# Lab: Strings and Regular Expressions

Problems for in-class lab for the [“JavaScript Fundamentals” course @ SoftUni](https://softuni.bg/courses/javascript-fundamentals). Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/312>.

## Print Letters

Write a JS function that prints all the symbols of a string, each on a new line.

The **input** comes as a single string argument.

The **output** is printed on the console, each letter on a new line.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 'Hello, World!' | str[0] -> H  str[1] -> e  str[2] -> l  str[3] -> l  str[4] -> o  str[5] -> ,  str[6] ->  str[7] -> W  str[8] -> o  str[9] -> r  str[10] -> l  str[11] -> d  str[12] -> ! |
| 'SoftUni' | str[0] -> S  str[1] -> o  str[2] -> f  str[3] -> t  str[4] -> U  str[5] -> n  str[6] -> i |

## Concatenate Reversed

Write a JS function that reverses a series of strings and prints them concatenated from last to first.

The **input** comes as an array of strings.

The **output** is printed on the console. Print all strings concatenated on a single line, starting from the last input string, going to the first. Reverse each individual string’s letters.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['I', 'am', 'student'] | tnedutsmaI |
| ['race', 'car'] | racecar |

## Count Occurrences

Write a JS function that counts how many times a string occurs in a given text. Overlapping strings are allowed.

The **input** comes as two string arguments. The first element is the target string and the second element is the text in which to search for occurrences.

The **output** should be a number, printed on the console.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 'the', 'The quick brown fox jumps over the lay dog.' | 1 |
| 'ma', 'Marine mammal training is the training and caring for marine life such as, dolphins, killer whales, sea lions, walruses, and other marine mammals. It is also a duty of the trainer to do mental and physical exercises to keep the animal healthy and happy.' | 7 |

## Extract Text

You will be given a text as a string. Write a JS function that extracts and prints only the text that’s surrounded by parentheses.

The **input** comes as a single string argument.

The **output** is printed on the console on a single line, in the form of a comma-separated list.

### Examples

|  |
| --- |
| **Input** |
| 'Rakiya (Bulgarian brandy) is self-made liquor (alcoholic drink)' |
| **Output** |
| Bulgarian brandy, alcoholic drink |

## Aggregate Table

You will be given a list of towns and incomes for each town, formatted in a table, separated by pipes (|). Write a JS function that extracts the names of all towns and produces a sum of the incomes. Note that splitting may result in empty string elements and the number of spaces may be different in every table.

The **input** comes as array of string elements. Each element is one row in a formatted table.

The **output** is printed on the console on two lines. On the first line, print a comma-separated list of all towns and on the second, the sum of all incomes.

### Examples

|  |
| --- |
| **Input** |
| ['| Sofia | 300',  '| Veliko Tarnovo | 500',  '| Yambol | 275'] |
| **Output** |
| Sofia, Veliko Tarnovo, Yambol  1075 |

## Restaurant Bill

You are tasked to write a JS function that receives an array of purchases and their prices and prints all your purchases and their total sum.

The **input** comes as an array of string elements – the elements on even indexes (0, 2, 4…) are the product names, while the elements on odd indexes (1, 3, 5…) are the corresponding prices.

The **output** should be printed on the console - a single sentence containing all products and their total sum in the format “**You purchased {all products separated by comma + space} for a total sum of {total sum of products}**”.

### Examples

|  |
| --- |
| **Input** |
| ['Beer Zagorka', '2.65', 'Tripe soup', '7.80','Lasagna', '5.69'] |
| **Output** |
| You purchased Beer Zagorka, Tripe soup, Lasagna for a total sum of 16.14 |

|  |
| --- |
| **Input** |
| ['Cola', '1.35', 'Pancakes', '2.88'] |
| **Output** |
| You purchased Cola, Pancakes for a total sum of 4.23 |

## Usernames

Write a JS function that parses a list of emails and returns a list of usernames, generated from them. Each username is composed from the name of the email address, a period and the first letter of every element in the domain name. See the examples for more information.

The **input** comes as array of string elements. Each element is an email address.

The **output** is printed on the console on a single line as a comma-formatted list.

### Examples

|  |
| --- |
| **Input** |
| ['peshoo@gmail.com', 'todor\_43@mail.dir.bg', 'foo@bar.com'] |
| **Output** |
| peshoo.gc, todor\_43.mdb, foo.bc |

## Censorship

The thought police are at it again and they need your help! Write a JS function that would **censor news articles**. You will be given a text and then a list of strings that need to be blacked out from the text. Replace all occurrences of the strings with dashes of the same length as the string. The strings will **not overlap**, so order of processing is not important. See the examples for more information.

The **input** comes as two arguments – one string and one array of strings. The first element is the text to scan and the array contains the strings to be censored.

The **output** is the return value of your functions. Save the censored results in a string and return it.

### Examples

|  |
| --- |
| **Input** |
| 'roses are red, violets are blue', ['**, violets are**', '**red**'] |
| **Output** |
| roses are ---------------- blue |

|  |
| --- |
| **Input** |
| 'David Ruben Piqtoukun (born 1950) is an Inuit artist from Paulatuk, Northwest Territories. His output includes sculpture and prints; the sculptural work is innovative in its use of mixed media. His materials and imagery bring together modern and traditional Inuit stylistic elements in a personal vision. An example of this is his work "The Passage of Time" (1999), which portrays a shaman in the form of a salmon moving through a hole in a hand. While shamanic imagery is common in much of Inuit art, the hand in this work is sheet metal, not a traditional material such as walrus ivory, caribou antler or soapstone. Ruben\'s brother, Abraham Apakark Anghik Ruben, is also a sculptor. Fellow Inuit artist Floyd Kuptana learned sculpting techniques as an apprentice to David Ruben.', ['**Inuit**'] |
| **Output** |
| David Ruben Piqtoukun (born 1950) is an ----- artist from Paulatuk, Northwest Territories. His output includes sculpture and prints; the sculptural work is innovative in its use of mixed media. His materials and imagery bring together modern and traditional ----- stylistic elements in a personal vision. An example of this is his work "The Passage of Time" (1999), which portrays a shaman in the form of a salmon moving through a hole in a hand. While shamanic imagery is common in much of ----- art, the hand in this work is sheet metal, not a traditional material such as walrus ivory, caribou antler or soapstone. Ruben's brother, Abraham Apakark Anghik Ruben, is also a sculptor. Fellow ----- artist Floyd Kuptana learned sculpting techniques as an apprentice to David Ruben. |

## Escaping

You will be given a list of strings, containing user-submitted data. Write a JS function that prints an HTML list from the data. The strings, however, may contain special HTML characters, which is an oft-used method for injection attacks. To prevent unwanted behavior or harmful content, all special characters need to be replaced with their encoded counterparts – they will look the same to the user, but will not pose a security risk. Use the following table to compose your function:

|  |  |
| --- | --- |
| **Raw** | **Encoded** |
| < | &lt; |
| > | &gt; |
| & | &amp; |
| " | &quot; |

Use the provided HTML template to visually test your code – if you don’t escape the control characters, formatted HTML will show up. Don’t care how the HTML template works. Your job is to write the JS escaping function only.

The **input** comes as array of string elements.

The **output** is the return value of your function. Compose the list in a string and return it. See the examples for formatting details.

|  |
| --- |
| **HTML** |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Escaping</**title**> </**head**> <**body**> <**div**><**label for="userInput"**>Paste test input here:</**label**></**div**> <**div**>  <**textarea rows="12" cols="40" id="userInput"**></**textarea**>  <**input type="button" value="Escape"  onclick="document**.getElementById(**'result'**).**innerHTML** = html*Escape*(**JSON**.parse(**document**.getElementById(**'userInput'**).**value**.replace(/'/g, String.fromCharCode(34))));**"**/> </**div**> <**div**><**label for="result"**>Results will show up here:</**label**></**div**> <**div id="result"**></**div**> <**script**>  **function html***Escape*(input) {  *//* ***TODO*** } </**script**> </**body**> </**html**> |

### Examples

|  |
| --- |
| **Input** |
| ['<b>unescaped text</b>', 'normal text'] |
| **Output** |
| <ul>  <li>&lt;b&gt;unescaped text&lt;/b&gt;</li>  <li>normal text</li>  </ul> |

|  |
| --- |
| **Input** |
| ['<div style=\"color: red;\">Hello, Red!</div>', '<table><tr><td>Cell 1</td><td>Cell 2</td><tr>'] |
| **Output** |
| <ul>  <li>&lt;div style=\&quot;color: red;\&quot;&gt;Hello, Red!&lt;/div&gt;</li>  <li>&lt;table&gt;&lt;tr&gt;&lt;td&gt;Cell 1&lt;/td&gt;&lt;td&gt;Cell 2&lt;/td&gt;&lt;tr&gt;</li>  </ul> |

## Match All Words

Write a JS function that matches all words in a text, a word is anything that consists of letters, numbers or underscores (\_).

The **input** comes as single string argument – the text from which to extract the words.

The **output** should be printed on the console and should consist of all words concatenated with a **“|“**(pipe), check the examples bellow to better understand the format.

### Examples

|  |
| --- |
| **Input** |
| 'A Regular Expression needs to have the global flag in order to match all occurrences in the text' |
| **Output** |
| A|Regular|Expression|needs|to|have|the|global|flag|in|order|to|match|all|occurrences|in|the|text |

|  |
| --- |
| **Input** |
| '\_(Underscores) are also word characters' |
| **Output** |
| \_|Underscores|are|also|word|characters |

### Hints

* Read about the special characters in Regular Expressions at MDN to find some that can ease your task <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions>

## Simple Email Validation

Write a JS function that validates simple emails. The emails should have a **username**, which consists only of **English alphabet letters** and **digits**, **a “@” sign**, and a domain name after it. The domain should consist **only of** **2 strings** **separated** by a **single dot**. The 2 strings should contain **NOTHING** but **lowercase English alphabet letters**.

The **input** comes as single string argument which is an email.

The **output** should be printed on the console. If the given email is valid, print “Valid”, if it is not, print “Invalid”.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| valid@email.bg | Valid |
| invalid@emai1.bg | Invalid |

## \*Expression Split

Write a JS function that splits a passed in JS code into separate parts. The passed in code will always have one or more spaces between operators and operands. Normal brackets (**‘(‘**,**’)’**), commas (**,**), semicolons (**;**) and the member access operator (**‘.’(dot)**, as in “console**.**log”) should also be used for splitting. String literals will always be initialized with double quotes (") and will **contain only letters**. Make sure there are no empty entries in the output.

The **input** comes as a single string argument - the JS code that has to be split.

The **output** should be printed on the console, with each elements obtained from the split is printed on a new line.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 'let sum = 4 \* 4,b = "wow";' | let  sum  =  4  \*  4  let  b  =  "wow" |
| 'let sum = 1 + 2;if(sum > 2){\tconsole.log(sum);}' | let  sum  =  1  +  2  if  sum  >  2  {  console  log  sum  } |

## Match the Dates

Write a JS function that finds and extracts all the dates in the given sentences. The dates should be in format   
**d-MMM-yyyy**. **Example: 12-Jun-1999**, **3-Dec-2017**.

The **input** comes as an array of strings. Each string represents a sentence.

The **output** should be printed on the console. The output should consist of all extracted **VALID** dates. Each element should be printed on a new line.

### Examples

|  |
| --- |
| **Input** |
| ['I am born on 30-Dec-1994.',  'This is not date: 512-Jan-1996.',  'My father is born on the 29-Jul-1955.'] |
| **Output** |
| 30-Dec-1994 (Day: 30, Month: Dec, Year: 1994)  29-Jul-1955 (Day: 29, Month: Jul, Year: 1955) |

|  |
| --- |
| **Input** |
| 1-Jan-1999 is a valid date. So is 01-July-2000. I am an awful liar, by the way – Ivo, 28-Sep-2016. |
| **Output** |
| 1-Jan-1999 (Day: 1, Month: Jan, Year: 1999)  28-Sep-2016 (Day: 28, Month: Sep, Year: 2016) |

## Parse the Employee Data

Write a JS function that **validates employee data**, and stores it **if it is** **valid**. The employee data consists of 3 elements – **employee name**, **employee salary** and **employee position**.

The **input** comes as an array of strings. Each element represents input employee data. You should capture only the valid from them. The input will have the following format:

{employeeName} - {employeeSalary} - {employeePosition}

The **Employee name** will be a **string**, which can contain only **English alphabet letters** and must **start with a capital**. The **Employee salary** should be a **VALID** **number**. The **employee position** can contain **English alphabet letters**, **digits, dashes**, **and can consist of several words**. Any input that **does NOT follow** the specified above rules, is to be treated as **invalid,** and is to **be ignored**.

The **output** should be printed on the console. For every **valid employee data** found, you should print each of its elements. Check the examples.

### Examples

|  |
| --- |
| **Input** |
| ['Isacc - 1000 - CEO',  'Ivan - 500 - Employee',  'Peter - 500 - Employee'] |
| **Output** |
| Name: Isacc  Position: CEO  Salary: 1000  Name: Ivan  Position: Employee  Salary: 500  Name: Peter  Position: Employee  Salary: 500 |

|  |
| --- |
| **Input** |
| Jonathan - 2000 - Manager  Peter- 1000- Chuck  George - 1000 - Team Leader |
| **Output** |
| Name: Jonathan  Position: Manager  Salary: 2000  Name: George  Position: Team Leader  Salary: 1000 |

### Hints

* Use **Groups** for this problem, it would be a lot easier.

## Form Filler

Write a JS function that automatically fills a form for a lazy client. The client will give you **3 elements** of **data** about himself – his **username**, his **email**, and his **phone number**. After those 3 elements you will be given the form, as text, with several placeholders in it. You must replace each **valid placeholder** with its corresponding value. The placeholders have special symbols and can **contain only English alphabet letters**. There are **3 types** of valid placeholders:

* **<!{letters}!>** - put the given username in place of this
* **<@{letters}@>** - put the given email in place of this
* **<+{letters}+>** - put the given phone number in place of this

The **input** comes as four string arguments and an array of strings. The **first 3 arguments** will represent the **username**, the **email** and the **phone number**. Each element of the array will represent a sentence, if you find a placeholder somewhere in those sentences you should replace it.

The **output** should be printed on the console. The output should consist of all sentences, printed again, this time with their placeholders replaced with the actual data.

### Examples

|  |
| --- |
| **Input** |
| 'Pesho', 'pesho@softuni.bg', '90-60-90', ['Hello, <!username!>!',  'Welcome to your Personal profile.',  'Here you can modify your profile freely.',  'Your current username is: <!fdsfs!>. Would you like to change that? (Y/N)',  'Your current email is: <@DasEmail@>. Would you like to change that? (Y/N)',  'Your current phone number is: <+number+>. Would you like to change that? (Y/N)'] |
| **Output** |
| Hello, Pesho!  Welcome to your Personal profile.  Here you can modify your profile freely.  Your current username is: Pesho. Would you like to change that? (Y/N) Your current email is: pesho@softuni.bg. Would you like to change that? (Y/N) Your current phone number is: 90-60-90. Would you like to change that? (Y/N) |

## \*Match Multiplication

You are given a text with **numbers** multiplied by **\*** in format **{num1} \* {num2}**. Your job is to extract each two numbers in the above format, multiply them and replace them with their product. The **first number** is integer, can be negative. The **second number** is integer or floating-point and can be negative. There could be whitespace around the “**\***” symbol.

The **input** comes as a single string argument – the text holding the numbers.

The **output** should be printed on the console – it consists of the same text with the multiplied numbers replaced by their product.

### Examples

|  |
| --- |
| **Input** |
| My bill: **2\*2.50** (beer); **2\* 1.20** (kepab); **-2 \* 0.5** (deposit). |
| **Output** |
| My bill: **5** (beer); **2.4** (kepab); **-1** (deposit). |
| Input |

### Hint

* Match the numbers to be multiplied by regex with groups. Check the overloads for the **String.replace** function, there may be an overload with a **callback** that can help you.

## Split a String with a Delimiter

Write a JS function that **splits** a **string** with a given **delimiter**.

The **input** comes as 2 string arguments. The **first one is the string** you need to split and the **second** **one is the delimiter**.

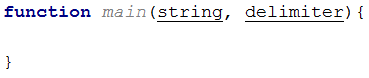
The **output** should consist of all elements you’ve received, after you’ve **split** **the given string** by **the given delimiter**. Each element should be printed on a new line.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| One-Two-Three-Four-Five  - | One  Two  Three  Four  Five |  | http://platform.softuni.bg  . | http://platform  softuni  bg |

### Hints

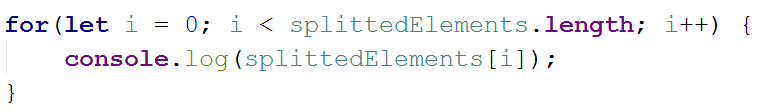
* This “main” function will hold all of the code of our solution.



* Now that we have the string and the delimiter, we can split the string and save the split elements we received as result to the action we did.



* The split() function returns an array of elements, which we can iterate over. The last thing we need to do is print each of the elements on a new line.



## Repeat a String N Times

Write a JS function that repeats a given string, N times.

The **input** comes as 2 arguments. The **first argument is a string that** will represent **the one you need to repeat**. The second one is a number will represent **the times you need to repeat it**.

The **output** is a big string, containing the **given one**, **repeated N times**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| repeat  5 | repeatrepeatrepeatrepeatrepeat |
|  | |
| **Input** | **Output** |
| magic is real  3 | magic is realmagic is realmagic is real |

### Hints

* You can easily use **string concatenation** to solve this problem.
* You could, however, see if there is a **built-in function** that does the same thing.

## Check if String starts with a given Substring.

Write a JS function that checks if a **given string**, **starts with a given substring**.

The **input** comes as 2 string arguments. The **first string** will represent **the main one**. The second one will represent **the substring**.

The **output** is either “true” or “false” based on the result of the check.  
The comparison is **case-sensitive**!

### Examples

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Input** | | **Output** |  | **Input** | | **Output** |
| How have you been?  how | | false |  | The quick brown fox…  The quick brown fox… | | true |
|  | |  |  |  | |  |
| **Input** | | | | **Output** |
| Marketing Fundamentals, starting 19/10/2016  Marketing Fundamentals, sta | | | | true |

## Check if String ends with given Substring.

Write a JS function that checks if a **given string**, **ends with a given substring**.

The **input** comes as 2 string arguments. The **first string** will represent **the main one**. The second one will represent **the substring**.

The **output** is either “true” or “false” based on the result of the check.  
The comparison is **case-sensitive**!

### Examples

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Input** | | **Output** |  | **Input** | | **Output** |
| This sentence ends with fun?  fun? | | true |  | This is Houston, we have…  We have… | | false |
|  | |  |  |  | |  |
| **Input** | | | | **Output** |
| The new iPhone has no headphones jack.  o headphones jack. | | | | true |

## \*Capitalize the Words

Write a JS function that capitalizes the given words. You need to make **every word**’s **first letter** – **uppercase** and **all** **other letters** – **lowercase**.

The **input** comes as a **single string**, containing words, separated by a space.

The **output** is the same string, however with all of its words capitalized.

Note: The words can contain **any ASCII character**. You need to **affect only the letters**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Capitalize these words | Capitalize These Words |
|  |  |
| **Input** | **Output** |
| Was that Easy? tRY thIs onE for SiZe! | Was That Easy? Try This One For Size! |

## Capture the Numbers

Write a JS function that **finds all numbers** in a sequence of strings.

The **input** comes as array of strings. Each element represents one of the strings.

The **output** is all the numbers, **extracted** and **printed on a single line** – each separated by a **single space**.

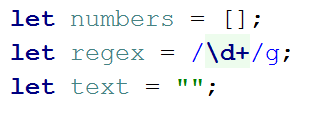
### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| The300  What is that?  I think it’s the 3rd movie.  Lets watch it at 22:45 | 300 3 22 45 |
|  |  |
| **Input** | **Output** |
| 123a456  789b987  654c321  0 | 123 456 789 987 654 321 0 |
| **Input** | **Output** |
| Let’s go11!!!11!  Okey!1! | 11 11 1 |

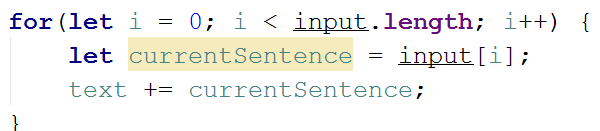
### Hints

We can solve this problem in multiple ways; first let’s see the more complex way in order to understand how the regex actually works:

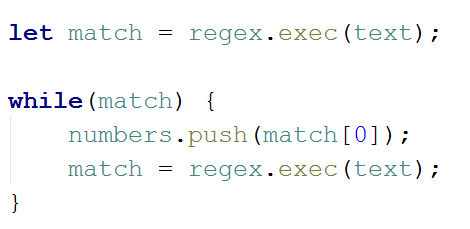
* We receive several sentences in the form of an array of strings. Let’s create a variable named text, and gather all the sentences into one big string. Also, we need to initialize our regex pattern, and an array that will hold the numbers we found.



* We create the needed things. The regex is “\d+” which will catch **one or more consecutive digits**. We also give it a global modifier, “g”, which will prevent the regex from returning on the first match.
* We can now proceed with combining all the strings into one big string.



* Now that we have that, we can proceed to the main thing. The matching.



* First we create a match variable which will hold our matches. The regex **anchors itself** every time, to the index of the match it has found, thus to iterate all matches we need a while loop. Every time we **match something**, we **push it** to the array of numbers… The match variable represents an **array of all groups it has found**, so we just take the first element, which represents the **whole match**. Then we match again, to **move the anchor**.
* Last but not least, we print the result:



Now that we understand how the matching works underneath, we can actually write a simpler solution. Having learned the Array built-in functions we know we can group the input into one string using the **Array.join()** function:



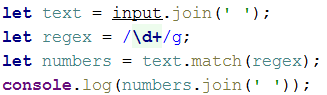
The regex we’ll use will be the same:



In case we don’t need capturing subgroups, as it is in this problem, we can just use the **String.match()** function to get all matches from our string (the regex still has to have the global flag **“g”**).



Thus we managed to write the program in just a few lines:



## Find Variable Names in Sentences

Write a JS function that finds all **variable names** in a given string. A variable name starts with an **underscore** (“\_”) and contains **only Capital and Non-Capital English Alphabet letters and digits**. Extract only their names, **without the underscore.** Try to do this **only with regular expressions**.

The **input** comes as single string, on which you have to perform the matching.

The **output** consists of all variable names, **extracted** and **printed on a single line**, each **separated** by a **comma**.

|  |  |
| --- | --- |
| **Input** | **Output** |
| The \_id and \_age variables are both integers. | id,age |
|  |  |
| **Input** | **Output** |
| Calculate the \_area of the \_perfectRectangle object. | area,perfectRectangle |
| **Input** | **Output** |
| \_\_invalidVariable \_evenMoreInvalidVariable\_ \_validVariable | validVariable |

### Hints

* Think how to ensure that your match is a separate word (not part of a bigger word that may be invalid).
* Think of a way to ensure your regex matches only the variable and not parts before/after it. Check the [special characters](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions) online to see if one of them fits your needs.

## Find Occurrences of Word in Sentence

Write a JS function that finds, **how many times** a **given word**, is **used** in a **given sentence.** Note that letter case does not matter – it is **case-insensitive**.

The **input** comes as 2 string arguments. The **first one** will be the **sentence**, and the **second one** – the **word**.

The **output** is a single number indicating the **amount of times** the sentence contains the word.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| The waterfall was so high, that the child couldn’t see its peak.  the | 2 |
|  |  |
| **Input** | **Output** |
| How do you plan on achieving that? How? How can you even think of that?  how | 3 |
| **Input** | **Output** |
| There was one. Therefore I bought it. I wouldn’t buy it otherwise.  there | 1 |

### Hints

* Remember how we always used the global modifier **“g”**. There is also a modifier for case-insensitive matching. It might help you since the case does **NOT** matter in this problem.
* Think how to create a regex that includes a string that will be known only at runtime. It is important to note that there is a Regex constructor that accepts a string representing the regex pattern.

## \*Extract the Links

Write a JS function that **extracts links** from a **given text**. The text will come in the form of strings, each representing a sentence. You need to extract **only the valid links** from it. Example:

“www.internet.com”

**Sub-Domain**  **Domain** **name** **Domain extension**

The **Sub-Domain** must always be “www”. The **Domain name** can consist of English alphabet letters (**uppercase** and **lowercase**), digits and dashes (“–“). The **Domain extension** consists of one or more **domain blocks**, a **domain block** consists of a **dot** followed by **one or more lowercase** English alphabet **letters**, a **Domain extension** must have at least **one** **domain block** in order to be **valid**. The Sub-Domain and Domain name must be separated by a single **dot**. Any link that **does NOT follow** the specified above rules should be treated as **invalid**.

**Example incorrect links:**

* "**ww**.justASite.bg"
* "**lel**.awesome.com"
* "www.stamat**\_**gosho.hit.bg"
* "www.no-symb**#^**ols-allow**%**ed.com"
* "www.pesho**.12**"
* "www.gosho-site**.**"
* "www.example-site**.\_\*^#**"

**Example correct links:**

* "Some textwww.softuni.bg"
* "Just a link in a www.sea-of-text.bg-swimming around"
* "Instruments www.Instruments.rom.com.trombone2000 Instrument here"
* "All your ice cream flavors-www.scream.for.ice.cream(We also have squirrels)"

The **input** comes as array of strings. Each element represents a sentence.

The **output** is all valid links you’ve found, printed – each on a new line.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Join WebStars now for free, at www.web-stars.com  You can also support our partners:  Internet - www.internet.com  WebSpiders - www.webspiders101.com  Sentinel - www.sentinel.-ko | www.web-stars.com  www.internet.com  www.webspiders101.com |
|  |  |
|  |  |
| **Input** | **Output** |
| Need information about cheap hotels in London?  You can check us at www.london-hotels.co.uk!  We provide the best services in London.  Here are some reviews in some blogs:  "London Hotels are awesome!" - www.indigo.bloggers.com  "I am very satisfied with their services" - ww.ivan.bg  "Best Hotel Services!" - www.rebel21.sedecrem.moc | www.london-hotels.co.uk  www.indigo.bloggers.com  www.rebel21.sedecrem.moc |

### Hints

* Sites such as <https://regex101.com/> can be very helpful, when designing regular expressions.

## \*\*Secret Data

Write a JS function that hides essential client data from secret documents that went public. You have to hide people’s names, phone numbers, ids and secret base names.

* The **names of the clients** will be preceded by a single **asterisk** (“\*”), they will always be **exactly one word**, they will contain **only English alphabet letters**, they will **start with a capital letter** and they will always be followed by either a **space,** a **tabulation** or the **end of the string**. Anything else is **NOT** **to be considered** as a name.
* The **phone numbers** of the clients will be preceded by a single **plus sign** (“+”) and will consist of exactly 10 symbols. The phone numbers can consist only of **digits** and **dashes** and they will always be followed by a **space, tabulation** or the **end of the string**. Anything else is **NOT to be considered** as a phone number.
* The **ID**s of the clients will be preceded by a single **exclamation mark** (“!”). The IDs of the clients will consist only of **Lowercase** and **Uppercase English alphabet letters** and **digits** and they will always be followed by a **space, tabulation** or the **end of the string**. Anything else is **NOT to be considered** as an ID.
* The **names of the secret bases** will be preceded by a single **underscore** (“\_”) and will consist only of **Uppercase** and **Lowercase English alphabet letters** and **digits** and they will always be followed by a **space, tabulation** or the **end of the string**. Anything else is **NOT to be considered** as a secret base name.

### Constraints

* **Usernames, phone numbers, IDs and names of secret bases** can start glued to other text.
* **Usernames, phone numbers, IDs and names of secret bases** will never be split given across 2 lines.
* **Usernames, phone numbers, IDs and names of secret bases** will always have a **space, tabulation** or the **end of the string** after them.

The **input** comes as an array of strings. Each string represents a sentence of the secret document. You need to find every **client data element** that is supposed to be secret, and **hide it**, by **replacing it** with a **string of** **vertical bars** (“|”) with the **same length**, as the **discovered element**.

**NOTE: The preceding symbol counts towards the discovered element’s length when deciding how many pipes to use to cover it.**

The **output** should be the same document, with the sensitive **client** **data replaced by pipes**. See the example for more info.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| Agent \*Ivankov was in the room when it all happened.  The person in the room was heavily armed.  Agent \*Ivankov had to act quick in order.  He picked up his phone and called some unknown number.  I think it was +555-49-796  I can't really remember...  He said something about "finishing work" with subject !2491a23BVB34Q and returning to Base \_Aurora21  Then after that he disappeared from my sight.  As if he vanished in the shadows.  A moment, shorter than a second, later, I saw the person flying off the top floor.  I really don't know what happened there.  This is all I saw, that night.  I cannot explain it myself... | Agent |||||||| was in the room when it all happened.  The person in the room was heavily armed.  Agent |||||||| had to act quick in order.  He picked up his phone and called some unknown number.  I think it was |||||||||||  I can't really remember...  He said something about "finishing work" with subject |||||||||||||| and returning to Base |||||||||  Then after that he disappeared from my sight.  As if he vanished in the shadows.  A moment, shorter than a second, later, I saw the person flying off the top floor.  I really don't know what happened there.  This is all I saw, that night.  I cannot explain it myself... |