

Extra Credit: Associative Arrays

1. Garage

Write a function that **stores cars** in garages. You will be given an **array of strings**. Each string will contain **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 **"{key}: {value}, {key}: {value}..."**. If the garage **does not exist, create it**. The cars will always be **unique**. At the end print the result **ordered by the numbers of the garages** (no further sorting required) 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

2. Armies

Write a function that stores information about an army leader and his armies. The input will be 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**.

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}

..."

Constraints

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

Example

Input	Output
<pre>['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']</pre>	<pre>Porter: 58507 >>> Legion - 55302 >>> Retix - 3205 Findlay: 39040 >>> Britox - 39040</pre>

3. Comments

Write a function that stores information about users and their comments in a website. You have to store the **users**, the **comments as an object with title and content** and the **article** that 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 array of strings. The strings will be in 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 **count of comments** and print the **users with their comments** ordered by **usernames in ascending**.

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
<pre>['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']</pre>	<pre>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</pre>

4. Book Shelf

Write a function that stores information about **shelves** and the **books in the shelves**. Each shelf has an **Id** and a **genre** of books that can be in it. Each book has a **title**, an **author** and **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 finished reading input, sort the shelves by **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

5. Kingsland Students

Write a function that stores the **students** that signed up for different **courses** at Kingsland. 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**.

Print the result in the format:

"{course one}: {places left} places left

--- {credits}: {username one}, {email one}

..."

Example

Input	Output
<pre>['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']</pre>	<pre>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</pre>