# Exercise: Objects and JSON

Problems for in-class lab for the [“Technology Fundamentals” course @ SoftUni](https://softuni.bg/trainings/2056/technology-fundamental-september-2018#lesson-9616).   
Submit your solutions in the SoftUni judge system at: [Objects and JSON - Exercise](https://judge.softuni.bg/Contests/1322/Objects-and-JSON-Exercise).

# Classes

## Class Vehicle

Create a class with name **Vehicle** that has the following properties:

**type** – a string

**model** – a string

**parts** – an object that contains:

engine – number (quality of the engine)

power – number

quality – engine \* power

**fuel** – a number

**drive** – a function that receives fuel loss and decreases the fuel of the vehicle by that number

The **constructor** should receive the **type**, the **model**, the **parts as an object** and the **fuel**

In judge post your **class** (**Note: all names should be as described**)

### Example

Test your Vehicle class

|  |  |
| --- | --- |
| **Input** | **Output** |
| let parts = {engine: 6, power: 100}  let vehicle = new Vehicle('a', 'b', parts, 200)  vehicle.drive(100)  console.log(vehicle.fuel)  console.log(vehicle.parts.quality) | 100  600 |

## Class Storage

Create a class **named Storage**. It should have the following **properties**:

**capacity** – a number that **decreases when adding a given quantity** of products in storage

**storage** – **list of products** (object). **Each product** should have:

**name** - a string

**price** – a number (price is for a single piece of product)

**quantity** – a number

**totalCost** – sum of the cost of the products

**addProduct** – a function that receives a product and adds it to the storage

**getProcuts** – a function that returns all the products in storage in JSON format, each on a new line

The **constructor** should receive a **capacity**

Paste only the **Storage class in judge** (**Note: all names should be as described**)

### Example

Test your Storage class

|  |  |
| --- | --- |
| **Input** | **Output** |
| let productOne = {name: 'Cucamber', price: 1.50, quantity: 15}  let productTwo = {name: 'Tomato', price: 0.90, quantity: 25}  let productThree = {name: 'Bread', price: 1.10, quantity: 8}  let storage = new Storage(50)  storage.addProduct(productOne)  storage.addProduct(productTwo)  storage.addProduct(productThree)  storage.getProducts()  console.log(storage.capacity)  console.log(storage.totalCost) | {"name":"Cucamber","price":1.5,"quantity":15}  {"name":"Tomato","price":0.9,"quantity":25}  {"name":"Bread","price":1.1,"quantity":8}  2  53.8 |

# Objects and JSON

## Movies

Write a function that stores information about movies. The info will be **name, director, date**. You can receive several types on input:

**"add movie {movie name}"** – add the movie

**"{movie name} directed by {director}"** – check if the movie exists and then add the director

**"{movie name} on date {date}"** – check if the movie exists and then add the date

At the end print all the movies that have **all the info** (if the movie has no author, name or data, don’t print it) in **JSON format**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['add movie Fast and Furious',  'add movie Godfather',  'Inception directed by [Christopher Nolan](https://www.imdb.com/name/nm0634240/?ref_=tt_ov_dr)',  'Godfather directed by [Francis Ford Coppola](https://www.imdb.com/name/nm0000338/?ref_=tt_ov_dr)',  'Godfather on date 29.07.2018',  'Fast and Furious on date 30.07.2018',  'Batman on date 01.08.2018',  'Fast and Furious directed by [Rob Cohen](https://www.imdb.com/name/nm0003418/?ref_=tt_ov_dr)'] | {"name":"Fast and Furious","director":"Rob Cohen","date":"30.07.2018"}  {"name":"Godfather","director":"Francis Ford Coppola","date":"29.07.2018"} |

## Store provision

You will receive an **array** which contains **2 other arrays**. The first array represents a current stock of the local store. The second array will contain products which the store has ordered for delivery.

The following information applies to both arrays:

Every **odd** index will hold the name of the product and on every **even** index will hold the quantity of that product. You should store them into an **object**, and print them **as an object**.

All of the arrays values will be **strings.**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| [['Chips', '5', 'CocaCola', '9', 'Bananas', '14', 'Pasta', '4', 'Beer', '2'],['Flour', '44', 'Oil', '12', 'Pasta', '7', 'Tomatoes', '70', 'Bananas', '30']] | { Chips: 5,  CocaCola: 9,  Bananas: 44,  Pasta: 11,  Beer: 2,  Flour: 44,  Oil: 12,  Tomatoes: 70 } |

## 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"]}] |

## JSON Towns

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 and formatted to the second decimal point**!

### Examples

|  |
| --- |
| **Input** |
| ['| Town | Latitude | Longitude |',  '| Sofia | 42.696552 | 23.32601 |',  '| Beijing | 39.913818 | 116.363625 |']; |
| **Output** |
| [{"Town":"Sofia","Latitude":42.70,"Longitude":23.33}, {"Town":"Beijing","Latitude":39.91,"Longitude":116.36}] |

## Unique Names

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 |

## 8.\*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** |
| 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 |

## 9.\*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,** which will **be flight status** you need to check. When you put all flights into an **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' } |

## 10. \*Systems Register

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 |