# Homework: JavaScript Associative Arrays and Objects

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.

## 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.

## – \* Uncle Scrooge’s Bag

Scrooge McDuck has worked his way up the financial ladder from humble immigrant roots. As a young boy, he took up a job polishing and shining boots in his native Glasgow. His turning point was when a ditchdigger paid him with 1875 rotten coins, which was useless. Enraged, Scrooge vowed to never be taken advantage of again.

For that reason he knows all the time how much money he has in his bag. Write a **function** that will help Uncle Scrooge to find the exact amount of **gold**, **silver** and **bronze** in his bag.

Think of the input **array** as Uncle Scrooge’s bag. Each element is an **item** from the bag. The **type** of the element must be **а coin** and the amount must be a **positive number** with **0 fraction** in order to be a valid coin. Every **100** coins make **1** gold, every **10** coins make **1** silver and **1** coin is **1** bronze.

For example a **valid** coin elements are the strings “coin 111.00” and “COIN 1001”. “Coin 111.23” on the other side is **not a valid** string because 111.23 is **not an integer value**.

### Input

### The input data will be received as an array. It contains a couple of strings. Each string is containing a type and a value separated by a whitespace. 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 three lines. The** first **line is for the** gold**, the** second **line is for the** silver **and the** third **line is for the** bronze**.** See the examples for better understanding.

### Constraints

* The amount of coins will be a number no greater than **100000**.
* Time limit: 0.3 sec. Memory limit: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| [**'coin 1'**,**'coin 2'**, **'coin 5'**, **'coin 10'**, **'coin 20'**, **'coin 50'**, **'coin 100'**, **'coin 200'**, **'coin 500'**,**'cigars 1'**] | gold : 8  silver : 8  bronze : 8 |
| [**'coin one'**, **'coin two'**, **'coin five'**, **'coin ten'**, **'coin twenty'**, **'coin fifty'**, **'coin hundred'**, **'cigars 1'**] | gold : 0  silver : 0  bronze : 0 |
| [**'coin 1'**, **'coin two'**, **'coin 5'**, **'coin 10.50'**, **'coin 20'**, **'coin 50'**, **'coin hundred'**, **'cigars 1'**] | gold : 0  silver : 7  bronze : 6 |

## \* Flea Racing

One relatively lazy day at the office, your boss decided to bring his pet dog with him to work. Unfortunately it had been infested with fleas that started to jump all over the office. Our creative programmers decided to catch them and race them for entertainment. Each flea would be examined carefully and its **jumping distance** calculated. After that a **name** would be assigned to it. Programmers would **pick the length of the track** and align all fleas on the start line. They would also choose the **maximum allowed number of jumps**. Alternating, each flea would jump the distance it can until the maximum jumps have passed for each flea. If no flea has passed the finish line until this moment the winner will be deemed the one that has gotten the **furthest**. If multiple fleas have gotten to the same position, the winner will be the flea (amongst those that have jumped furthest) that has **jumped last** (There must be only 1 winner, right?). To compensate for that order of jumping problem programmers decided that if at any point during the race a flea jumps **exactly on the finish line or behind** **it**, it would be deemed winner and all other fleas denied the chance to jump any more (even if the next flea would have jumped further behind the finish line). Because you are the smartest and most ambitious programmer in your team, you are assigned the task of creating the system for simulating the races. The system should print the **final state of the track** which holds the **current position of each flea**. If any flea has jumped behind the finished line and is deemed winner it should be displayed at the **last possible position of the track**. Each flea is represented on the track by the capitalized version of the first letter in its name. Don’t forget these are **programmer fleas**, so their starting position is 0 not 1. Progress is marked in red in the first example.

### Input

The input will be passed to the first JavaScript function found in your code as **a single** **array,** containing the **number of jumps allowed**, **the length of the track**, **strings** in the format **'name, jumpDistance'** containing the **data for each flea**.

### Output

The output consists of several lines. On the first 2 lines print “#” symbols equal to the length of the track. After those lines (called the audience) print the **final** state of the track. Print another set of audience (2 lines, “#” symbol, length of the track), followed by the winner. Check the examples for correct formatting.

### Constraints

* The **length of the track and flea jump distance capability** will be integers in the range **[1…100]**.
* The array of fleas will contain each **flea** as an individual **array of 2 elements** – **name** and **jump distance**.
* The **amount of participating fleas and number of jumps allowed** will be in range **[1…10]**
* For cosmetic purposes the name of each flea will begin with a different letter.
* Time limit: 0.3 sec. Memory limit: 16 MB.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 10  19  angel, 9  Boris, 10  Georgi, 3  Dimitar, 7 | ###################  ###################  ..................A  ..........B........  ...G...............  .......D...........  ###################  ###################  Winner: angel |  | 3  5  cura, 1  Pepi, 1  UlTraFlea, 1  BOIKO, 1 | #####  #####  ...C.  ...P.  ...U.  ...B.  #####  #####  Winner: BOIKO |

## \* Goshko The Rabbit

Goshko is a small rabbit that really likes to eat all kind of vegetables. One day Goshko manages to sneak into a fertile garden. He starts to move along a path and eat all vegetables he sees. Help Goshko count all the stuff he ate.

### Input

The input is passed to the first JavaScript function found in your code as array of strings. The first string of the input is representing a set of directions describing the path Goshko moved on. It will be in the format:

**'direction, direction, direction, direction,…'**

The next n strings from the input represent a rectangular matrix describing the garden. Each cell of the matrix will contain a string representing the vegetables and the other items that are in this cell. Every matrix cell is separated from the others by a ', ' (a comma and an interval). The vegetables will be in following format: carrots – {!}, cabbage – {\*}, lettuce – {&}, turnip – {#}. All other strings are non-vegetable items (example: 'askjd{!}mnci{\*}aiwj!nw#id').

The matrix input is in the following format:

**'matrixCell, matrixCell, matrixCell, matrixCell,…'**

See the examples below for more clarity.

Goshko is jumping one cell at a time depending on the direction. **At the beginning he is at the first row and the first column of the matrix.** The garden is surrounded by walls so if a direction leads Goshko out of the garden’s bounds **he hits the wall and remains at the same position**. You must count how many times Goshko hits the walls. After Goshko jumps in a cell he eats **all the vegetables** there and leaves **@** at their places. **If Goshko stays at the same position** because he hits a wall he doesn’t eat from the same cell again. The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

On the first line print the vegetables Goshko ate and the count of wall hits in the following format:

**{"&": countOfLettuce,"\*": countOfCabbage,"#": countOfTurnip,"!": countOfCarrots,"wall hits": countOfWallHits}.**

On the second line print all cells Goshko passed, divided by pipes '**|**'. If no cells are passed by Goshko, **print 'no'**. See the given examples below.

### Constraints

* The **starting position** of the rabbit will always be: [**0, 0**].
* The **string's characters**, **row count**, **column count** and **direction** **count** will be between [**1…20**].

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| right, up, up, down  asdf, as{#}aj{g}dasd, kjldk{}fdffd, jdflk{#}jdfj  tr{X}yrty, zxx{\*}zxc, mncvnvcn, popipoip  poiopipo, nmf{X}d{X}ei, mzoijwq, omcxzne | {"&":0,"\*":1,"#":1,"!":0,"wall hits":2}  as@aj{g}dasd|zxx@zxc |
| up, right, left, down  as{!}xnk | {"&":0,"\*":0,"#":0,"!":0,"wall hits":4}  no |

## \* Rolland Garros

You are given a sequence of tennis results in format **"player1 vs. player2 : result"**. The result will be a series of sets in format **"x-y",** separated by whitespaces, where **x** is the number of games won by **player1** and **y** is the number of games won by **player2**. There will be **no whitespaces within the set result**. A match will have between 2 and 5 sets and will **always have a winner** – the player with more sets won. Each set will also have a winner – the player who has won more games in the set.

**Example:** "Novak Djokovic vs. Roger Federer : 6-7 6-2 5-7 6-3 6-2". Novak Djokovic (player1) won by 3 sets to 2. Do not check whether the results are valid tennis scores (4-3 is valid in the context of this problem).

Your task is to write a JavaScript function that prints at the console a JSON string that holds the players, and for each player name, matchesWon, matchesLost, setsWon, setsLost, gamesWon, gamesLost.

### Input

The input data is passed to the first JavaScript function found in your code as an **array of strings**. Each input line holds a match description in format **"player1 vs. player2 : result"**. 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 players, and for each player name, matchesWon, matchesLost, setsWon, setsLost, gamesWon, gamesLost**. The player names will consist of two words; if there is more than one space between a player’s first and last name you should replace it with a single space. The players should be sorted by several criteria:

1. **matchesWon** in **descending** order: the players with the most matches won will be first.
2. **setsWon** in **descending** order *(if there are several players with the same win-loss ratio)* – if two players have the same amount of matches won you should sort them by **setsWon**.
3. **gamesWon** in **descending** order (*if there are several players with the same win-loss and set ratio*) – if two players have the same amount of sets won you should sort them by **gamesWon**.
4. Their **names** sorted **alphabetically** in ascending order in case several players have the same gamesWon, setsWon and gamesWon. (Hint: Use *string*.**localeCompare**(*string*) function to compare the names.)

Please follow exactly the JSON format from the example.

### Constraints

* The number of input lines will be between 1 and 1000.
* The values player1 and player2 will consist of Latin letters and spaces. Their length is between 3 and 50 characters. Leading and trailing whitespaces should be removed. All whitespaces between a player’s first and last name should be replaced with a single space.
* The number of games won by each player in a set will be in the range [0 … 7].
* Whitespace may be found or missing around the separators "vs." and ":".
* Allowed working time for your program: 0.2 seconds. Allowed memory: 16 MB.

### Examples

|  |
| --- |
| **Input** |
| Novak Djokovic vs. Roger Federer : 6-3 6-3  Roger Federer vs. Novak Djokovic : 6-2 6-3  Rafael Nadal vs. Andy Murray : 4-6 6-2 5-7  Andy Murray vs. David Ferrer : 6-4 7-6  Tomas Bedrych vs. Kei Nishikori : 4-6 6-4 6-3 4-6 5-7  Grigor Dimitrov vs. Milos Raonic : 6-3 4-6 7-6 6-2  Pete Sampras vs. Andre Agassi : 2-1  Boris Beckervs.Andre Agassi:2-1 |
| **Output** |
| [{"name":"Andy Murray","matchesWon":2,"matchesLost":0,"setsWon":4,"setsLost":1,"gamesWon":28,"gamesLost":25},{"name":"Kei Nishikori","matchesWon":1,"matchesLost":0,"setsWon":3,"setsLost":2,"gamesWon":26,"gamesLost":25},{"name":"Grigor Dimitrov","matchesWon":1,"matchesLost":0,"setsWon":3,"setsLost":1,"gamesWon":23,"gamesLost":17},{"name":"Roger Federer","matchesWon":1,"matchesLost":1,"setsWon":2,"setsLost":2,"gamesWon":18,"gamesLost":17},{"name":"Novak Djokovic","matchesWon":1,"matchesLost":1,"setsWon":2,"setsLost":2,"gamesWon":17,"gamesLost":18},{"name":"Boris Becker","matchesWon":1,"matchesLost":0,"setsWon":1,"setsLost":0,"gamesWon":2,"gamesLost":1},{"name":"Pete Sampras","matchesWon":1,"matchesLost":0,"setsWon":1,"setsLost":0,"gamesWon":2,"gamesLost":1},{"name":"Tomas Bedrych","matchesWon":0,"matchesLost":1,"setsWon":2,"setsLost":3,"gamesWon":25,"gamesLost":26},{"name":"Milos Raonic","matchesWon":0,"matchesLost":1,"setsWon":1,"setsLost":3,"gamesWon":17,"gamesLost":23},{"name":"Rafael Nadal","matchesWon":0,"matchesLost":1,"setsWon":1,"setsLost":2,"gamesWon":15,"gamesLost":15},{"name":"David Ferrer","matchesWon":0,"matchesLost":1,"setsWon":0,"setsLost":2,"gamesWon":10,"gamesLost":13},{"name":"Andre Agassi","matchesWon":0,"matchesLost":2,"setsWon":0,"setsLost":2,"gamesWon":2,"gamesLost":4}] |