Lab: Objects and Associative Arrays

Problems for in-class lab for the "JavaScript Fundamentals" course @ SoftUni. Submit your solutions in the SoftUni judge system at https://judge.softuni.bg/Contests/315/.

1. Towns to JSON

You're tasked to create and print a JSON from a text table. You will receive input as an array of strings, where each string represents a row of a table, with values on the row encompassed by pipes **"|"** and optionally spaces. The table will consist of exactly 3 columns **"Town"**, **"Latitude"** and **"Longitude"**. The **latitude** and **longitude** columns will always contain **valid numbers**. Check the examples to get a better understanding of your task.

The **input** comes as an array of strings – the first string contains the table's headings, each next string is a row from the table.

The **output** should be printed on the console – for each entry row in the input print the object representing it.

Examples

Input
['| Town | Latitude | Longitude |', '| Sofia | 42.696552 | 23.32601 |', '| Beijing | 39.913818 | 116.363625 |'];
Output
[{"Town":"Sofia","Latitude":42.69,"Longitude":23.32}, {"Town":"Beijing","Latitude":39.91,"Longitude":116.36}]
Input
['| Town | Latitude | Longitude |', '| Veliko Turnovo | 43.0757 | 25.6172 |', '| Monatevideo | 34.50 | 56.11 |']
Output
[{"Town":"Veliko Turnovo","Latitude":43.0757,"Longitude":25.6172}, {"Town":"Monatevideo","Latitude":34.5,"Longitude":56.11}]

2. Score to HTML

You are given a JSON string representing an array of objects, parse the JSON and create a table using the supplied objects. The table should have 2 columns **"name"** and **"score"**, each object in the array will also have these keys.

Any text elements must also be **escaped** in order to ensure no dangerous code can be passed.

You can either write the HTML escape function yourself or use the one from the Strings and Regular Expressions Lab.

The **input** comes as a single string argument – the array of objects as a JSON.

The **output** should be printed on the console – a table with 2 columns - **"name"** and **"score"**, containing the values from the objects as rows.

Input
'[{"name":"Pesho","score":479},{"name":"Gosho","score":205}]'
Output

```
<table>
  <tr><th>name</th><th>score</th></tr>
  <tr><td>Pesho</td><td>479</td></tr>
  <tr><td>Gosho</td><td>205</td></tr>
</table>
```

Input
'[{"name":"Pesho & Kiro","score":479},{"name":"Gosho, Maria & Viki","score":205}]'

Output
`<table><tr><th>name</th><th>score</th></tr><tr><td>Pesho &amp; Kiro</td><td>479</td></tr><tr><td>Gosho, Maria &amp; Viki</td><td>205</td></tr></table>`

3. From JSON to HTML Table

You're tasked with creating an HTML table of students and their scores. You will receive a single string representing an **array of objects**, the **table's headings** should be equal to the **objects' keys**, while **each object's values** should be a **new entry** in the table. Any **text values** in an object should be **escaped**, in order to avoid introducing dangerous code into the HTML.

Object's keys will always be the **same.**

The **input** comes as single string argument – the array of objects.

The **output** should be printed on the console – for each entry row in the input print the object representing it.

HTML

You are provided with an HTML file to test your table in the browser.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>FromJSONToHTMLTable</title>
    <style>
        table,th{
            border: groove;
            border-collapse: collapse;
        }
        td{
            border: 1px solid black;
        }
        td,th{
            padding: 5px;
        }
    </style>
</head>
<body>
    <div id="wrapper">
    </div>
    <script>
        function fromJSONToHTMLTable(input){
            //Write your code here
        }
        window.onload = function(){
            let container = document.getElementById('wrapper');
            container.innerHTML = fromJSONToHTMLTable(['[{"Name":"Tomatoes &
```

```
Chips","Price":2.35},{"Name":"J&B Chocolate","Price":0.96}]']);
        };
    </script>
</body>
</html>
```

Examples

Input
'[{"Name":"Tomatoes & Chips","Price":2.35},{"Name":"J&B Chocolate","Price":0.96}]'
Output
<table> <tr><th>Name</th><th>Price</th></tr> <tr><td>Tomatoes & Chips</td><td>2.35</td></tr> <tr><td>J&B Chocolate</td><td>0.96</td></tr> </table>
Input
'[{"Name":"Pesho -a","Age":20,"City":"Sofia"}, {"Name":"Gosho","Age":18,"City":"Plovdiv"},{"Name":"Angel","Age":18,"City":"Veliko Tarnovo"}]'
Output
<table> <tr><th>Name</th><th>Age</th><th>City</th></tr> <tr><td>Pesho <div>-a</td><td>20</td><td>Sofia</td></tr> <tr><td>Gosho</td><td>18</td><td>Plovdiv</td></tr> <tr><td>Angel</td><td>18</td><td>Veliko Tarnovo</td></tr> </table>

4. Sum by Town

You're tasked with calculating the total sum of income for a number of Towns. You will receive an array of strings representing towns and their incomes, every **even** index will be a **town** and every **odd** index will be an **income** belonging to that town. Create an object that will hold all the **towns as keys** and their **total income** (the sum of their incomes) **as values** to those keys and print it as a JSON.

The **input** comes as an array of strings - each even index is the name of a town and each odd index is an income belonging to that town.

The **output** should be printed on the console - JSON representation of the object containing all towns and their total incomes.

Examples

Input	Output
Sofia 20 Varna 3 Sofia 5 Varna 4	{"Sofia":25,"Varna":7}

Sofia 20 Varna 3 sofia 5 varna 4	{"Sofia":20,"Varna":3,"sofia":5,"varna":4}

5. Count Words in a Text

You are tasked to count the number of words in a text using an object as an associative array, any combination of letters, digits and _ (underscore) should be counted as a word. The words should be stored in the object as properties - the **key** being the **word** and the **value** being the **amount of times the word is contained in the text**.

The **input** comes as an array of strings containing one entry - the text whose words should be counted. The text may consist of more than one sentence.

The **output** should be printed on the console - the JSON representation of the object containing the words.

Examples

Input
Far too slow, you're far too slow.
Output
{"Far":1,"too":2,"slow":2,"you":1,"re":1,"far":1}
Input
JS devs use Node.js for server-side JS.-- JS for devs
Output
{"JS":3,"devs":2,"use":1,"Node":1,"js":1,"for":2,"server":1,"side":1}

6. Count Words with Maps

You are tasked to count the number of words in a text using a Map, any combination of letters, digits and _ (underscore) should be counted as a word. The words should be stored in a Map - the **key** being the **word** and the **value** being the **amount of times the word is contained in the text**. The matching should be **case insensitive**. Print the words in a **sorted order**.

The **input** comes as an array of strings containing one entry - the text whose words should be counted. The text may consist of more than one sentence.

The **output** should be printed on the console - print each word in the map in the format **"'<word>' -> <count> times"**, each on a new line.

Examples

Input
Far too slow, you're far too slow.
Output
'far' -> 2 times

're' -> 1 times 'slow' -> 2 times 'too' -> 2 times 'you' -> 1 times
Input
JS devs use Node.js for server-side JS. JS devs use JS. -- JS for devs --
Output
'devs' -> 3 times 'for' -> 2 times 'js' -> 6 times 'node' -> 1 times 'server' -> 1 times 'side' -> 1 times 'use' -> 2 times

7. Populations in Towns

You have been tasked to create a register for different **towns** and their **population**.

The **input** comes as array of strings. Each element will contain data for a town and its population in the following format:

"**{townName} <-> {townPopulation}**"

If you receive the same town twice, **you should add** the **given population** to the **current one**.

As **output**, you must print all the towns, and their population.

Examples

Input	Output
Sofia <-> 1200000 Montana <-> 20000 New York <-> 10000000 Washington <-> 2345000 Las Vegas <-> 1000000	Sofia : 1200000 Montana : 20000 New York : 10000000 Washington : 2345000 Las Vegas : 1000000

Input	Output
Istanbul <-> 100000 Honk Kong <-> 2100004 Jerusalem <-> 2352344 Mexico City <-> 23401925 Istanbul <-> 1000	Istanbul : 101000 Honk Kong : 2100004 Jerusalem : 2352344 Mexico City : 23401925

8. City Markets

You have been tasked to follow the sales of products in the different towns. For every town you need to keep track of all the products sold, and for every product, the amount of total income.

The **input** comes as array of strings. Each element will represent data about a product and its sales. The format of input is:

{town} -> {product} -> {amountOfSales} : {priceForOneUnit}

The **town** and **product** are both **strings**. The **amount of sales** and **price for one unit** will be **numbers**. Store all towns, for every town, store its products, and for every product, its amount of **total income**. The total income is calculated with the following formula - **amount of sales * price for one unit**. If you receive as input a town you already have, you should just **add** the **new product** to it.

As **output** you must print every town, its products and their total income in the following format:

"Town - {townName}

```
$$${product1Name} : {productTotalIncome}
$$${product2Name} : {productTotalIncome}
...”
```

The **order of output** for each of those entries is – by **order of entrance**.

Examples

Input	Output
Sofia -> Laptops HP -> 200 : 2000 Sofia -> Raspberry -> 200000 : 1500 Sofia -> Audi Q7 -> 200 : 100000 Montana -> Portokals -> 200000 : 1 Montana -> Qgodas -> 20000 : 0.2 Montana -> Chereshas -> 1000 : 0.3	Town - Sofia $$$Laptops HP : 400000 $$$Raspberry : 300000000 $$$Audi Q7 : 20000000 Town - Montana $$$Portokals : 200000 $$$Qgodas : 4000 $$$Chereshas : 300

9. Lowest Prices in Cities

You will be given several towns, with products and their price. You need to find **the lowest price** for **every product** and **the town it is sold at** for that price.

The **input** comes as array of strings. Each element will hold data about a **town**, **product**, and **its price** at that town. The **town** and **product** will be **strings**; the **price** will be a **number**. The input will come in the following format:

```
{townName} | {productName} | {productPrice}
```

If you receive the same **town** and **product more than once,** you should **update** the **old value** with the **new one**.

As **output** you must print **each product** with its **lowest price** and **the town** at which the product is **sold at that price**. If **two towns share** the **same lowest price**, print the one that was **entered first**.
The output, for every product, should be in the following format:

```
{productName} -> {productLowestPrice} ({townName})
```

The **order of output** is – **order of entrance**. See the examples for more info.

Examples

Input	Output
Sample Town | Sample Product | 1000 Sample Town | Orange | 2 Sample Town | Peach | 1 Sofia | Orange | 3 Sofia | Peach | 2 New York | Sample Product | 1000.1 New York | Burger | 10	Sample Product -> 1000 (Sample Town) Orange -> 2 (Sample Town) Peach -> 1 (Sample Town) Burger -> 10 (New York)

10. Extract Unique Words

Write a JS function that **extracts** all **UNIQUE** words from a **valid text**, and **stores them**. Ensure that there are **NO duplicates** in the stored words. Once you find a word, there is no need for you to store it again if you meet it again in the text. You also need to make all characters from the words you've stored – **lowercase**.

The **input** comes as array of strings. Each element will represent a sentence.

The **output** is all of the unique words you've found, each with each, **separated** by a **coma and a space**, printed in the order in which you've found them.

Examples

Input	Output
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque quis hendrerit dui. Quisque fringilla est urna, vitae efficitur urna vestibulum fringilla. Vestibulum dolor diam, dignissim quis varius non, fermentum non felis. Vestibulum ultrices ex massa, sit amet faucibus nunc aliquam ut. Morbi in ipsum varius, pharetra diam vel, mattis arcu. Integer ac turpis commodo, varius nulla sed, elementum lectus. Vivamus turpis dui, malesuada ac turpis dapibus, congue egestas metus.	lorem, ipsum, dolor, sit, amet, consectetur, adipiscing, elit, pellentesque, quis, hendrerit, dui, quisque, fringilla, est, urna, vitae, efficitur, vestibulum, diam, dignissim, varius, non, fermentum, felis, ultrices, ex, massa, faucibus, nunc, aliquam, ut, morbi, in, pharetra, vel, mattis, arcu, integer, ac, turpis, commodo, nulla, sed, elementum, lectus, vivamus, malesuada, dapibus, congue, egestas, metus

Input	Output
Interdum et malesuada fames ac ante ipsum primis in faucibus. Vestibulum volutpat lacinia blandit. Pellentesque dignissim odio in hendrerit lacinia. Vivamus placerat porttitor purus nec hendrerit. Aliquam erat volutpat. Donec ac augue ligula. Praesent venenatis sapien vitae libero ornare, nec pulvinar velit finibus. Proin dui neque, rutrum vel dolor ut, placerat blandit sapien. Pellentesque at est arcu. Nullam eget orci laoreet, feugiat nisi vitae, egestas libero. Pellentesque pulvinar aliquet felis. Interdum et malesuada fames ac ante ipsum primis in faucibus. Etiam sit amet nisl ex. Sed lacinia pretium metus quis fermentum. Praesent a ante suscipit, efficitur risus cursus, scelerisque risus.	interdum, et, malesuada, fames, ac, ante, ipsum, primis, in, faucibus, vestibulum, volutpat, lacinia, blandit, pellentesque, dignissim, odio, hendrerit, vivamus, placerat, porttitor, purus, nec, aliquam, erat, donec, augue, ligula, praesent, venenatis, sapien, vitae, libero, ornare, pulvinar, velit, finibus, proin, dui, neque, rutrum, vel, dolor, ut, at, est, arcu, nullam, eget, orci, laoreet, feugiat, nisi, egestas, aliquet, felis, etiam, sit, amet, nisl, ex, sed, pretium, metus, quis, fermentum, a, suscipit, efficitur, risus, cursus, scelerisque