# More Exercise: Objects and Classes

Submit your solutions in the SoftUni judge system at <https://judge.softuni.org/Contests/3793/Objects-and-Classes-More-Exercises>

## 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 to 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** – the 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
* **getProdcuts** – 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);  console.log(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 |
| let productOne = {name: 'Tomato', price: 0.90, quantity: 19};  let productTwo = {name: 'Potato', price: 1.10, quantity: 10};  let storage = new Storage(30);  storage.addProduct(productOne);  storage.addProduct(productTwo);  console.log(storage.totalCost); | 28.1 |

## Catalogue

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

The **input** comes as an **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 |
| [  'Omlet : 5.4',  'Shirt : 15',  'Cake : 59'  ] | C  Cake: 59  O  Omlet: 5.4  S  Shirt: 15 |

## Class Laptop

Create a **class Laptop** that has the following properties:

* **info** – object that contains:
* **producer** – string
* **age** – number
* **brand** – string
* **isOn** – boolean (false by default)
* **turnOn** – a function that **sets the isOn** variable to **true**
* **turnOff** – a function that **sets the isOn** variable to **false**
* **showInfo** – a function that returns the **producer, age, and brand as JSON**
* **quality** – number (every time the laptop **is turned on/off the quality decreases by 1**)
* **getter price** – number (**800 – {age \* 2} + (quality \* 0.5)**)

The **constructor** should receive the **info as an object and the quality.**

### Examples

Test your class.

|  |  |
| --- | --- |
| **Input** | **Output** |
| let info = {producer: "Dell", age: 2, brand: "XPS"}  let laptop = new Laptop(info, 10)  laptop.turnOn()  console.log(laptop.showInfo())  laptop.turnOff()  console.log(laptop.quality)  laptop.turnOn()  console.log(laptop.isOn)  console.log(laptop.price) | {"producer":"Dell","age":2,"brand":"XPS"}  8  true  799.5 |
| let info = {producer: "Lenovo", age: 1, brand: "Legion"}  let laptop = new Laptop(info, 10)  laptop.turnOn()  console.log(laptop.showInfo())  laptop.turnOff()  laptop.turnOn()  laptop.turnOff()  console.log(laptop.isOn) | {"producer":"Lenovo","age":1,"brand":"Legion"}  false |

## Flight Schedule

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

The first array (**at index 0**) will hold all flights on a **specific** **sector** in the airport. The second array (**at index 1**) will contain **newly changed statuses** of **some** of the **flights** at this airport. The third array (**at index 2**) will have a single **string,** which will **be the 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**.

* If the value of the string obtained from the third array is "**Ready to fly**":
* then you must **print** flights that have **not changed** their **status** in the second array
* and automatically **change** the status to "**Ready to fly**"
* Otherwise, print **only flights** that have **changed** their status.

### 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 Vegas', Status: 'Ready to fly' }  { Destination: 'Ohio', Status: 'Ready to fly' }  { Destination: 'New York', Status: 'Ready to fly' }  { Destination: 'Illinois', Status: 'Ready to fly' }  { Destination: 'Pennsylvania', Status: 'Ready to fly' } |

## School Register

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 hits. As a draft, you have a list of all the students from **last year** but mixed. Keep in mind that if a student has a lower score than 3, he does not go into the next class. As a 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 the format:

**`{nextGrade} Grade**

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

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

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

The input will be an **array** with strings, each containing a student's name, last year's grade, and an annual score. The **average annual score from last year** should be **formatted to the second decimal point**.

### 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 score from last year: 5.35  10 Grade  List of students: Ethan, Joey, Bill  Average annual score from last year: 5.52  11 Grade  List of students: Steven, Philip, Gavin  Average annual score from last year: 4.42  12 Grade  List of students: Bob, Peter  Average annual score from last year: 5.02 |
| [  'Student name: George, Grade: 5, Graduated with an average score: 2.75',  'Student name: Alex, Grade: 9, Graduated with an average score: 3.66',  'Student name: Peter, Grade: 8, Graduated with an average score: 2.83',  'Student name: Boby, Grade: 5, Graduated with an average score: 4.20',  'Student name: John, Grade: 9, Graduated with an average score: 2.90',  'Student name: Steven, Grade: 2, Graduated with an average score: 4.90',  'Student name: Darsy, Grade: 1, Graduated with an average score: 5.15'  ] | 2 Grade  List of students: Darsy  Average annual score from last year: 5.15  3 Grade  List of students: Steven  Average annual score from last year: 4.90  6 Grade  List of students: Boby  Average annual score from last year: 4.20  10 Grade  List of students: Alex  Average annual score from last year: 3.66 |

## Browser History

As input, you will receive **two parameters**: **an** **object and a 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 array 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 the current action contains a 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 is a **special case** in which you can get an action that says: **"Clear History and Cache".** That means you should **empty the whole object**.

In the end, print the object in the format:

**{Browser name}**

**Open Tabs: {[…]}** // Joined by comma and space

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

**Browser Logs: {[…]}** // Joined by comma and 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 |

## Sequences

You are tasked with storing sequences of numbers. You will receive an **array of strings; each of them will contain** 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 array should be printed in **descending order** in the format **"[a1, a2, a3,… an]"**. Check the examples below.

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] |
| ["[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] |

## Garage

Write a function that **stores cars** in garages. You will be given an **array of strings**. Each string will contain a **number of a garage** and **info about a car**. You have to store the car (with its info) in the given garage. The info about the car will be in the format:

**"{key1}: {value1}, {key2}: {value2}…"**

If the garage **does not exist, create it**. The cars will always be **unique.** At the endprint the result in the format:  
**"Garage № {number}:  
--- {carOneKeyOne} - {carOneValueOne}, {carOneKeyTwo} - {carOneValueTwo}…  
--- {the same for the next car}  
Garage № {number}: …"**

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['1 - color: blue, fuel type: diesel', '1 - color: red, manufacture: Audi', '2 - fuel type: petrol', '4 - color: dark blue, fuel type: diesel, manufacture: Fiat'] | Garage № 1  --- color - blue, fuel type - diesel  --- color - red, manufacture - Audi  Garage № 2  --- fuel type - petrol  Garage № 4  --- color - dark blue, fuel type - diesel, manufacture - Fiat |
| ['1 - color: green, fuel type: petrol',  '1 - color: dark red, manufacture: WV',  '2 - fuel type: diesel',  '3 - color: dark blue, fuel type: petrol'] | Garage № 1  --- color - green, fuel type - petrol  --- color - dark red, manufacture - WV  Garage № 2  --- fuel type - diesel  Garage № 3  --- color - dark blue, fuel type - petrol |

## Armies

Write a function that stores information about an army leader and his armies. The input will be an array of strings. The strings can be in some of the following formats:  
**"{leader} arrives"** – add the leader (no army)

**"{leader}: {army name}, {army count}"** – add the army with its count to the leader (if he exists)  
**"{army name} + {army count}"** – if the army exists somewhere add the count  
**"{leader} defeated"** – delete the leader and his army (if he exists)

When finished reading the input sort the **leaders** by **total army count** in **descending**. Then each **army** should be sorted by **count in descending**.

### Output

Print in the following format:  
**"{leader one name}: {total army count}  
>>> {armyOne name} - {army count}  
>>> {armyTwo name} - {army count}  
 …  
{leader two name}: {total army count}  
…"**

### Constrains

* The **new leaders** will always be **unique**
* When **adding a new army** to the leader, the army will be **unique**

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['Rick Burr arrives', 'Fergus: Wexamp, 30245', 'Rick Burr: Juard, 50000', 'Findlay arrives', 'Findlay: Britox, 34540', 'Wexamp + 6000', 'Juard + 1350', 'Britox + 4500', 'Porter arrives', 'Porter: Legion, 55000', 'Legion + 302', 'Rick Burr defeated', 'Porter: Retix, 3205'] | Porter: 58507  >>> Legion - 55302  >>> Retix - 3205  Findlay: 39040  >>> Britox - 39040 |
| ['Rick Burr arrives', 'Findlay arrives', 'Rick Burr: Juard, 1500', 'Wexamp arrives', 'Findlay: Wexamp, 34540', 'Wexamp + 340', 'Wexamp: Britox, 1155', 'Wexamp: Juard, 43423'] | Wexamp: 44578  >>> Juard - 43423  >>> Britox - 1155  Findlay: 34880  >>> Wexamp - 34880  Rick Burr: 1500  >>> Juard - 1500 |

## Comments

Write a function that stores information about users and their comments on a website. You have to store the **users**, the **comments as an object with title and content,** and the **article** that the comment is about. The user can only comment, when he is on the **list of users** and **the article is in the list of articles**. The input comes as an array of strings. The strings will be in the format:  
**"user {username}"** – add the user to the list of users  
**"article {article name}"** – add the article to the article list  
**"{username} posts on {article name}: {comment title}, {comment content}"** –save the info

At the end **sort** the articles by a **count of comments** and print the **users with their comments** ordered by **usernames in ascending**.

### Output

Print the result in the following format:  
**"Comments on {article1 name}  
--- From user {username1}: {comment title} - {comment content}  
--- From user {username2}: …  
Comments on {article2 name}  
…"**

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['user aUser123', 'someUser posts on someArticle: NoTitle, stupidComment', 'article Books', 'article Movies', 'article Shopping', 'user someUser', 'user uSeR4', 'user lastUser', 'uSeR4 posts on Books: I like books, I do really like them', 'uSeR4 posts on Movies: I also like movies, I really do', 'someUser posts on Shopping: title, I go shopping every day', 'someUser posts on Movies: Like, I also like movies very much'] | Comments on Movies  --- From user someUser: Like - I also like movies very much  --- From user uSeR4: I also like movies - I really do  Comments on Books  --- From user uSeR4: I like books - I do really like them  Comments on Shopping  --- From user someUser: title - I go shopping every day |
| ['user Mark', 'Mark posts on someArticle: NoTitle, stupidComment', 'article Bobby', 'article Steven', 'user Liam', 'user Henry', 'Mark posts on Bobby: Is, I do really like them', 'Mark posts on Steven: title, Run', 'someUser posts on Movies: Like'] | Comments on Bobby  --- From user Mark: Is - I do really like them  Comments on Steven  --- From user Mark: title - Run |

## Book Shelf

Write a function that stores information about **shelves** and the **books on the shelves**. Each shelf has an **Id** and a **genre** of books that can be on it. Each book has a **title**, an **author,** and a **genre**. The input comes as an **array of strings**. They will be in the format:  
**"{shelf id} -> {shelf genre}"** – create a shelf **if the id is not taken**.  
**"{book title}: {book author}, {book genre}"** – if a shelf with that **genre exists**, add the book to the shelf.  
After finishing reading input, sort the shelves by a **count of books** in it in **descending**. For each shelf sort the **books by title** in ascending. Then print them in the following format.  
**"{shelfOne id} {shelf genre}: {books count}  
--> {bookOne title}: {bookOne author}  
--> {bookTwo title}: {bookTwo author}  
…  
{shelfTwo id} {shelf genre}: {books count}  
…"**

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['1 -> history', '1 -> action', 'Death in Time: Criss Bell, mystery', '2 -> mystery', '3 -> sci-fi', 'Child of Silver: Bruce Rich, mystery', 'Hurting Secrets: Dustin Bolt, action', 'Future of Dawn: Aiden Rose, sci-fi', 'Lions and Rats: Gabe Roads, history', '2 -> romance', 'Effect of the Void: Shay B, romance', 'Losing Dreams: Gail Starr, sci-fi', 'Name of Earth: Jo Bell, sci-fi', 'Pilots of Stone: Brook Jay, history'] | 3 sci-fi: 3  --> Future of Dawn: Aiden Rose  --> Losing Dreams: Gail Starr  --> Name of Earth: Jo Bell  1 history: 2  --> Lions and Rats: Gabe Roads  --> Pilots of Stone: Brook Jay  2 mystery: 1  --> Child of Silver: Bruce Rich |
| ['1 -> mystery', '2 -> sci-fi',  'Child of Silver: Bruce Rich, mystery',  'Lions and Rats: Gabe Roads, history',  'Effect of the Void: Shay B, romance',  'Losing Dreams: Gail Starr, sci-fi',  'Name of Earth: Jo Bell, sci-fi'] | 2 sci-fi: 2  --> Losing Dreams: Gail Starr  --> Name of Earth: Jo Bell  1 mystery: 1  --> Child of Silver: Bruce Rich |

## SoftUni Students

Write a function that stores the **students** that signed up for different **courses** at SoftUni. For each **course,** you have to **store the name**, the **capacity,** and the **student**s that are in it. For each **student** store the **username, the email, and their credits**. The input will come as an **array of strings**. The strings will be in some of the following formats:  
**"{course name}: {capacity}"** – add the course with that capacity. If the **course exists**, **add** the **capacity** to the existing one  
**"{username}[{credits count}] with email {email} joins {course name}"** – add the student **if the course exists** (each student can be in **multiple courses**) and if there are **places left** (**count of students** are **less than the capacity**)

Finally, you should sort the courses by the **count of students** in **descending**. Each course should have its students sorted by **credits in descending**.

### Output

Print the result in the format:  
**"{course one}: {places left} places left  
--- {credits}: {username one}, {email one}  
…"**

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['JavaBasics: 2', 'user1[25] with email user1@user.com joins C#Basics', 'C#Advanced: 3', 'JSCore: 4', 'user2[30] with email user2@user.com joins C#Basics', 'user13[50] with email user13@user.com joins JSCore', 'user1[25] with email user1@user.com joins JSCore', 'user8[18] with email user8@user.com joins C#Advanced', 'user6[85] with email user6@user.com joins JSCore', 'JSCore: 2', 'user11[3] with email user11@user.com joins JavaBasics', 'user45[105] with email user45@user.com joins JSCore', 'user007[20] with email user007@user.com joins JSCore', 'user700[29] with email user700@user.com joins JSCore', 'user900[88] with email user900@user.com joins JSCore'] | JSCore: 0 places left  --- 105: user45, user45@user.com  --- 85: user6, user6@user.com  --- 50: user13, user13@user.com  --- 29: user700, user700@user.com  --- 25: user1, user1@user.com  --- 20: user007, user007@user.com  JavaBasics: 1 places left  --- 3: user11, user11@user.com  C#Advanced: 2 places left  --- 18: user8, user8@user.com |
| ['JavaBasics: 15',  'user1[26] with email user1@user.com joins JavaBasics',  'user2[36] with email user11@user.com joins JavaBasics',  'JavaBasics: 5',  'C#Advanced: 5',  'user1[26] with email user1@user.com joins C#Advanced',  'user2[36] with email user11@user.com joins C#Advanced',  'user3[6] with email user3@user.com joins C#Advanced',  'C#Advanced: 1',  'JSCore: 8',  'user23[62] with email user23@user.com joins JSCore'] | C#Advanced: 3 places left  --- 36: user2, user11@user.com  --- 26: user1, user1@user.com  --- 6: user3, user3@user.com  JavaBasics: 18 places left  --- 36: user2, user11@user.com  --- 26: user1, user1@user.com  JSCore: 7 places left  --- 62: user23, user23@user.com |