# Exercises: Strings and Regular Expressions

Problems for exercises and homework 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/314/>.

In this exercise you are supposed to **write program logic** using the basic operations between Strings and the built-in String functions, as well as regular expressions, in JavaScript. You will practice working with **strings**, **regular expressions** and using **built-in** **functions** (concat(), trim(), split()…). In some of the exercises you might need to combine both in order to find the best solution.

# Text Processing and String Manipulation

## 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**.

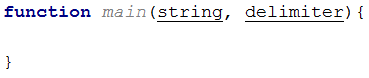
The **output** should consist of all elements you’ve received, after you’ve **split** **the given string** by **the given delimiter**. Each element should be printed on a new line.

### Examples

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

### Hints

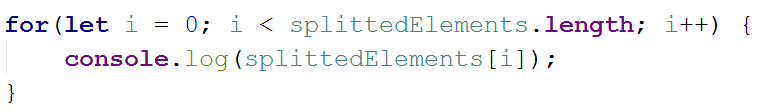
* This “main” function will hold all of the code of our solution.



* 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! |

# Working with Regular Expressions

For the following tasks, it will be most appropriate, if you use regular expressions in your solutions.

## 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**.

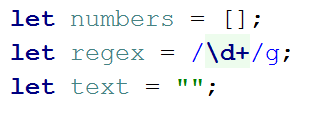
### Examples

|  |  |
| --- | --- |
| **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 |

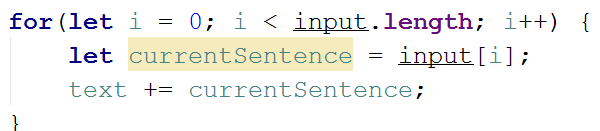
### Hints

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:

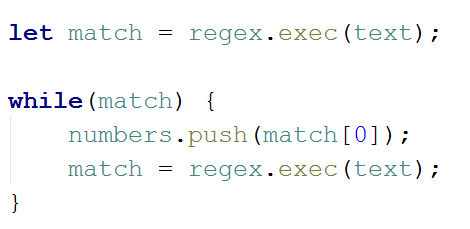
* We receive several sentences in the form of an array of strings. Let’s create a variable named text, and gather all the sentences into one big string. Also, we need to initialize our regex pattern, and an array that will hold the numbers we found.



* 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:



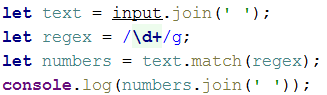
The regex we’ll use will be the same:



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 896666666666  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... |