# **More Exercise: Associative Arrays**

Problems for exercise and homework for the "JS Fundamentals" Course @ SoftUni. Submit your solutions in the SoftUni judge system at: https://judge.softuni.org/Contests/1305

## 1. 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 end print 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:	Garage № 1
diesel', '1 - color: red,	color - blue, fuel type - diesel
manufacture: Audi', '2 - fuel type:	color - red, manufacture - Audi
petrol', '4 - color: dark blue, fuel	Garage № 2
type: diesel, manufacture: Fiat']	fuel type - petrol
	Garage № 4
	color - dark blue, fuel type -
	diesel, manufacture - Fiat
['1 - color: green, fuel type:	Garage № 1
petrol',	color - green, fuel type -
'1 - color: dark red, manufacture:	petrol
WV',	color - dark red, manufacture -
'2 - fuel type: diesel',	WV
'3 - color: dark blue, fuel type:	Garage № 2
petrol']	fuel type - diesel
	Garage № 3
	color - dark blue, fuel type -
	petrol

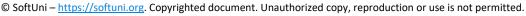
## 2. Armies

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)
```

<sup>&</sup>quot;{leader} defeated" – delete the leader and his army (if he exists)



















<sup>&</sup>quot;{leader}: {army name}, {army count}" – add the army with its count to the leader (if he exists)

<sup>&</sup>quot;{army name} + {army count}" - if the army exists somewhere add the count

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

### 3. 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
```

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

















<sup>&</sup>quot;article {article name}" – add the article to the article list

<sup>&</sup>quot;{username} posts on {article name}: {comment title}, {comment content}" - save the info

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

#### 4. Book Shelf

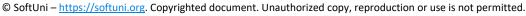
Write a function that stores information about shelves and the books on the shelves. Each shelf has an Id and a genre of books that can be on it. Each book has a title, an author, and a genre. The input comes as an array of strings. They will be in the format:

```
"{shelf id} -> {shelf genre}" - create a shelf if the id is not taken.
```

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}
```

















<sup>&</sup>quot;{book title}: {book author}, {book genre}" – if a shelf with that genre exists, add the book to the shelf.

#### **Example**

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

#### 5. 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 students 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

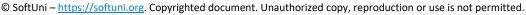
Print the result in the format:

```
"{course one}: {places left} places left
--- {credits}: {username one}, {email one}
```

## **Example**

Input	Output
['JavaBasics: 2', 'user1[25] with	JSCore: 0 places left
email user1@user.com joins	105: user45, user45@user.com
C#Basics', 'C#Advanced: 3', 'JSCore:	85: user6, user6@user.com
4', 'user2[30] with email	50: user13, user13@user.com

















```
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']
['JavaBasics: 15',
'user1[26] with email user1@user.com
joins JavaBasics',
'user2[36] with email
user11@user.com joins JavaBasics',
'JavaBasics: 5',
```

```
--- 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
```

```
'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















