade** and average visits for each course and a **list of students** for each course (in alphabetical order). **Duplicates** should be removed (all strings are **case-sensitive**). Please follow exactly the **JSON format** from the example below.

The average numbers should be **rounded to 2 digits** after the decimal point and printed **without trailing zeroes**:

* 5 🡪 5; 5.50 🡪 5.5; 5.491 🡪 5.49; 5.495 🡪 5.5; 5.000001 🡪 5; 5.500 🡪 5.5

### Constraints

* The numbers of **input lines** is between 1 and 10 000.
* The names of **students** and **courses** consists of Latin letters and spaces. Their **length** is between 1 and 50 characters. Leading and trailing **whitespace** should be removed.
* The values of **grades** and **visits** will be numbers in the range [0…50].
* Allowed working time: 0.2 seconds. Allowed memory: 16 MB.

### Examples

|  |
| --- |
| **Input** |
| Peter Nikolov | PHP | 5.50 | 8  Maria Ivanova | Java | 5.83 | 7  Ivan Petrov | PHP | 3.00 | 2  Ivan Petrov | C# | 3.00 | 2  Peter Nikolov | C# | 5.50 | 8  Maria Ivanova | C# | 5.83 | 7  Ivan Petrov | C# | 4.12 | 5  Ivan Petrov | PHP | 3.10 | 2  Peter Nikolov | Java | 6.00 | 9 |
| **Output** |
| {"C#":{"avgGrade":4.61,"avgVisits":5.5,"students":["Ivan Petrov","Maria Ivanova","Peter Nikolov"]},"Java":{"avgGrade":5.92,"avgVisits":8,"students":["Maria Ivanova","Peter Nikolov"]},"PHP":{"avgGrade":3.87,"avgVisits":4,"students":["Ivan Petrov","Peter Nikolov"]}} |

# Exam Problems

All problems below are given from the JavaScript Basics exam from **4-Septmeber-2014**. You can submit your solutions [here](http://judge.softuni.bg/Contests/31/JavaScript-Basics-Exam-4-September-2014). **You are not obligated** to submit any of them in your homework, but it is highly recommend that you solve some or all of them so you can be well prepared for the upcoming exam. You may read [this post](https://softuni.bg/forum/questions/details/1627) to see how to submit JS code in the Judge system.

## \*Keep The Change

Don Vlado likes to eat at expensive restaurants. In such restaurants it is accepted that a customer should tip (leave extra change when paying his bill). However, don Vlado happens to be very stingy and wants to spare every penny he can when tipping at his favorite restaurant. Help him by **calculating his exact tip**!

Don Vlado's tip very much depends on his **mood**:

* When *happy*, don Vlado tips for 10% of the bill
* When *married*, don Vlado tips for 0.05% of the bill
* When *drunk*, don Vlado tips for (15% of the bill)**n**, where **n** is the **first digit** of the tip. (e.g. if the bill is 200, **30** is **15% of the bill**. **3** is the **first digit of 30**, so Don Vlado leaves the tip **303 = 30 \* 30 \* 30 = 27000**)
* In every other scenario, don Vlado is simply grumpy and tips for only 5% of the bill

### Input

The input data will be received as an **array**. It contains two arguments – the first one is don Vlado’s **bill**. The second one is **don Vlado’s mood**.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

The output consists of only one line – don Vlado’s tip, **rounded to 2 places after the decimal point**. Use the **toFixed()** function.

### Constraints

* The billwill be a number no greater than 100000.
* Time limit: 0.3 sec. Memory limit: 16 MB.

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 120.44  happy | 12.04 | 1230.83  drunk | 184.62 | 716.00  bored | 35.80 |

## \*The Numbers

*"The numbers, Mason, what do they mean?"*

We’ve just received a ton of unreadable signals from the International Space Station. We've lost all contact with the team up there, and all we got are these messages. Aliens? Might be. Can you please clear up the messages for us, so we can pass them to the decryption team?

Your job is to **clear the text from any unnecessary symbols** (only the numbers are needed) and **convert the remaining number sequences to hex format**. If a hex value has less than 4 characters, you need to **add leading zeros**. Finally, you need toplace a **"0x" prefix before each hex value** and **concatenate them all with dashes** '-'.

For example, we have the following message: "**5tffwj(//\*7837xzc2---34rlxXP%$**". After trimming the unnecessary data (non-numeric characters), we've got the numbers **5**, **7837**, **2** and **34** left. We convert them to hex: **5**, **1E9D**, **2**, **22**; add leading zeros where needed: **0005, 1E9D, 0002, 0022**, place 0x before each oneand concatenate them with dashes: **0x0005-0x1E9D-0x0002-0x0022**.

(Note: hex values *MUST* be uppercase)

### Input

The input data will be received as an **array**. It contains one argument – the initial message you need to transform.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

The output consists of only one line – the transformed message.

### Constraints

* The message will be no longer than 10000 characters.
* The message will consist of ASCII characters only.
* The numbers encoded in the message will be in the range **[0…65 535]**.
* Time limit: 0.3 sec. Memory limit: 16 MB.

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 5tffwj(//\*7837xzc2---34rlxXP%$”. | 0x0005-0x1E9D-0x0002-0x0022 | 482vMWo(\*&^%$213;k!@41341((()&^>><///]42344p;e312 | 0x01E2-0x00D5-0xA17D-0xA568-0x0138 | 20 | 0x0014 |

## \*To The Stars!

The year is 2185 and the SSR Normandy spaceship explores our galaxy. Unfortunately, the ship suffered severe damage in the last battle with Batarian pirates, and her navigation system is broken. Your task is to write a JavaScript program to help the Normandy safely navigate through the stars back home.

The navigation field is a 2D grid. You are given the names of **3 star systems**, along with **their coordinates(sx,sy)** and **the Normandy’s initial coordinates(nx, ny)**. Assume that a **star’s coordinates are in the center of a 2x2 rectangle**. The Normandy **always** **moves in an upwards direction, 1 unit every turn**. Your task is to inform the Normandy of its current location during its movement.



### The Normandy can only be at one location at a time. The possible locations are "<star1 name>", "<star2 name>", "<star3 name>" and "space". In other words, if the Normandy is in the range of Alpha-Centauri, her location is "alpha-centauri". If she's not in the range of any star system, her location is just "space".

Star systems will **NOT** overlap.

*Example*: the Normandy’s initial location is at (8, 1). There, she in outside of any star system, so she is in "space". She starts moving up and her next two locations at (8, 2) and (8, 3) are again in "space". After that, at (8, 4), (8, 5), (8, 6) she is in the range of Alpha-Centauri – therefore, she is in "alpha-centauri". Her final location (8, 7) is outside any star, and her location is "space".

### Input

The input is passed to the first JavaScript function found in your code as **array of several arguments**:

* The first arguments will contain each star system's name and coordinates in the format "<**name**> <**x**> <**y**>", separated by spaces. The **name will be a single word, without spaces**.
* The fourth argument will contain the Normandy’s initial coordinates in the format "<**x**> <**y**>", separated by spaces.
* The fifth, last argument, will contain the number **n** – the number of turns the Normandy will be moving.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

The output consists of **n + 1** lines – the Normandy’s **initial** location, plus the **locations she was in during her movement**, each on a separate line. All locations must be printed **lowercase**.

### Constraints

* The grid dimensions will be no larger than 30x30.
* All star systems will be squares with a fixed size: 2x2.
* The turns will be no more than 20.
* Time limit: 0.3 sec. Memory limit: 16 MB.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| Sirius 3 7  Alpha-Centauri 7 5  Gamma-Cygni 10 10  8 1  6 | space  space  space  alpha-centauri  alpha-centauri  alpha-centauri  space | Terra-Nova 16 2  Perseus 2.6 4.8  Virgo 1.6 7  2 5  4 | perseus  virgo  virgo  virgo  space |

## \*Cloud Manager

Write a program that reads file information from the console and **groups the files according to their extensions** in the format **<file-extension> <[file1, file2, … ]> <total memory in MB>**, where total memory is the sum of the sizes of the respective files. For example, given the files:

* sentinel **.exe** 15MB
* zoomIt **.msi** 3MB
* skype **.exe** 45MB
* trojanStopper **.bat** 23MB
* kindleInstaller **.exe** 120MB
* setup **.msi** 33.4MB
* winBlock **.bat** 1MB

The result should be:

* **.bat** [trojanStopper, winBlock] 24MB
* **.msi** [setup, zoomIt] 36.4MB
* **.exe** [kindleInstaller, sentinel, skype] 180MB

Extension lines should be **sorted by the extension names**. The files themselves should **also** **be** **ordered alphabetically**. Finally, the information is converted to JSON format and printed:

### {".bat":{"files":["trojanStopper","winBlock"],"memory":"24.00"},".exe":{"files":["kindleInstaller","sentinel","skype"],"memory":"180.00"},".msi":{"files":["setup","zoomIt"],"memory":"36.40"}}

### Input

The input is passed to the first JavaScript function found in your code as **array of several strings:** each string will contain information about a file in the format **<name> <extension> <memory>**, separated by spaces.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

Print the file information in **JSON format** as shown in the examples. The memory should be printed with **2 places after the decimal point**.

### Constraints

* File memory will be in the range **[0..1000000]**.
* Time limit: 0.3 sec. Memory limit: 16 MB.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| sentinel .exe 15MB  zoomIt .msi 3MB  skype .exe 45MB  trojanStopper .bat 23MB  kindleInstaller .exe 120MB  setup .msi 33.4MB  winBlock .bat 1MB | {"**.bat**":{"files":["trojanStopper","winBlock"],"memory":"24.00"},"**.exe**":{"files":["kindleInstaller","sentinel","skype"],"memory":"180.00"},"**.msi**":{"files":["setup","zoomIt"],"memory":"36.40"}} |  | eclipse .tar.gz 198.00MB  uTorrent .gyp 33.02MB  nodeJS .gyp 14MB  nakov-naked .jpeg 3MB  gnuGPL .pdf 5.6MB  skype .tar.gz 66MB  selfie .jpeg 7.24MB  myFiles .tar.gz 783MB | {"**.gyp**":{"files":["nodeJS","uTorrent"],"memory":"47.02"},"**.jpeg**":{"files":["nakov-naked","selfie"],"memory":"10.24"},"**.pdf**":{"files":["gnuGPL"],"memory":"5.60"},"**.tar.gz**":{"files":["eclipse","myFiles","skype"],"memory":"1047.00"}} |