**JavaScript Basics #5: Regular Expressions**

The topic we are going to discuss in this article is called regular expression. It is technically not a part of JavaScript, it’s a separate language that is built into JavaScript as well as other programming languages. Regular expression has a very awkward and cryptic syntax, but it is also very useful. It is widely used among programmers as a tool to describe, match and replace patterns in string data.

Table of Contents

* [Create a Regular Expression](https://www.ericsdevblog.com/index.php/2021/10/javascript-basics-5-regular-expressions/#Create_a_Regular_Expression)
* [Matching Patterns](https://www.ericsdevblog.com/index.php/2021/10/javascript-basics-5-regular-expressions/#Matching_Patterns)
  + [Match a Set of Characters](https://www.ericsdevblog.com/index.php/2021/10/javascript-basics-5-regular-expressions/#Match_a_Set_of_Characters)
  + [Match Repeating Patterns](https://www.ericsdevblog.com/index.php/2021/10/javascript-basics-5-regular-expressions/#Match_Repeating_Patterns)
  + [Other Methods for Matching Patterns](https://www.ericsdevblog.com/index.php/2021/10/javascript-basics-5-regular-expressions/#Other_Methods_for_Matching_Patterns)
  + [Boundary Markers](https://www.ericsdevblog.com/index.php/2021/10/javascript-basics-5-regular-expressions/#Boundary_Markers)
  + [Choice Patterns](https://www.ericsdevblog.com/index.php/2021/10/javascript-basics-5-regular-expressions/#Choice_Patterns)
* [Replacing a Pattern](https://www.ericsdevblog.com/index.php/2021/10/javascript-basics-5-regular-expressions/#Replacing_a_Pattern)

**CREATE A REGULAR EXPRESSION**

A regular expression is an object. There are two ways you can create a regular expression in JavaScript. You can either use a RegExp() constructor or you can enclose the pattern inside a pair of forward-slash (/) characters.

let re1 = new RegExp("abc");

let re2 = /abc/;

Both of these examples describe the same pattern: a character a followed by a b followed by a c. The second notation, however, treats backslash (\) characters differently. For example, since the forward-slash denotes the pattern, if you want a forward-slash to be a part of the pattern, you need to put a backslash in front of it.

**MATCHING PATTERNS**

Regular expression offers a handful of methods for us to use, the most commonly used one should be the test() method, which is used for matching patterns in string data.

console.log(/abc/.test("abcde"));

// → true

console.log(/abc/.test("abxde"));

// → false

In this example, the test() method will examine the string that is passed to it, and return a boolean value telling you if a pattern match is found.

**Match a Set of Characters**

However, simply testing if the pattern "abc" is found in a string does not seem very useful. Sometimes we want to test for a match using a set of characters. For example, the following code test if **at least one** of the characters, from character 0 to character 9, exists in the string "in 1992".

console.log(/[0123456789]/.test("in 1992"));

// → true

// A hyphen character can be used to indicate a range of characters

console.log(/[0-9]/.test("in 1992"));

// → true

It is also possible to match any character that is **not** in the set. For example, this time we’ll match any character that is **not** 1 or 0.

let notBinary = /[^01]/;

console.log(notBinary.test("1100100010100110"));

// → false

// The string contains a character "2" which is not in the set [01]

console.log(notBinary.test("1100100010200110"));

// → true

Some of the commonly used character sets have shortcuts in regular expressions. For instance, \d represents all digit characters, same as [0-9].

* \d Any digit character
* \w Any alphanumeric character (word character)
* \s Any whitespace character (space, tab, new line …)
* \D Any nondigit character
* \W Any nonalphanumeric character
* \S Any nonwhitespace character
* . Any character except for new line

Now, we could match a date-time format (10-07-2021 16:06) like this:

let dateTime = /\d\d-\d\d-\d\d\d\d \d\d:\d\d/;

console.log(dateTime.test("10-07-2021 16:06"));

// → true

**Match Repeating Patterns**

You may have noticed that in our previous example, each \d only matches one digit character. What if we want to match a sequence of digits of arbitrary length? We can do that by putting a plus mark (+) after the element we wish to repeat.

console.log(/'\d+'/.test("'123'"));

// → true

console.log(/'\d+'/.test("''"));

// → false

The star sign has a similar meaning except it allows the element to match for zero times.

console.log(/'\d\*'/.test("'123'"));

// → true

console.log(/'\d\*'/.test("''"));

// → true

We can also indicate precisely how many times we want the element to repeat. For example, if we put {4} after an element, that means this element will be repeated four times. If we put {2,4} after that element, it means the element will be repeated at least twice and at most four times.

It is possible to repeat a group of elements as well. We only need to enclose that group of elements inside a pair of parentheses.

let cartoonCrying = /boo+(hoo+)+/i;

console.log(cartoonCrying.test("Boohoooohoohooo"));

// → true

In some cases, we need a part of the pattern to be optional. For example, the word “neighbour” can also be spelled “neighbor”, which means the character “u” should be optional. Here is what we can do:

let neighbor = /neighbou?r/;

console.log(neighbor.test("neighbour"));

// → true

console.log(neighbor.test("neighbor"));

// → true

**Other Methods for Matching Patterns**

The test() method is the simplest way of finding out if a pattern match is found in a string. However, it doesn’t give you much information besides returning a boolean value telling you if a match is found.

The regular expression also has an exec() method (exec stands for execute) that would return an object giving you more information, such as what the match is and where it is found.

let match = /\d+/.exec("one two 100");

console.log(match);

// → ["100"]

// The index property tells you where in the string the match begins

console.log(match.index);

// → 8

There is also a match() method that belongs to the string type, which behaves similarly.

console.log("one two 100".match(/\d+/));

// → ["100"]

The exec() method can be very useful in practice. For example, we can extract a date and time from a string like this:

let [\_, month, day, year] = /(\d{1,2})-(\d{1,2})-(\d{4})/.exec("1-30-2021");

The underscore (\_) is ignored, it is used to skip the full match that is returned by the exec() method.

**Boundary Markers**

However, now we have another problem from the previous example. If we pass to the exec() method a sequence of nonsense like "100-1-3000", it would still happily extract a date from it.

In this case, we must enforce that the match must span the entire string. To do that, we use the boundary markers ^ and $. The caret sign (^) marks the start of the string and the dollar sign ($) matches the end of the string. So, for instance, the pattern /^\d$/ would match a string that only consists of one digit character.

Sometimes you don’t want the match to be the entire string, but you want it to be a whole word and not just a part of the word. To mark a word boundary, we use the \b marker.

console.log(/cat/.test("concatenate"));

// → true

console.log(/\bcat\b/.test("concatenate"));

// → false

**Choice Patterns**

The Last type of pattern I’d like to introduce is the choice pattern. Sometimes we don’t want to match a specific pattern, but instead, we have a list of acceptable patterns. we can divide the different patterns using the pipe character (|).

let animalCount = /\b\d+ (pig|cow|chicken)s?\b/;

console.log(animalCount.test("15 pigs"));

// → true

console.log(animalCount.test("15 pigchickens"));

// → false

**REPLACING A PATTERN**

Besides the match() method, string values also have a replace() method that replaces part of the string with another string.

console.log("papa".replace("p", "m"));

// → mapa

The first argument of the replace() method can also be a regular expression, in which case the first match of that regular expression will be replaced with the second argument. If you wish to replace all matches of the regular expression, add a g option (global option) to that regular expression.

console.log("Borobudur".replace(/[ou]/, "a"));

// → Barobudur

console.log("Borobudur".replace(/[ou]/g, "a"));

// → Barabadar