**Exercise: Associative Arrays**

Problems for exercise and homework for the ["JS Fundamentals" Course @ HYPERLINK "https://softuni.bg/trainings/3839/programming-fundamentals-with-javascript-september-2022" HYPERLINK "https://softuni.bg/trainings/3839/programming-fundamentals-with-javascript-september-2022" HYPERLINK "https://softuni.bg/trainings/3839/programming-fundamentals-with-javascript-september-2022" HYPERLINK "https://softuni.bg/trainings/3839/programming-fundamentals-with-javascript-september-2022" HYPERLINK "https://softuni.bg/trainings/3839/programming-fundamentals-with-javascript-september-2022" HYPERLINK "https://softuni.bg/trainings/3839/programming-fundamentals-with-javascript-september-2022" HYPERLINK "https://softuni.bg/trainings/3839/programming-fundamentals-with-javascript-september-2022"SoftUni HYPERLINK "https://softuni.bg/trainings/3839/programming-fundamentals-with-javascript-september-2022" HYPERLINK "https://softuni.bg/trainings/3839/programming-fundamentals-with-javascript-september-2022" HYPERLINK "https://softuni.bg/trainings/3839/programming-fundamentals-with-javascript-september-2022" HYPERLINK "https://softuni.bg/trainings/3839/programming-fundamentals-with-javascript-september-2022" HYPERLINK "https://softuni.bg/trainings/3839/programming-fundamentals-with-javascript-september-2022" HYPERLINK "https://softuni.bg/trainings/3839/programming-fundamentals-with-javascript-september-2022" HYPERLINK "https://softuni.bg/trainings/3839/programming-fundamentals-with-javascript-september-2022".](https://softuni.bg/trainings/3839/programming-fundamentals-with-javascript-september-2022)   
Submit your solutions in the SoftUni judge system at: <https://judge.softuni.org/Contests/1306>

* **Words Tracker**

Write a function that receives an **array of words** and finds **occurrences of given words** in that sentence.

The input will come as an **array of strings**. The **first string** will contain the **words** you will be looking for separated by a **space**. All **strings after that** will be the words in which you will check for a match.

Print for **each word** how many times it **occurs**. The words should be **sorted by count in descending**.

**Example**

|  |  |
| --- | --- |
| **Input** | **Output** |
| [  'this sentence',  'In', 'this', 'sentence', 'you', 'have', 'to', 'count', 'the', 'occurrences', 'of', 'the', 'words', 'this', 'and', 'sentence', 'because', 'this', 'is', 'your', 'task'  ] | this - 3 sentence - 2 |
| [  'is the',  'first', 'sentence', 'Here', 'is', 'another', 'the', 'And', 'finally', 'the', 'the', 'sentence'] | the – 3  is - 1 |

* **Odd Occurrences**

Write a function that extracts the elements of a sentence, if it appears an odd number of times (**case-insensitive**).

The input comes as a **single string**. The words will be **separated by a single space**.

**Example**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 'Java C# Php PHP Java PhP 3 C# 3 1 5 C#' | c# php 1 5 |
| 'Cake IS SWEET is Soft CAKE sweet Food' | soft food |

* **Piccolo**

Write a function that:

* Records a **car number** for every car that enters the **parking lot**
* Removes a **car number** when the car goes out
* Input will be an array of strings in format **[direction, carNumber]**

Print the output with all car numbers which are in the parking lot **sorted in ascending by number.**

If the parking lot is empty, print: **"Parking Lot is Empty"**.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['IN, CA2844AA',  'IN, CA1234TA',  'OUT, CA2844AA',  'IN, CA9999TT',  'IN, CA2866HI',  'OUT, CA1234TA',  'IN, CA2844AA',  'OUT, CA2866HI',  'IN, CA9876HH',  'IN, CA2822UU'] | CA2822UU  CA2844AA  CA9876HH  CA9999TT |
| ['IN, CA2844AA',  'IN, CA1234TA',  'OUT, CA2844AA',  'OUT, CA1234TA'] | Parking Lot is Empty |

* **Party Time**

There is a party at SoftUni. Many guests are invited and they are **two types**: VIP and regular. When guests come to the party check if he/she **contains** in any of the **two reservation lists**.

The input will come as an **array of strings**. You will be given the list with the guests before you receive a command **"PARTY"**.

All **VIP numbers start with a digit.**

When you receive the command **"PARTY",** the guests start coming.

Print the **count** of **guests** then all guests, who didn't come to the party (VIP must be printed first).

Hint: Guest names are not unique. Only the first match is removed when receiving a name.

**Examples**

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Output** | **Input** | **Output** |
| ['7IK9Yo0h',  '9NoBUajQ',  'Ce8vwPmE',  'SVQXQCbc',  'tSzE5t0p',  'PARTY',  '9NoBUajQ',  'Ce8vwPmE',  'SVQXQCbc'  ] | 2  7IK9Yo0h  tSzE5t0p | ['m8rfQBvl',  'fc1oZCE0',  'UgffRkOn',  '7ugX7bm0',  '9CQBGUeJ',  '2FQZT3uC',  'dziNz78I',  'mdSGyQCJ',  'LjcVpmDL',  'fPXNHpm1',  'HTTbwRmM',  'B5yTkMQi',  '8N0FThqG',  'xys2FYzn',  'MDzcM9ZK',  'PARTY',  '2FQZT3uC',  'dziNz78I',  'mdSGyQCJ',  'LjcVpmDL',  'fPXNHpm1',  'HTTbwRmM',  'B5yTkMQi',  '8N0FThqG',  'm8rfQBvl',  'fc1oZCE0',  'UgffRkOn',  '7ugX7bm0',  '9CQBGUeJ'  ] | 2  xys2FYzn  MDzcM9ZK |

* **Card Game**

You are given a sequence of people and for every person what cards he draws from the deck. The input will be an **array of strings**. Each string will be in the format:

**{personName}: {PT, PT, PT,… PT}**

Where **P** (2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A) is the power of the card and **T** (S, H, D, C) is the type. The name can contain any ASCII symbol except **':'**. The input will always be valid and in the format described, there is no need to check it.

A single person **cannot have more than one** card with the same power and type, if he draws such a card he discards it. The people are playing with **multiple decks**. Each card has a value that is calculated by the power multiplied by the type. Powers **2 to 10** have the same value and **J to A** are **11 to 14**. Types are mapped to multipliers the following way (**S -> 4, H-> 3, D -> 2, C -> 1**).

Finally print out the total value each player has in his hand in the format:

**{personName}: {value}**

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| [  'Peter: 2C, 4H, 9H, AS, QS',  'Tomas: 3H, 10S, JC, KD, 5S, 10S',  'Andrea: QH, QC, QS, QD',  'Tomas: 6H, 7S, KC, KD, 5S, 10C',  'Andrea: QH, QC, JS, JD, JC',  'Peter: JD, JD, JD, JD, JD, JD'  ] | Peter: 167  Tomas: 175  Andrea: 197 |
| [  'John: 2C, 4H, 9H, AS, QS',  'Slav: 3H, 10S, JC, KD, 5S, 10S',  'Alex: 6H, 7S, KC, KD, 5S, 10C',  'Thomas: QH, QC, JS, JD, JC',  'Slav: 6H, 7S, KC, KD, 5S, 10C',  'Thomas: QH, QC, JS, JD, JC',  'Alex: 6H, 7S, KC, KD, 5S, 10C',  'Thomas: QH, QC, JS, JD, JC',  'John: JD, JD, JD, JD'  ] | John: 167  Slav: 175  Alex: 115  Thomas: 125 |

* **Company Users**

Write a function, which keeps the information about companies and their employees.

You will receive an **array of strings** containing the **company** **name** and **employee's id.** Add each employee to the given company. Keep in mind that a company **cannot have two employees with the same id**.

When you finish reading data, **order the companies by their name in ascending order**.

Print the company name and each employee's id in the following format:

**{companyName}**

**-- {id1}**

**-- {id2}**

**-- {idN}**

**Input / Constraints**

* The input come **as array of strings**, each in the format: "**{companyName} -> {employeeId}**".
* The input always will be valid.

**Examples**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| [  'SoftUni -> AA12345',  'SoftUni -> BB12345',  'Microsoft -> CC12345',  'HP -> BB12345'  ] | HP  -- BB12345  Microsoft  -- CC12345  SoftUni  -- AA12345  -- BB12345 |  | [  'SoftUni -> AA12345',  'SoftUni -> CC12344',  'Lenovo -> XX23456',  'SoftUni -> AA12345',  'Movement -> DD11111'  ] | Lenovo  -- XX23456  Movement  -- DD11111  SoftUni  -- AA12345  -- CC12344 |

* **A Miner Task**

You are given an **array of strings**. Every **odd string** is representing a **resource** (e.g. Gold, Silver, Copper, and so on), and **every even** – **quantity**. Your task is to collect the resources and print them each on a new line.

**Print the resources and their quantities in the format:**

**{resource} –> {quantity}**

The quantities inputs will be in the range [1 … 2 000 000 000].

**Examples**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| [  'Gold',  '155',  'Silver',  '10',  'Copper',  '17'  ] | Gold -> 155  Silver -> 10  Copper -> 17 |  | [  'gold',  '155',  'silver',  '10',  'copper',  '17',  'gold',  '15'  ] | gold -> 170  silver -> 10  copper -> 17 |

* **\*Travel Time**

Write a function that **collects** and **orders** information about travel destinations.

As **input,** you will receive an **array of strings**.

Each string will consist of the following information with the format:

**"Country name > Town name > Travel cost"**

The **Country name** will be a string, the **Town name** will be a unique string, **Travel cost** will be a number.

If you receive **the same** **Town name** twice, you should keep the **cheapest** offer. **Have in mind** that one Country may have **several** Towns to visit.

After you finish the organizational part, you need to let Steven know which destination point to visit first. The order will be as follows: First sort Country names **alphabetically** and then sort by **lowest** Travel cost.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| [  "Bulgaria > Sofia > 500",  "Bulgaria > Sopot > 800",  "France > Paris > 2000",  "Albania > Tirana > 1000",  "Bulgaria > Sofia > 200"  ] | Albania -> Tirana -> 1000  Bulgaria -> Sofia -> 200 Sopot -> 800  France -> Paris -> 2000 |
| [  'Bulgaria > Sofia > 25000',  'Bulgaria > Sofia > 25000',  'Kalimdor > Orgrimar > 25000',  'Albania > Tirana > 25000',  'Bulgaria > Varna > 25010',  'Bulgaria > Lukovit > 10'  ] | Albania -> Tirana -> 25000  Bulgaria -> Lukovit -> 10 Sofia -> 25000 Varna -> 25010  Kalimdor -> Orgrimar -> 25000 |

* **\*Arena Tier**

Pesho is a pro gladiator, he is struggling to become master of the Arena.

You will receive **several input lines** in one of the following formats:

**"{gladiator} -> {technique} -> {skill}"**

**"{gladiator} vs {gladiator}"**

The **gladiator and technique** are strings, the given **skill** will be an integer number. You need to keep track of **every gladiator**.

When you receive a **gladiator and his technique and skill**, add him to the gladiator pool, if he isn't present, else add his technique or update his skill, only if the **current** technical skill is **lower** than the new value. (Max)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

If you receive **"{gladiator} vs {gladiator}"** and both gladiators exist in the tier, they duel with the following rules:

Compare their techniques, if they got at **least one in common**, the gladiator with better **total skill** points **wins** and the other is demoted from the tier -> remove him.

If they don't have techniques in common, the duel isn't happening and both continue in the Season.

You should end your program when you receive the command **"Ave Cesar"**. At that point, you should print the gladiators, **ordered by total skill in descending order, then ordered by name in ascending order**. Foreach gladiator prints their technique and skill **ordered descending, then ordered by technique name in ascending order.**

**Input / Constraints**

You will receive an **array of strings** as a parameter to your solution.

* The input comes in the form of commands in one of the formats specified above.
* Gladiator and technique **will always be one-word string, containing no whitespaces**.
* Skill will be an **integer** in the **range [0, 1000]**.
* There will be **no invalid** input lines.
* The program ends when you receive the command **"Ave Cesar"**.

**Output**

* The output format for each gladiator is:

**"{gladiator}: {totalSkill} skill"**

**"- {technique} <!> {skill}"**

***Scroll down to see examples.***

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| [  'Peter -> BattleCry -> 400',  'Alex -> PowerPunch -> 300',  'Stefan -> Duck -> 200',  'Stefan -> Tiger -> 250',  'Ave Cesar'  ] | Stefan: 450 skill  - Tiger <!> 250  - Duck <!> 200  Peter: 400 skill  - BattleCry <!> 400  Alex: 300 skill  - PowerPunch <!> 300 | We order the gladiators by total skill points descending, then by name. We print every technique along its skill ordered descending by skill, then by technique name. |
| **Input** | **Output** |  |
| [  'Peter -> Duck -> 400',  'Julius -> Shield -> 150',  'Gladius -> Heal -> 200',  'Gladius -> Support -> 250',  'Gladius -> Shield -> 250',  'Peter vs Gladius',  'Gladius vs Julius',  'Gladius vs Maximilian',  'Ave Cesar'  ] | Gladius: 700 skill  - Shield <!> 250  - Support <!> 250  - Heal <!> 200  Peter: 400 skill  - Duck <!> 400 | Gladius and Peter don't have a common technique, so the duel isn't valid.  Gladius wins vs Julius /common technique: "Shield". Julius is demoted.  Maximilian doesn't exist so the duel isn't valid.  We print every gladiator left in the tier. |

* **10. \*Legendary Farming**

You’ve beaten all the content and the last thing left to accomplish is to own a legendary item. However, it’s a tedious process and requires quite a bit of farming. Anyway, you are not too pretentious – any legendary will do. The possible items are:

* **"Shadowmourne"** – requires **250 Shards**
* **"Valanyr"** – requires **250 Fragments**
* **"Dragonwrath"** – requires **250 Motes**

**"Shards", "Fragments",** and **"Motes"** are the **key materials**, all else is **junk.**

You will be given lines of input in the format:

**"{quantity1} {material1} {quantity2} {material2} … {quantityN} {materialN}"**

Keep track of the **key materials -** the **first** that reaches the **250 mark** **wins** the **race**. At that point, print the corresponding legendary obtained.

Then, print the **remaining** shards, fragments, motes, ordered by **quantity** in **descending** order, then by **name** in **ascending** order, each on a new line. Finally, print the collected **junk** items, in **alphabetical** order.

**Input**

* Each line comes in the following format:

**{quantity1} {material1} {quantity2} {material2} … {quantityN} {materialN}**

**Output**

* On the **first line**, print the obtained item in format: "**{Legendary item} obtained!**"
* On the next **three lines**, print the remaining key materials in descending order by quantity
* If two key materials have the same quantity, print them in alphabetical order
* On the final **several lines**, print the junk items in alphabetical order
* All materials are printed in format **"{material}: {quantity}"**
* All output should be **lowercase**, except the first letter of the legendary

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| '3 Motes 5 stones 5 Shards 6 leathers 255 fragments 7 Shards' | Valanyr obtained!  fragments: 5  shards: 5  motes: 3  leathers: 6  stones: 5 |

|  |  |
| --- | --- |
| **Input** | **Output** |
| '123 silver 6 shards 8 shards 5 motes 9 fangs 75 motes 103 MOTES 8 Shards 86 Motes 7 stones 19 silver' | Dragonwrath obtained!  shards: 22  motes: 19  fragments: 0  fangs: 9  silver: 123 |