# Lab: Objects and Associative Arrays

Problems for in-class lab for the [“JavaScript Fundamentals” course @ SoftUni](https://softuni.bg/courses/javascript-fundamentals). Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/315/>.

## 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 an array of objects wrapped in **JSON.stringify()**. Latitude and longitude must be parsed to **numbers**!

### 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}] |

## 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> |

## 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**: 1**px solid black**;  }  **td**,**th**{  **padding**: 5**px**;  }  </**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 &amp; Chips</td><td>2.35</td></tr>  <tr><td>J&amp;B Chocolate</td><td>0.96</td></tr>  </table> |
| Input |
| '[{"Name":"Pesho <div>-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 &lt;div&gt;-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> |

## 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} |

## 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} |

## 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 |

## 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** |  | **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 |  | ['Istanbul <-> 100000',  'Honk Kong <-> 2100004',  'Jerusalem <-> 2352344',  'Mexico City <-> 23401925',  'Istanbul <-> 1000'] | Istanbul : 101000  Honk Kong : 2100004  Jerusalem : 2352344  Mexico City : 23401925 |

## 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 |

## 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) |

## 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 |

## Heroic Inventory

In the era of heroes, every hero has his own items which make him unique. Create a function which creates a **register for the heroes**, with their **names**, **level**, and **items**, if they have such. The register should accept data in a specified format, and return it presented in a specified format.

The **input** comes as array of strings. Each element holds data for a hero, in the following format:

“{heroName} / {heroLevel} / {item1}, {item2}, {item3}...”

You must store the data about every hero. The **name** is a **string**, the **level** is a **number** and the items are all **strings.**

The **output** is a **JSON representation** of the data for all the heroes you’ve stored. The data must be an **array of all the heroes**. Check the examples for more info.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['Isacc / 25 / Apple, GravityGun',  'Derek / 12 / BarrelVest, DestructionSword',  'Hes / 1 / Desolator, Sentinel, Antara'] | [{"name":"Isacc","level":25,"items":["Apple","GravityGun"]},{"name":"Derek","level":12,"items":["BarrelVest","DestructionSword"]},{"name":"Hes","level":1,"items":["Desolator","Sentinel","Antara"]}] |
|  |  |
| **Input** | **Output** |
| ['Jake / 1000 / Gauss, HolidayGrenade'] | [{"name":"Jake","level":1000,"items":["Gauss","HolidayGrenade"]}] |

### Hints

* We need an array that will hold our hero data. That is the first thing we create.



* Next, we need to loop over the whole input, and process it. Let’s do that with a simple for loop.



* Every element from the input holds data about a hero, however the **elements from the data** we need are **separated by some delimiter**, so we just split each string with that **delimiter**.
* Next, we need to take the elements from the **string array**, which is a result of the **string split**, and parse them.



* However, if you do this, you could get quite the error in the current logic. If you go up and read the problem definition again, you will notice that there might be a **case** where the hero **has** **no items**; in that case, if we try to take the **3rd element** of the currentHeroArguments array, it will **result in an error**. That is why we need to perform a simple check.



* If **there are any items** in the **input**, the **variable** will be set to the **split version of them**. If not, it will just remain an **empty array**, **as it is supposed to**.
* We have now extracted the needed data – we have stored the **input name** in a **variable**, we have parsed the **given level** to a **number**, and we have also **split** the **items** that the **hero holds** by their **delimiter**, which would result in a **string array** of elements. By definition, the **items** are **strings**, so we don’t need to process the array we’ve made anymore.
* Now what is left is to add that data into **an object** and **add** that object to the **array**.



* Lastly, we need to turn the array of objects we have made, into a JSON string, which is done by the JSON.stringify() function



## JSON’s Table

JSON’s Table is a magical table which turns JSON data into an HTML table. You will be given **JSON strings** holding data about employees, including their **name**, **position** and **salary**. You need to **parse that data** into **objects**, and create an **HTML table** which holds the data for each **employee on a different row**, as **columns**.

The **name** and **position** of the employee are **strings**, the **salary** is a **number**.

The **input** comes as array of strings. Each element is a JSON string which represents the data for a certain employee.

The **output** is the HTML code of a table which holds the data exactly as explained above. Check the examples for more info.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['{"name":"Pesho","position":"Promenliva","salary":100000}',  '{"name":"Teo","position":"Lecturer","salary":1000}',  '{"name":"Georgi","position":"Lecturer","salary":1000}'] | <table>  <tr>  <td>Pesho</td>  <td>Promenliva</td>  <td>100000</td>  </tr>  <tr>  <td>Teo</td>  <td>Lecturer</td>  <td>1000</td>  </tr>  <tr>  <td>Georgi</td>  <td>Lecturer</td>  <td>1000</td>  </tr>  </table> |

### Hints

* You might want to **escape the HTML**. Otherwise you might find yourself victim to vicious JavaScript **code in the input**, which aims only to hack you.

## Cappy Juice

You will be given different juices, as **strings**. You will also **receive quantity** as a **number**. If you receive a juice, you already have, **you must sum** the **current quantity** of that juice, with the **given one**. When a juice reaches **1000 quantity**, it produces a bottle. You must **store all produced bottles** and you must **print them** at the end.

**Note:** **1000 quantity** of juice is **one bottle**. If you happen to have **more than 1000**, you must make **as much bottles as you can**, and store **what** **is** **left** from the juice.

**Example:** **You have 2643 quantity** of Orange Juice – this is **2 bottles** of Orange Juice and **643 quantity left**.

The **input** comes as array of strings. Each element holds data about a juice and quantity in the following format:

“{juiceName} => {juiceQuantity}”

The **output** is the produced bottles. The bottles are to be printed in **order of obtaining the bottles**. Check the second example bellow - even though we receive the Kiwi juice first, we don’t form a bottle of Kiwi juice until the 4th line, at which point we have already create Pear and Watermelon juice bottles, thus the Kiwi bottles appear last in the output.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| ['Orange => 2000',  'Peach => 1432',  'Banana => 450',  'Peach => 600',  'Strawberry => 549'] | Orange => 2  Peach => 2 |  | ['Kiwi => 234',  'Pear => 2345',  'Watermelon => 3456',  'Kiwi => 4567',  'Pear => 5678',  'Watermelon => 6789'] | Pear => 8  Watermelon => 10  Kiwi => 4 |

## Store Catalogue

You have to create a sorted catalogue of store products. You will be given the products’ names and prices. You need to order them by **alphabetical order**.

The **input** comes as array of strings. Each element holds info about a product in the following format:

“{productName} : {productPrice}”

The **product’s name** will be a **string**, which will **always** **start with a capital letter**, and the **price** will be **a number**. You can safely assume there will be **NO duplicate product input**. The comparison for alphabetical order is **case-insensitive**.

As **output** you must print all the products in a specified format. They must be ordered **exactly as specified above**. The products must be **divided into groups**, by the **initial of their name**. The **group’s initial should be printed**, and after that the products should be printed with **2 spaces before their names**. For more info check the examples.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| ['Appricot : 20.4',  'Fridge : 1500',  'TV : 1499',  'Deodorant : 10',  'Boiler : 300',  'Apple : 1.25',  'Anti-Bug Spray : 15',  'T-Shirt : 10'] | A  Anti-Bug Spray: 15  Apple: 1.25  Appricot: 20.4  B  Boiler: 300  D  Deodorant: 10  F  Fridge: 1500  T  T-Shirt: 10  TV: 1499 |  | ['Banana : 2',  'Rubic's Cube : 5',  'Raspberry P : 4999',  'Rolex : 100000',  'Rollon : 10',  'Rali Car : 2000000',  'Pesho : 0.000001',  'Barrel : 10'] | B  Banana: 2  Barrel: 10  P  Pesho: 0.000001  R  Rali Car: 2000000  Raspberry P: 4999  Rolex: 100000  Rollon: 10  Rubic's Cube: 5 |

## Auto-Engineering Company

You are tasked to create a register for a company that produces cars. You need to store **how many cars** have been produced from a **specified model** of a **specified brand**.

The **input** comes as array of strings. Each element holds information in the following format:

“{carBrand} | {carModel} | {producedCars}”

The car **brands** and **models** are **strings**, the **produced cars** are **numbers**. If the **car brand** you’ve received **already exists**, just add the **new** **car model** to it with the **produced cars** **as its value**. If even the car model exists, just **add** the **given value** to the **current one**.

As **output** you need to print – **for every car brand**, the **car models**, and **number of cars produced** from that model. The output format is:

“{carBrand}

###{carModel} -> {producedCars}

###{carModel2} -> {producedCars}

...”

The order of printing is the order in which the brands and models first appear in the input. The first brand in the input should be the first printed and so on. For each brand, the first model received from that brand, should be the first printed and so on.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['Audi | Q7 | 1000',  'Audi | Q6 | 100',  'BMW | X5 | 1000',  'BMW | X6 | 100',  'Citroen | C4 | 123',  'Volga | GAZ-24 | 1000000',  'Lada | Niva | 1000000',  'Lada | Jigula | 1000000',  'Citroen | C4 | 22',  'Citroen | C5 | 10'] | Audi  ###Q7 -> 1000  ###Q6 -> 100  BMW  ###X5 -> 1000  ###X6 -> 100  Citroen  ###C4 -> 145  ###C5 -> 10  Volga  ###GAZ-24 -> 1000000  Lada  ###Niva -> 1000000  ###Jigula -> 1000000 |

### Hints

* The **Map structure** should be perfect for this problem.

## System Components

You will be given a register of systems with components and subcomponents. You need to build an ordered database of all the elements that have been given to you.

The elements are registered in a very simple way. When you have processed all of the input data, you must print them in a specific order. For every System you must print its components in a specified order, and for every Component, you must print its Subcomponents in a specified order.

The **Systems** you’ve stored must be ordered by **amount of components**, in **descending order**, as **first criteria**, and by **alphabetical order** as **second criteria**. The **Components** must be ordered by **amount of Subcomponents**, in **descending order**.

The **input** comes as array of strings. Each element holds **data** about a **system**, a **component** in that **system**, and a **subcomponent** in that **component**. If the given **system already exists**, you should just **add the new component** to it. If even the **component exists**, you should just **add** the **new subcomponent** to it. The **subcomponents** will **always be** **unique**. The input format is:

“{systemName} | {componentName} | {subcomponentName}”

All of the elements are strings, and can contain **any ASCII character**. The **string comparison** for the alphabetical order is **case-insensitive**.

As **output** you need to print all of the elements, ordered exactly in the way specified above. The format is:

“{systemName}

|||{componentName}

|||{component2Name}

||||||{subcomponentName}

||||||{subcomponent2Name}

{system2Name}

...”

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['SULS | Main Site | Home Page', 'SULS | Main Site | Login Page', 'SULS | Main Site | Register Page', 'SULS | Judge Site | Login Page', 'SULS | Judge Site | Submittion Page', 'Lambda | CoreA | A23', 'SULS | Digital Site | Login Page', 'Lambda | CoreB | B24', 'Lambda | CoreA | A24', 'Lambda | CoreA | A25', 'Lambda | CoreC | C4', 'Indice | Session | Default Storage', 'Indice | Session | Default Security'] | Lambda |||CoreA ||||||A23 ||||||A24 ||||||A25 |||CoreB ||||||B24 |||CoreC ||||||C4 SULS |||Main Site ||||||Home Page ||||||Login Page ||||||Register Page |||Judge Site ||||||Login Page ||||||Submittion Page  |||Digital Site  ||||||Login Page Indice  |||Session  ||||||Default Storage  ||||||Default Security |

### Hints

* Creating a sorting function with two criteria might seem a bit daunting at first, but it can be simplified to the following:
  + If elements **a** and **b** are different based on the **first criteria**, then that result is the result of the sorting function, checking the second criteria is not required.
  + If elements **a** and **b** are **equal** based on the **first criteria**, then the result of comparing **a** and **b** on the **second criteria** is the result of the sorting.

## Usernames

You are tasked to create a catalogue of usernames. The usernames will be strings that **may contain any ASCII** character. You **need to order** them **by their length**, in **ascending order**, as **first criteria**, and by **alphabetical order** as **second criteria**.

The **input** comes as array of strings. Each element represents a **username**. Sometimes the input may contain **duplicate usernames**. Make it so that there are **NO duplicates** in the output.

The **output** is all of the usernames, **ordered** exactly as **specified above** – each printed on a new line.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| ['Ashton', 'Kutcher', 'Ariel', 'Lilly', 'Keyden', 'Aizen', 'Billy', 'Braston'] | Aizen Ariel Billy Lilly Ashton Keyden Braston Kutcher |  | ['Denise', 'Ignatius', 'Iris', 'Isacc', 'Indie', 'Dean', 'Donatello', 'Enfuego', 'Benjamin', 'Biser', 'Bounty', 'Renard', 'Rot'] | Rot Dean Iris Biser Indie Isacc Bounty Denise Renard Enfuego Benjamin Ignatius Donatello |

### Hints

* Try to find a **structure** which **does NOT allow duplicates**, it will be best for the current problem.

## Unique Sequences

You are tasked with storing sequences of numbers. You will receive an unknown amount of **arrays containing numbers** from which you must store only the **unique** arrays (duplicate arrays should be discarded). An array is considered the **same** (**NOT unique**) if it contains the **same numbers** as another array**, regardless of their order**.

After storing all arrays, your program should print them back in **ascending** order based on their **length**, if two arrays have the same length they should be printed in **order of being received from the input**. Each individual array should be printed in **descending order** in the format **"[a1, a2, a3,… an]"**. Check the examples bellow.

The **input** comes as an array of strings where each entry is a JSON representing an array of numbers.

The **output** should be printed on the console - each array printed on a new line in the format **"[a1, a2, a3,… an]"** , following the above mentioned ordering.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| ["[-3, -2, -1, 0, 1, 2, 3, 4]",  "[10, 1, -17, 0, 2, 13]",  "[4, -3, 3, -2, 2, -1, 1, 0]"] | [13, 10, 2, 1, 0, -17]  [4, 3, 2, 1, 0, -1, -2, -3] |

|  |  |
| --- | --- |
| **Input** | **Output** |
| ["[7.14, 7.180, 7.339, 80.099]",  "[7.339, 80.0990, 7.140000, 7.18]",  "[7.339, 7.180, 7.14, 80.099]"] | [80.099, 7.339, 7.18, 7.14] |

### Hints

* Think of an easy way to compare arrays.
* Sometimes the most obvious collection choice is not the best one.

## \*Arena Tier

Pesho is a pro gladiator, he is struggling to become master of the Arena. // TODO some more story

You will receive **several input lines** in one of the following formats:

"{gladiator} -> {technique} -> {skill}"

"{gladiator} vs {gladiator}"

The gladiator and technique are strings, the given **skill** will be an integer number. You need to keep track of **every gladiator**.

When you receive a **gladiator and his technique and skill**, add him to the gladiator pool, if he isn`t present, else add his technique or update his skill, only if the current technique skill is lower than the new value.

If you receive **"{gladiator} vs {gladiator}"** and both gladiators exist in the tier, they duel with the following rules:

Compare their techniques, if they got at least one in common, the gladiator with better total skill points wins and the other is demoted from the tier -> remove him.

If they don't have techniques in common, the duel isn`t happening and both continue in the Season.

You should end your program when you receive the command "Ave Cesar". At that point you should print the gladiators, **ordered by total skill in desecending order, then ordered by name in ascending order**. Foreach gladiator print their technique and skill, **ordered desecending, then ordered by technique name in ascending order**

### Input / Constraints

You will receive an **array of strings** as a parameter to your solution.

* The input comes in the form of commands in one of the formats specified above.
* Gladiator and technique **will always be one word string, containing no whitespaces**.
* Skill will be an **integer** in the **range [0, 1000]**.
* There will be **no invalid** input lines.
* The programm ends when you receive the command "Ave Cesar".

### Output

* The output format for each gladiator is:

"{gladiator}: {totalSkill} skill"

"- {technique} <!> {skill}"

***Scroll down to see examples.***

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| Pesho -> BattleCry -> 400  Gosho -> PowerPunch -> 300  Stamat -> Duck -> 200  Stamat -> Tiger -> 250  Ave Cesar | Stamat: 450 skill  - Tiger <!> 250  - Duck <!> 200  Pesho: 400 skill  - BattleCry <!> 400  Gosho: 300 skill  - PowerPunch <!> 300 | We order the gladiators by total skill points descending, then by name. We print every technique along its skill ordered descending by skill, then by technique name. |
| **Input** | **Output** |  |
| Pesho -> Duck -> 400  Julius -> Shield -> 150  Gladius -> Heal -> 200  Gladius -> Support -> 250  Gladius -> Shield -> 250  Pesho vs Gladius  Gladius vs Julius  Gladius vs Gosho  Ave Cesar | Gladius: 700 skill  - Support <!> 250  - Shield <!> 250  - Heal <!> 200  Pesho: 400 skill  - Duck <!> 400 | Gladius and Pesho don`t have common technique, so the duel isn`t valid.  Gladius wins vs Julius /common technique: "Shield". Julius is demoted.  Gosho doesn`t exist so the duel isn`t valid.  We print every gladiator left in the tier. |

## \*Game of Epicness

Write a JavaScript program that **determines** the **winner** from **all battles**. You will receive **two** arguments:

The **first** argument is an **array of kingdoms with generals and their army** in the form of an **object** with format:

{ kingdom: String, general: String, army: Number }

Every **general** has his own **army** that fights for a certain **kingdom**. Note that, every **kingdom’s name** is **unique,** and every general’s **name** is **unique** in **this kingdom**. If the **general** already **exists** **in** this **kingdom** **add** the **army** to his current one. After you go through all the kingdoms with their generals with armies and store the information about them, it’s time to start the battles.

The **second** argument is **matrix of strings** showing which **kingdom’s generals** are **fighting** in this format:

**[**

**[ "{AttackingKingdom} ", "{AttackingGeneral}", "{DefendingKingdom} ", "{DefendingGeneral}" ],**

**…**

**]**

The **first two elements** are the **names** of the **attacking general from** certain **kingdom** and the **second two** are the **names** of the **defending general from** certain **kingdom**. **Compare** the two general’s **armies to determine** who **wins** and who **losses** based on who have the **larger army wins.** The **winner’s army increases** with **10%** and the **loser’s army decreases** with **10%.** Keep in mind to **round** them **down** if there is any excess **army** **after the battle.** If there is a **draw**, **do not do anything**. **Keep track** of the **wins** and **losses** for every general’s battle.

Note that, **generals** from the **same kingdom** **cannot** **attack** **each other**.

After you finish with all battles you need to **find** which **kingdom** **wins** the game. To decide that, **first** **order them** by all their **general’s wins (descending)** then by their **losses (ascending),** and finally by the **kingdom’s name** in **ascending alphabetical** order.

### Input

You will receive **two arguments –** an **array of objects** with properties and a **matrix of strings** as shown above.

### Output

Print on the **console** the winning kingdom and **sort** the generals by their **armies in** **descending** order, **formatted** as seen in the examples.

### Constraints

* The **number** of **elements** in the **first input argument** will be in range **[1..100] inclusive**
* The **number** of **elements** in the **second input argument** will be in range **[0..100] inclusive**
* General’s **army** will be always an **integer** in range **[0..1,000,000] inclusive**
* There **will** be **no invalid** **input**
* There **will** be **no matching number** of **armies** in the **output**

### Examples

|  |
| --- |
| **Input** |
| [ { kingdom: "Maiden Way", general: "Merek", army: 5000 },  { kingdom: "Stonegate", general: "Ulric", army: 4900 },  { kingdom: "Stonegate", general: "Doran", army: 70000 },  { kingdom: "YorkenShire", general: "Quinn", army: 0 },  { kingdom: "YorkenShire", general: "Quinn", army: 2000 },  { kingdom: "Maiden Way", general: "Berinon", army: 100000 } ],  [ ["YorkenShire", "Quinn", "Stonegate", "Ulric"],  ["Stonegate", "Ulric", "Stonegate", "Doran"],  ["Stonegate", "Doran", "Maiden Way", "Merek"],  ["Stonegate", "Ulric", "Maiden Way", "Merek"],  ["Maiden Way", "Berinon", "Stonegate", "Ulric"] ] |
| **Output** |
| Winner: Stonegate  /\general: Doran  ---army: 77000  ---wins: 1  ---losses: 0  /\general: Ulric  ---army: 5336  ---wins: 2  ---losses: 1 |
| **Explanation** |
| After you successfully store the kingdoms information, the first battle’s result is victory for the defender Ulric and a loss for the attacker Quinn. Second battle is ignored because the generals are from the same kingdom. Third battle is a victory for Doran and a loss for Merek. Fourth battle is a win for Ulric and a loss for Merek. Fifth battle is a victory for Berinon and a defeat for Ulric. All winners increase their armies with 10% for each win and all losers decrease their armies with 10% for each loss.  The result from the battles are – Stonegate: 3 wins and 1 loss; Maiden Way: 1 win and 2 losses; YorkenShire: 0 wins and 1 loss. Making Stonegate the winner of the games because they have the most wins from kingdoms. |

|  |
| --- |
| **Input** |
| [ { kingdom: "Stonegate", general: "Ulric", army: 5000 },  { kingdom: "YorkenShire", general: "Quinn", army: 5000 },  { kingdom: "Maiden Way", general: "Berinon", army: 1000 } ],  [ ["YorkenShire", "Quinn", "Stonegate", "Ulric"],  ["Maiden Way", "Berinon", "YorkenShire", "Quinn"] ] |
| **Output** |
| Winner: YorkenShire  /\general: Quinn  ---army: 5500  ---wins: 1  ---losses: 0 |
| **Explanation** |
| The first battle between Quinn and Ulric is a draw because they have even armies because of that it is not recorded and their armies size does not change. The second battle is a win for Quinn and a loss for Berinon making YorkenShire the winner of the game with 1 win and 0 losses. |

|  |
| --- |
| **Input** |
| [ { kingdom: "Maiden Way", general: "Merek", army: 5000 },  { kingdom: "Stonegate", general: "Ulric", army: 4900 },  { kingdom: "Stonegate", general: "Doran", army: 70000 },  { kingdom: "YorkenShire", general: "Quinn", army: 0 },  { kingdom: "YorkenShire", general: "Quinn", army: 2000 } ],  [ ["YorkenShire", "Quinn", "Stonegate", "Doran"],  ["Stonegate", "Ulric", "Maiden Way", "Merek"] ] |
| **Output** |
| Winner: Maiden Way  /\general: Merek  ---army: 5500  ---wins: 1  ---losses: 0 |

## Followers

You need to write a function that allows **users in a website to follow one another**. In order for users to be able to follow one another, **they have to be registered in the website**. There are four more possible input strings:  
 - "Welcome, {user}" – if the user is **not registered**, you need to **register him** - "{user} followed {another user}" – the **first user** starts **following the other one**, only if **both of them exist.**

When you finish reading the input, print the results as follows:  
 "Total users registered: {total users}"  
 "1. {first user}: {following} following, {followers} followers"  
 "\* {follower one}"  
 "\* {follower two}"  
 "2. {second user}: {following} following, {followers} followers"  
 "3. {second user}: {following} following, {followers} followers"

Users must be sorted by **count of followers in descending**, then **by username in descending**. For the **user with most followers**, print **their names ordered alphabetically**.

### Input

The **input** will come as an **array of strings.**

### Output

Print the result as described above.

### Constraints

* There will be no invalid input.
* There will be no situation where two users have equal amount of followers and equal amount of followings
* The subscribers will be strings.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Welcome, EmilConrad  Welcome, VenomTheDoctor  Welcome, Saffrona  Saffrona followed EmilConrad  Saffrona followed VenomTheDoctor  EmilConradfollowed VenomTheDoctor  VenomTheDoctorfollowed VenomTheDoctor  Saffrona followed EmilConrad | Total users registered: 3  1. VenomTheDoctor : 0 following, 2 followers  \* EmilConrad  \* Saffrona  2. EmilConrad : 1 following, 1 followers  3. Saffrona : 2 following, 0 followers |
| Welcome, JennaMarbles  JennaMarbles followed Zoella  Welcome, AmazingPhil  JennaMarbles followed AmazingPhil  Welcome, Zoella  Welcome, JennaMarbles  Zoella followed AmazingPhil  Christy followed Zoella  Zoella followed Christy  Welcome, JacksGap  JacksGap followed JennaMarbles  Welcome, PewDiePie  Welcome, Zoella | Total users registered: 5  1. AmazingPhil : 0 following, 2 followers  \* JennaMarbles  \* Zoella  2. JennaMarbles : 1 following, 1 followers  3. Zoella : 1 following, 0 followers  4. PewDiePie : 0 following, 0 followers  5. JacksGap : 1 following, 0 followers |

## Travellers Log

Write a function that stores information about different people travelling in different countries and visiting different landmarks. They will also be spending money to do that, so you need to keep track of their money too! There are two types of input:  
 **"{name} gets {money}"** – if the traveller does not exist already, add him and the money to his account, otherwise just add the money   
 **"{name} visited the {landmark} in {country} – costs {money}"** – if the traveller does **not exist** yet, **add him**, if the traveller has yet **not been in that country**, **add it to his log with the landmark**; if the person **has been to the country**, but **not yet to this landmark**, **add the landmark** to the country in his log, **otherwise ignore**. Keep in mind that all of this happens, **only if the traveller has enough money**! If the traveller **does not have enough money**, print the following message: **"Not enough money to visit {landmark}".** If you add anything to his log, **reduce the money he has with by the given amount**.

### Input

The input will come as an **array of strings**

### Output

After processing all of the data, print the result in the following format:  
**"{name} visited {countries count} countries and has {money left} money left"  
"- {country one} -> {total landmarks} landmarks"  
"-- {landmark one}"**  
**Travellers** should be **sorted** by **total countries visited** by **descending**. **Countries** should be sorted by **total landmarks in descending** and for **each country the landmarks should be sorted alphabetically**.

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['Peter gets 100',  'Peter visited the StatueOfLiberty in USA - 50',  'Bill gets 250',  'Tim visited the ChristTheRedeemer in Brazil - 150',  'Bill gets 400',  'Bill visited the MountFuji in Japan - 600',  'Bill visited the TeatroAmazonas in Brazil - 50',  'Bill gets 150',  'Bill visited the ChristTheRedeemer in Brazil - 150',  'Tim gets 500',  'Bill visited the StatueOfLiberty in USA - 440',  'Tim visited the StatueOfLiberty in USA - 440',  'Maria gets 650',  'Maria visited the StatueOfLiberty in USA - 440',  'Maria visited the CapeCod in USA - 100'] | Not enough money to visit ChristTheRedeemer  Not enough money to visit StatueOfLiberty  Bill visited 2 countries and has 0 money left  - Brazil -> 2 landmarks  -- ChristTheRedeemer  -- TeatroAmazonas  - Japan -> 1 landmarks  -- MountFuji  Peter visited 1 countries and has 50 money left  - USA -> 1 landmarks  -- StatueOfLiberty  Tim visited 1 countries and has 60 money left  - USA -> 1 landmarks  -- StatueOfLiberty  Maria visited 1 countries and has 110 money left  - USA -> 2 landmarks  -- CapeCod  -- StatueOfLiberty |
| ['Peter gets 100',  'Peter visited the StatueOfLiberty in USA - 50',  'Bill gets 250',  'Bill gets 400',  'Peter gets 150',  'Peter visited the ChristTheRedeemer in Brazil - 150'] | Peter visited 2 countries and has 50 money left  - USA -> 1 landmarks  -- StatueOfLiberty  - Brazil -> 1 landmarks  -- ChristTheRedeemer  Bill visited 0 countries and has 650 money left |

## School Grades

In this problem you have to arrange all students by **grade**. You as the secretary of the school principal will process students and store them into a school register before the new school year to hit. Аs a draft you have a list of all the students from **last year** but mixed. Кeep in mind that if a student has a lesser grade than 3 he does not go into the next class. As result of your work, you have to print the entire school register **sorted** in **Ascending order by grade** already filled with all the students from last year in format:

**"{nextGrade} Grade**

**List of student: {All students in that grade}**

**Average annual grade from last year: {average annual grade on the entire class from last year}**

**And empty row {console.log}"**

### Input

The input will be **array**, **with strings**, each containing a student's name, last year's grade, and an annual grade.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['Student name: Mark, Grade: 8, Graduated with an average score: 4.75',  'Student name: Ethan, Grade: 9, Graduated with an average score: 5.66',  'Student name: George, Grade: 8, Graduated with an average score: 2.83',  'Student name: Steven, Grade: 10, Graduated with an average score: 4.20',  'Student name: Joey, Grade: 9, Graduated with an average score: 4.90',  'Student name: Angus, Grade: 11, Graduated with an average score: 2.90',  'Student name: Bob, Grade: 11, Graduated with an average score: 5.15',  'Student name: Daryl, Grade: 8, Graduated with an average score: 5.95',  'Student name: Bill, Grade: 9, Graduated with an average score: 6.00',  'Student name: Philip, Grade: 10, Graduated with an average score: 5.05',  'Student name: Peter, Grade: 11, Graduated with an average score: 4.88',  'Student name: Gavin, Grade: 10, Graduated with an average score: 4.00'] | 9 Grade  List of students: Mark, Daryl  Average annual grade from last year: 5.35  10 Grade  List of students: Ethan, Joey, Bill  Average annual grade from last year: 5.52  11 Grade  List of students: Steven, Philip, Gavin  Average annual grade from last year: 4.42  12 Grade  List of students: Bob, Peter  Average annual grade from last year: 5.02 |

## Browser Logger

As input you will receive an array with **2 parameters**, **OBJECT and STRING ARRAY.**

The object will be in format: {Browser Name}:{Name of the browser}, Open tabs:[…], Recently Closed: […], Browser Logs: […]. Your task is to fill in the object based on the actions we will get in the string of strings.

You can **open** any site in the world as many times as you like, if you do that add it to the open tabs.

You can **close** only these tabs you have **opened already**! If current action contains valid opened site, you should remove it from ‘Open Tabs’ and put it into ‘Recently closed’, otherwise **don’t do anything!**

**Browser Logs** will hold every single **Valid** action which you did (Open and Close).

There's a **special case** in which you can get an action that says have to **clear your history and cache.** That means you should **empty the whole object**.

At the end print the object in format:

**"{Browser name}**

**Open Tabs: {[...]} // Joined by space**

**Recently Closed: {[…]} // Joined by space**

**Browser Logs: {[…]} // Joined by space"**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| {"Browser Name":"Google Chrome","Open Tabs":["Facebook","YouTube","Google Translate"],  "Recently Closed":["Yahoo","Gmail"],  "Browser Logs":["Open YouTube","Open Yahoo","Open Google Translate","Close Yahoo","Open Gmail","Close Gmail","Open Facebook"]},  ['Close Facebook', 'Open StackOverFlow', 'Open Google'] | Google Chrome  Open Tabs: YouTube, Google Translate, StackOverFlow, Google  Recently Closed: Yahoo, Gmail, Facebook  Browser Logs: Open YouTube, Open Yahoo, Open Google Translate, Close Yahoo, Open Gmail, Close Gmail, Open Facebook, Close Facebook, Open StackOverFlow, Open Google |
| {"Browser Name":"Mozilla Firefox",  "Open Tabs":["YouTube"],  "Recently Closed":['Gmail', 'Dropbox'],  "Browser Logs":['Open Gmail', 'Close Gmail', 'Open Dropbox', 'Open YouTube', 'Close Dropbox']},  ['Open Wikipedia', 'Clear History and Cache', 'Open Twitter'] | Mozilla Firefox  Open Tabs: Twitter  Recently Closed:  Browser Logs: Open Twitter |

## Browser Logger

As input you will receive an array with **2 parameters**, **OBJECT and STRING ARRAY.**

The object will be in format: {Browser Name}:{Name of the browser}, Open tabs:[…], Recently Closed: […], Browser Logs: […]. Your task is to fill in the object based on the actions we will get in the string of strings.

You can **open** any site in the world as many times as you like, if you do that add it to the open tabs.

You can **close** only these tabs you have **opened already**! If current action contains valid opened site, you should remove it from ‘Open Tabs’ and put it into ‘Recently closed’, otherwise **don’t do anything!**

**Browser Logs** will hold every single **Valid** action which you did (Open and Close).

There's a **special case** in which you can get an action that says have to **clear your history and cache.** That means you should **empty the whole object**.

At the end print the object in format:

**"{Browser name}**

**Open Tabs: {[...]} // Joined by space**

**Recently Closed: {[…]} // Joined by space**

**Browser Logs: {[…]} // Joined by space"**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| {"Browser Name":"Google Chrome","Open Tabs":["Facebook","YouTube","Google Translate"],  "Recently Closed":["Yahoo","Gmail"],  "Browser Logs":["Open YouTube","Open Yahoo","Open Google Translate","Close Yahoo","Open Gmail","Close Gmail","Open Facebook"]},  ['Close Facebook', 'Open StackOverFlow', 'Open Google'] | Google Chrome  Open Tabs: YouTube, Google Translate, StackOverFlow, Google  Recently Closed: Yahoo, Gmail, Facebook  Browser Logs: Open YouTube, Open Yahoo, Open Google Translate, Close Yahoo, Open Gmail, Close Gmail, Open Facebook, Close Facebook, Open StackOverFlow, Open Google |
| {"Browser Name":"Mozilla Firefox",  "Open Tabs":["YouTube"],  "Recently Closed":['Gmail', 'Dropbox'],  "Browser Logs":['Open Gmail', 'Close Gmail', 'Open Dropbox', 'Open YouTube', 'Close Dropbox']},  ['Open Wikipedia', 'Clear History and Cache', 'Open Twitter'] | Mozilla Firefox  Open Tabs: Twitter  Recently Closed:  Browser Logs: Open Twitter |

## Flight Schedule

You will receive an **array** with **arrays.**

First array (**at index 0**) will hold all flights on **specific** **sector** in the airport. The second array (**at index 1**) will contain **new changed statuses** of **some** of the **flights** at this airport. The third array (**at index 2**) will have a single **string,** who will **be flight status** you need to check. When you put all flights into a **OBJECT,** and change the statuses depends on the new information on the second array. You must print all flights with the given status from the last array.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| [['WN269 Delaware',  'FL2269 Oregon',  'WN498 Las Vegas',  'WN3145 Ohio',  'WN612 Alabama',  'WN4010 New York',  'WN1173 California',  'DL2120 Texas',  'KL5744 Illinois',  'WN678 Pennsylvania'],  ['DL2120 Cancelled',  'WN612 Cancelled',  'WN1173 Cancelled',  'SK430 Cancelled'],  ['Cancelled']  ] | { Destination: 'Alabama', Status: 'Cancelled' }  { Destination: 'California', Status: 'Cancelled' }  { Destination: 'Texas', Status: 'Cancelled' } |
| [['WN269 Delaware',  'FL2269 Oregon',  'WN498 Las Vegas',  'WN3145 Ohio',  'WN612 Alabama',  'WN4010 New York',  'WN1173 California',  'DL2120 Texas',  'KL5744 Illinois',  'WN678 Pennsylvania'],  ['DL2120 Cancelled',  'WN612 Cancelled',  'WN1173 Cancelled',  'SK330 Cancelled'],  ['Ready to fly']  ] | { Destination: 'Delaware', Status: 'Ready to fly' }  { Destination: 'Oregon', Status: 'Ready to fly' }  { Destination: 'Las', Status: 'Ready to fly' }  { Destination: 'Ohio', Status: 'Ready to fly' }  { Destination: 'New', Status: 'Ready to fly' }  { Destination: 'Illinois', Status: 'Ready to fly' }  { Destination: 'Pennsylvania', Status: 'Ready to fly' } |