# Homework: JavaScript Loops, Arrays, Strings

This document defines the homework assignments from the [“JavaScript Basics“ Course @ Software University](http://softuni.bg/courses/javascript-basics/). Please submit as homework a single zip / rar / 7z archive holding the solutions (source code) of all below described problems.

## Array manipulator

You are given an **array of objects**. Your task is to write a JavaScript code that **filters out** the numbers of that array (removes all non-number objects). Then you should: 1. Find the **Min number**, 2. Find the **Max number**, 3. Find the **most frequent number**. Finally you should **print out** the numbers you have found and the filtered array **sorted** in **descending** order. Example:

|  |  |
| --- | --- |
| **Input** | **Output** |
| ["Pesho", 2, "Gosho", 12, 2, "true", 9, undefined, 0, "Penka", { bunniesCount : 10}, [10, 20, 30, 40]] | Min number: 0  Max number: 12  Most frequent number: 2  [12, 9, 2, 2, 0] |

## Score modification

You are given an array of numbers. Your tasks are to first **filter out** all valid exam scores (between 0 and 400) and afterwards scale them **downwards** by removing **20%** out of each score. Finally you should **print** out the changed scores **sorted** in **ascending** order. Example:

|  |  |
| --- | --- |
| **Input** | **Output** |
| [200, 120, 23, 67, 350, 420, 170, 212, 401, 615, -1] | [ 18.4, 53.6, 96, 136, 160, 169.6, 280 ] |

## Replace <a> Tag

Write a JavaScript function **replaceATag(str)** that replaces in a HTML document given as string **all the tags <a href="…">…</a>** with corresponding **tags [URL=…]…/URL]**.Write JS program **aTagReplacer.js** that invokes your function with the sample input data below and prints the output at the console. Examples:

|  |  |
| --- | --- |
| **Input** | **Output** |
| '<ul>  <li>  <a href=[http://softuni.bg](http://softuni.bg/)>SoftUni</a>  </li>  </ul>' | <ul>  <li>  [URL href=[http://softuni.bg](http://softuni.bg/)]SoftUni[/URL]  </li>  </ul> |

## Course grade scaling

You are given a JSON string containing an array of Students (Objects). Your task is to **scale** their scores upwards by **increasing** them with **10%**. After that you should **add a field** that shows whether the student **has passed or failed** the exam (passed exam means 100 or more points at the exam). Finally you should **filter** out only the students that **have passed** the exam and print them out **sorted** **alphabetically**. Example:

|  |  |
| --- | --- |
| **Input** | **Output** |
| [  {  'name' : 'Пешо',  'score' : 91  },  {  'name' : 'Лилия',  'score' : 290  },  {  'name' : 'Алекс',  'score' : 343  },  {  'name' : 'Габриела',  'score' : 400  },  {  'name' : 'Жичка',  'score' : 70  } ] | [{"name":"Алекс","score":377.3,"hasPassed":true},{"name":"Габриела","score":440,"hasPassed":true},{"name":"Лилия","score":319,"hasPassed":true},{"name":"Пешо","score":100.1,"hasPassed":true}] |

## \* Group People

Write a JavaScript function **groupBy('criteria')** that groups an array of **people** by age, first or last name. Create a **Person** **constructor** to add every person in the person array. The **groupBy('criteria')** function must return an object, with **keys** – the groups (**age**, **firstName** and **lastName**) and **values** – arrays with people in this group. Print on the console every entry of the returned object. Use function **overloading** (i.e. just one function).   
You may need to find how to use constructors. Examples:

|  |
| --- |
| **Predefined array** |
| var people = [  new Person("Scott", "Guthrie", 38),  new Person("Scott", "Johns", 36),  new Person("Scott", "Hanselman", 39),  new Person("Jesse", "Liberty", 57),  new Person("Jon", "Skeet", 38)  ]; |
| **Input** |
| groupBy('firstName'); |
| **Output** |
| Group Scott : [Scott Guthrie(age 38), Scott Johns(age 36), Scott Hanselman(age 39)]  Group Jesse : [Jesse Liberty(age 57)]  Group Jon : [Jon Skeet(age 38)] // key : value |
| **Input** |
| groupBy('age'); |
| **Output** |
| Group 36 : [Scott Hanselman(age 36), Jon Skeet(age 36)]  Group 38 : [Scott Guthrie(age 38)]  Group 40 : [Scott Johns(age 40)]  Group 57 : [Jesse Liberty(age 57)] |
| **Input** |
| groupBy('lastName'); |
| **Output** |
| Group Guthrie : [Scott Guthrie(age 38)]  Group Johns : [Scott Johns(age 40),Jesse Johns(age 57)]  Group Hanselman : [Scott Hanselman(age 36)]  Group Skeet : [Jon Skeet(age 36)] |

# 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"]}} |