# Homework: JavaScript Functions 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.

## Last Digit of Number

Write a JavaScript function **lastDigitAsText(number)** that returns the **last digit** of given integer as an English word. Write a JS program **lastDigitOfNumber.js** that invokes your function with the sample input data below and prints the output at the console. Examples:

|  |  |
| --- | --- |
| **Input** | **Output** |
| 6 | Six |
| -55 | Five |
| 133 | Three |
| 14567 | Seven |

## N-th Digit of Number

Write a JavaScript function **findNthDigit(arr)** that accepts as a parameter an array of two numbers **num** and **n** and returns the **n-th** digit of given decimal number **num** counted from **right to left**. Return **undefined** when the number does not have n-th digit. Write a JS program **nthDigitOfNumber.js** that invokes your function with the sample input data below and prints the output at the console. Examples:

|  |  |
| --- | --- |
| **Input** | **Output** |
| [1, **6**] | 6 |
| [2, -**5**5] | 5 |
| [6, **9**23456] | 9 |
| [3, 145**1**.78] | 1 |
| [6, 888.88] | The number doesn’t have 6 digits |

## Number with Largest Sum of Digits

Write a JavaScript function **findLargestBySumOfDigits(arr)** that accepts as a parameter an array of numbers or/and strings and returns the element with the largest sum of its digits. The function should take a **variable number of arguments**. It should return **undefined** when 0 arguments are passed or when some of the arguments is not an integer number. Write a JS program **largestSumOfDigits.js** that invokes your function with the sample input data below and prints its output at the console. Examples:

|  |  |
| --- | --- |
| **Input** | **Output** |
| [5, 10, 15, 111] | 15 |
| [33, 44, -99, 0, 20] | -99 |
| ['hello'] | undefined |
| [5, 3.3] | undefined |

## Bigger Than Neighbors

Write a JavaScript function **biggerThanNeighbors(index, arr)** that accept a **number** and an **integer** **array** as parameters. The function should return a Boolean number: whether the element at the given position in the array is **bigger** than its two neighbors (when such exist). It should return **undefined** when the index doesn't exist. Write a JS program that invokes the function with the sample data below and prints the result at the console following the samples below:

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2, [1, 2, **3**, 3, 5] | not bigger |
| 2, [1, 2, **5**, 3, 4] | bigger |
| 5, [1, 2, 5, 3, 4] | invalid index |
| 0, [1, 2, 5, 3, 4] | only one neighbor |

## Reverse Every Word in a String

Write a JavaScript function **reverseWordsInString(str)** that accepts as a parameter a stringand reverses the characters of every word in the string but leaves the words in the **same order**. Words are considered to be sequences of characters separated by spaces. Write a JavaScript program **reverseWords.js** that prints on the console the output of the examples below:

|  |  |
| --- | --- |
| **Input** | **Output** |
| 'Hello, how are you.' | ,olleH woh era .uoy |
| 'Life is pretty good, isn’t it?' | efiL si ytterp ,doog t'nsi ?ti |

## Count Number of DIVs

Write a JavaScript function **countDivs(html)** to count the number of all DIVs in given HTML fragment passed as string. Write a JS program **countOfDivs.js** that invokes your function and prints the output at the console. Examples:

|  |  |
| --- | --- |
| **Input** | **Output** |
| <!DOCTYPE html>  <html>  <head lang="en">  <meta charset="UTF-8">  <title>index</title>  <script src="/yourScript.js" defer></script>  </head>  <body>  <div id="outerDiv">  <div  class="first">  <div><div>hello</div></div>  </div>  <div>hi<div></div></div>  <div>I am a div</div>  </div>  </body>  </html> | 7 |

## Find Youngest Person

Write a JavaScript function **findYoungestPerson(persons)** that accepts as parameter an **array** of **persons,** finds the youngest person and returns his **full name**. Write a JS program **youngestPerson.js** to execute your function for the below examples and print the result at the console.

|  |  |
| --- | --- |
| **Input** | **Output** |
| var persons = [  { firstname : 'George', lastname: 'Kolev', age: 32},  { firstname : 'Bay', lastname: 'Ivan', age: 81},  { firstname : 'Baba', lastname: 'Ginka', age: 40}]  findYoungestPerson(persons); | The youngest person is George Kolev |

## \* Sum of Two Huge Numbers

Write a JavaScript function **sumTwoHugeNumbers(value)** that accepts as parameter an array containing the two numbers. The input numbers are represented as **strings**. You are **not allowed** to use external libraries. The result should be printed on the console. Example:

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['155', '**65**'] | 220 |
| ['123456789', '123456789'] | 246913578 |
| ['887987345974539','4582796427862587'] | 5470783773837126 |
| ['347135713985789531798031509832579382573195807',  '817651358763158761358796358971685973163314321'] | 164787072748948293156827868804265355736510128 |

## \* Array Prototype Function

Write a JavaScript function **removeItem(value)** that accept as parameter a number or string. The function should **remove** all elements with the given value from an array. Attach the function to the **Array** type. You may need to read about **prototypes in JavaScript** and how to **attach** methods to object types. You should **return as a result** the modified array. Write a sample program to demonstrate that your function works correctly for the examples below:

|  |  |
| --- | --- |
| **Input** | **Output** |
| var arr = [1, 2, 1, 4, 1, 3, 4, 1, 111, 3, 2, 1, '1'];  arr.removeItem(1); | [2, 4, 3, 4, 111, 3, 2, '1'] |
| var arr = ['hi', 'bye', 'hello' ];  arr.removeItem('bye'); | ['hi', 'hello'] |

## \* Deep Copy of Object

Write a JavaScript function **clone(obj)** that accepts as parameter an object of **reference type**. The function should **return** a **deep copy** of the object. Write a second function **compareObjects(obj, objCopy)** that compare the two objects **by reference (==)** and print on the console the output given below. Read in Internet about "**deep copy**"of an object and how to create it. Examples:

|  |  |
| --- | --- |
| **Input** | **Output** |
| var a = {name: 'Pesho', age: 21}  var b = clone(a); // a deep copy  compareObjects(a, b); | a == b --> false |
| var a = {name: 'Pesho', age: 21} ;  var b = a; // not a deep copy  compareObjects(a, b); | a == b --> true |

## \*\* Group Persons

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

|  |
| --- |
| **Input** |
| var persons = {  people.push(new Person("Scott", "Guthrie", 38));  people.push(new Person("Scott", "Johns", 36));  people.push(new Person("Scott", "Hanselman", 39));  people.push(new Person("Jesse", "Liberty", 57));  people.push(new Person("Jon", "Skeet", 38));  };  group(persons, '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** |
| group(persons, '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** |
| group(persons, '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 **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"]}} |