# Exercise: Objects and Classes

## Employees

You're tasked to create a list of employees and their personal numbers.

You will receive an array of strings. Each string is an employee **name** and to assign them a personal number you have to find the **length of the name** (whitespace included).

***Try to use an object***.

At the end print all the list employees in the following format:

**"Name: {employeeName} -- Personal Number: {personalNum}"**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| [  'Silas Butler',  'Adnaan Buckley',  'Juan Peterson',  'Brendan Villarreal'  ] | Name: Silas Butler -- Personal Number: 12  Name: Adnaan Buckley -- Personal Number: 14  Name: Juan Peterson -- Personal Number: 13  Name: Brendan Villarreal -- Personal Number: 18 |

## Towns

You’re tasked to create and print **objects** from a text table.

You will receive the input as an **array** of strings, where each string represents a table row, with values on the row separated by pipes **" | "** and 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 **output** should be **objects**. Latitude and longitude must be parsed to **numbers and formatted to the second decimal point**!

### Examples

|  |
| --- |
| **Input** |
| ['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' } |

## Store Provision

You will receive **two 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 **even** index will hold the **name** of the **product** and on every **odd** index will hold the **quantity** of that **product**. The second array could contain products that are **already in** the local store. If that happens **increase** the **quantity** for the given product .You should store them into an **object**, and print them in the following format: **(product -> quantity)**

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 |

## Movies

Write a function that stores information about movies inside an array. The movies object info must be **name, director** and **date**. You can receive several types of input:

* **"addMovie {movie name}"** – add the movie
* **"{movie name} directedBy {director}"** – check if the movie **exists** and then add the director
* **"{movie name} onDate {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** director, name or date, **don’t** print it) in **JSON format.**

### Examples

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

## Inventory

Create a function which creates a **register for heroes**, with their **names**, **level**, and **items** (if they have such).

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 all of the data for all the heroes you’ve stored **sorted ascending by level** and **the items are sorted alphabetically**. The data must be in the following format for each hero:

**Hero: {heroName}**

**level => {heroLevel}**

**Items => {item1}, {item2}, {item3}**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| [  "Isacc / 25 / Apple, GravityGun",  "Derek / 12 / BarrelVest, DestructionSword",  "Hes / 1 / Desolator, Sentinel, Antara"  ] | Hero: Hes  level => 1  items => Antara, Desolator, Sentinel  Hero: Derek  level => 12  items => BarrelVest, DestructionSword  Hero: Isacc  level => 25  items => Apple, GravityGun |

## Make a Dictionary

You will receive an **array** with **strings in the form of JSON's.**

You have to parse these strings and combine them into **one object**. Every string from the array will hold **terms** and a **description.** If you receive the **same term** **twice** replace it with the **new definition**.

Print every term and definition in that dictionary on new line in format:

**Term: ${term} => Definition: ${definition}**

Don't forget to sort the dictionary **alphabetically** by the terms as in real dictionaries.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| [  '{"Coffee":"A hot drink made from the roasted and ground seeds (coffee beans) of a tropical shrub."}',  '{"Bus":"A large motor vehicle carrying passengers by road, typically one serving the public on a fixed route and for a fare."}',  '{"Boiler":"A fuel-burning apparatus or container for heating water."}',  '{"Tape":"A narrow strip of material, typically used to hold or fasten something."}',  '{"Microphone":"An instrument for converting sound waves into electrical energy variations which may then be amplified, transmitted, or recorded."}'  ] | Term: Boiler => Definition: A fuel-burning apparatus or container for heating water.  Term: Bus => Definition: A large motor vehicle carrying passengers by road, typically one serving the public on a fixed route and for a fare.  Term: Coffee => Definition: A hot drink made from the roasted and ground seeds (coffee beans) of a tropical shrub.  Term: Microphone => Definition: An instrument for converting sound waves into electrical energy variations which may then be amplified, transmitted, or recorded.  Term: Tape => Definition: A narrow strip of material, typically used to hold or fasten something. |

## 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** **Storage**. It should have the following **properties**, while the **constructor** should only receive a **capacity**:

* **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

The class should also have the following **methods:**

* **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

Paste only the **class** **Storage 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 |

## \*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 |

## 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 |