# Problem 1 – Travel Plans

Write a JavaScript program that calculates the **total amount** of **gold** Mariyka managed to collect. Mariyka is **specialized** in some professions, **average** in others and **clumsy** at yet others. Your program will receive an **array with professions and the amount of gold** for each activity. Mariyka is so good at her **Specialized** professions that **every second** customer of Specialized profession gives her **additional gold of cost: 200 gold right after she spend the gold mentioned below for candies**. She’s as well so bad at her **Clumsy** professions that every **second** customer from the Clumsy professions gives her **5% less** of the first given amount and **every third – 10%.**

Also, Mariyka is **spending 20% of every** activity she does that she’s **specialized** in, of course she spends them for candies.

Have in mind that Mariyka **does not** accept to work for **less** than **200 gold** in her **specialized** professions.

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

Each string will consist of the following information with format:

“Professions : Gold offered”

Professions will be as follows:

Specialized:  
Programming, Hardware maintenance, Cooking, Translating, Designing.  
Average:  
Driving, Managing, Fishing, Gardening.

Clumsy:  
Singing, Accounting, Teaching, Exam-Making, Acting, Writing, Lecturing, Modeling, Nursing.

As output you need to print the total cash Mariyka has made.

If the amount is less than 1000 gold , she need to collect more gold until she get enough. Else she will be able to move to the next task and start planning her travel!

### Input

You will receive **one argument–** an **array strings** as shown above.

### Output

Print on the **console** the total amount of **gold** and if Mariyka has collected **enough or not**.

The output should be **formatted** to the second delimiter.

### Constraints

* The **number** of **elements** in the **input argument** will be in range **[1..100] inclusive**
* There **will** be **no invalid** **input**
* There **will** be **no negative Gold**

### Examples

|  |
| --- |
| **Input** |
| ["Programming : 500", "Driving : 243", "Singing : 100", "Cooking : 199"] |
| **Output** |
| Final sum: 743.00  Mariyka need to earn 257.00 gold more to continue in the next task. |

|  |
| --- |
| **Input** |
| ["Programming : 500", "Driving : 243.55", "Acting : 200", "Singing : 100", "Cooking : 199’, "Hardware maintenance : 800", "Gardening : 700", "Programming : 500"] |
| **Output** |
| Final sum: 2878.55  Mariyka earned 1878.55 gold more. |

# Problem 2 – Travel Investigation

Write a JavaScript program that finds **specific unique companies** in a text. We need to **filter** our text so that we **separate** the **valid** and **invalid** sentences. Our **companies** will be separated by **exact delimiter**. The **rest of the input** will be **strings** which will be the text we will have to filter.

As **input** we will receive **array of strings**. On the **first** index of our array we always will be given the **companies** which will be a string  **separated by** the given on the **second** index **delimiter,** **followed by the sentences** we need to process.

The companies will always be given to lower.

Each string will consist of the following information with format:

[ “company1@, company2@, company3”,

“@, “,

“some text company1 more text company2 some more text company3”]

As output you need to print the valid sentences in format – “ValidSentences” followed by the numbered sentences each on a new line, a separator between the valid and invalid sentences in this case 30x”=“, followed by the “InvalidSentences” each on a new line and numbered.

Have in mind that there might be case of all Valid or all Invalid sentences.

We consider Valid, a sentence which has all of the companies.

### Input

You will receive **one argument –** an **array strings** as shown above.

### Output

Print on the **console** the **Valid** sentences if you found any, the delimiter shown above and the **Invalid** sentences if you found any.

### Constraints

* The **number** of **sentences** in the **input argument** will be in range **[1..15] inclusive**
* There **will** be **no invalid** **input**
* There **will** be **no repeated companies**

### Examples

|  |
| --- |
| **Input** |
| ["bulgariatour@, minkatrans@, koftipochivkaltd",  "@,",  "Mincho e KoftiPochivkaLTD Tip 123 ve MinkaTrans BulgariaTour",  "dqdo mraz some text but is KoftiPochivkaLTD MinkaTrans",  "someone continues as no "] |
| **Output** |
| ValidSentences  1.mincho e koftipochivkaltd tip 123 ve minkatrans bulgariaatour  ==============================  InvalidSentences  1. dqdo mraz some text but is koftipochivkaltd minkatrans  2. someone continues as no |

|  |
| --- |
| **Input** |
| ["bulgariatour@, minkatrans@, koftipochivkaltd",  "@,",  "Mincho e KoftiPockivkaLTD Tip 123 ve MinkaTrans BulgariaTour",  "We will koftipochivkaLTD travel e expenses no MinkaTrans mu e BulgariaTour",  "dqdo BuLGariaTOUR mraz some text but is KoftiPochivkaLTD minkaTRANS"] |
| **Output** |
| ValidSentences  1. mincho e koftipochivkaltd tip 123 ve minkatrans bulgariatour  2. we will koftipochivkaltd traavel e expenses no minkatrans mu e bulgariatour  3. dqdo bulgariatour mraz some text but is koftipochivkaltd minkatrans |

# Problem 3 – Minke, decode

You will recieve a **string** with the еncrypted message which contains **the encrypted country** and the code that need to be decrypted . The **country** will always **starts** with capital letter and **end** with capital letter. You’ll recieve char **starting point** and char **end points** with which you will decode the country.

Example for country is MolgarA which needs to become MoldovA

For **example** in the input we will recieve **char start point** **-3** which is the letter **g** and **char end point** **- 5** which is **r**. We need to **replace** everything from g to r with “dov” which will be given in the input aswell.

The final result must be **Moldova** (the last char needs to be lowered).

The second part is to **decode** **the numbers** from the **input string**. To extract the numbers from the input string you need to **match every three-digit number** (**whole** or **fractional**).

If the number is **fractional** you have to round it up. When you receive all maches they need to be decoded from **ascii to text** and print it next to the country **capitalized** and **separated by “ =>** “.

## Input

* The input will consist of **array of strings, containing** 4 strings:
* First is char **start point** for the country decoding:
* Second is char **end point** for the country decoding:
* Third is the **right word**:
* Fourth is the **encrypted text**:

## Constraints

* There will be always only one country in the input.
* The separator for the fractional num will always be dot
* All inputs in the array will be string

## Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| ["3", "5", "gar","114 sDfia 1, riDl10 confin$4%#ed117 likewise it humanity aTe114.223432 BultoriA - Varnd railLery101 an unpacked as he"] | Bulgaria => Ruse |
| ["1", "4","loveni", "SerbiA 67 – sDf1d17ia aTe 1, 108 confin$4%#ed likewise it humanity Bulg35aria - VarnA railLery1a0s1 111 an unpacked as 109 he"] | Slovenia => Lom |

# Problem 4 – Travel Time

Write a JavaScript program that **collects** and **orders** information about traveling destinations.

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

Each string will consist of the following information with format:

“Country name > Town name > Travel cost”

Country name will be unique string, Town name will also be 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.

If you receive Town name starting with lower case letter ,you should make it to upper, as shown in the example above.

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

### Input

You will receive **one argument–** an **array strings** as shown above.

### Output

Print on the **console** the information you’ve collected, **sorted** by the given rules, **formatted** as seen in the examples.

### Constraints

* The **number** of **elements** in the **input argument** will be in range **[1..100] inclusive**
* There **will** be **no invalid** **input**
* There **will** be **no negative Travel cost**

### Examples

|  |
| --- |
| **Input** |
| ["Bulgaria > Sofia > 500",  "Bulgaria > Sopot > 800",  "France > Paris > 2000",  "Albania > Tirana > 1000",  "Bulgaria > Sofia > 200" ] |
| **Output** |
| Albania -> Tirana -> 1000  Bulgaria -> Sofia -> 200 Sopot -> 800  France -> Paris -> 2000 |