**Most Used Regex Patterns In JavaScript**

Before going into detail and explaining the different operations doable with Regex, you should know the most used Regex Patterns in JavaScript.

* ^: Defines the start of the pattern.
* $: Defines the end of the pattern.
* [] Defines a character set.
* () Defines a capturing group that groups multiple characters.
* [abcdef]: Matches any single occurrence of any of the characters a, b, c, d, e, f.
* [a-z]: Matches any single lowercase letter.
* [A-Z]: Matches any single uppercase letter
* [a-zA-Z]: Matches any single letter from a-z ignoring the case
* .: Matches any character except line breaks.
* ?: Matches 0 or 1 occurrence of the preceding character, set, or group
* \*: Matches 0 or more occurrences of the preceding character, set, or group
* +: Matches 1 or more occurrences of the preceding character, set, or group
* {n}: Matches exactly n occurrences of the preceding character, set, or group
* {n, m}: Matches exactly n or m occurrences of the preceding character, set, or group

**How To Extract A Substring With Regex**

Substrings can be extracted from a large String using the match() function and a Regular Expression in JavaScript. match [is a native JavaScript String function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/match?ref=paulsblog.dev) that uses a provided Regex to retrieve the result of a matching String.

The JavaScript match function will always return an array containing any matching String or nothing.

See the following example to better understand match:

const paragraph = 'The quick brown fox jumps over the lazy dog. It barked.';

const regexUppercase = /[A-Z]/g;

const regexFox = /fox/;

console.log(paragraph.match(regexUppercase)); // Array ["T", "I"]

console.log(paragraph.match(regexFox)); // Array ["fox"]

Copy

In this example, a paragraph String, a regexUppercase Regex, and a regexFox Regex is defined:

* paragraph: A String containing the text: The quick brown fox jumps over the lazy dog. It barked.
* regexUppercase: A regex that matches any uppercase letter
* regexFox: A regex that matches the String "fox"

Next, two console.log will use the Regex to retrieve all results from the provided paragraph String. **Be aware that match returns an array of Strings, also if the result is only one String.**

**How To Replace A Substring With Regex**

Another feature of Regex in JavaScript is replacing a single part of any String with a new value using the String replace function. Normally, every JavaScript developer should know how to use the replace function with two provided Strings, but using it with a regex is also possible and often not known.

One benefit of using Regex instead of a simple String within the replace function is that it is possible to ignore the case or set it to global replacement. See the following example for some examples:

const regex = /apples/g;

const regexIgnoreCase = /apples/gi

const regexSingle = /apples/i

const str = "Apples are round, and apples are juicy.";

const newstr1 = str.replace(regex, "oranges"); // "Apples are round, and oranges are juicy."

const newstr2 = str.replace(regexIgnoreCase, "oranges"); // "oranges are round, and oranges are juicy."

const newstr3 = str.replace(regexSingle, "oranges"); // "oranges are round, and apples are juicy."

Copy

This example contains three different Regular Expressions:

* regex: This Regex will replace "apples" globally but will be case sensitive
* regexIgnoreCase: This Regex will replace "apples" globally and will be case insensitive
* regexSingle: This Regex will replace "apples" one time but will be case insensitive

In addition to simple replacements of Strings, it is possible to switch words with Regex in JavaScript:

const re = /(\w+)\s(\w+)/;

const str = "Maria Cruz";

const newstr = str.replace(re, "$2, $1");

console.log(newstr); // Cruz, Maria

Copy

This rather complex Regex will extract the first word $1 and the second word $2 from the String str and create the result String $2, $1.

**How To Split A String With Regex**

In JavaScript, any String can be split into an array of Substrings by using the split() function with a separator like " ", ",", or ".". If using split() with a separator, the input String will be split wherever the separator is present.

const str = 'The quick brown fox jumps over the lazy dog.';

const words = str.split(' ');

console.log(words); // Array ["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog."]

Copy

In this code snippet, the String str will be split wherever a space (" ") is present resulting in an output array  ["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog."]

In addition to using a separator, the split() function works with a Regular Expression. By using a Regex, the split() function will split the String wherever the provided Regex matches the String. This comes in handy if you want to extract specific parts of an input String. For example, split an html String into an array containing any <div> ... </div>. Also, this is needed if multiple separators have to be used to split the String.

For example, let's take the previously defined String str and replace the whitespace with different whitespace characters (\n, \t, and " "). By doing this the used split() function will not work because it only split the String on any " ". But, by providing the Regex /\s+/ the String will be split on ANY whitespace character.

const str = "The\tquick\tbrown\tfox jumps over\nthe lazy dog";

const words = str.split(/\s+/);

console.log(words); // Array ["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"]

Copy

A more complex example can be found in the next code snippet from the MDN web docs:

const names = "Harry Trump ;Fred Barney; Helen Rigby ; Bill Abel ;Chris Hand ";

const regex = /\s\*(?:;|$)\s\*/;

const nameList = names.split(regex); Array ["Harry Trump", "Fred Barney", "Helen Rigby", "Bill Abel", "Chris Hand", ""]

Copy

In this example, a rather complex Regex is used to split the String. With the Regex, the split() function searches for zero or more spaces that are followed by a semicolon, followed by zero or more spaces. Then all found spaces/semicolons are removed from the String. The resulting array nameList will only return the names from the input String names.

By using Regular Expressions as a separator for the split() function, you can split Strings in many ways in a single line of code resulting in a well-structured array.

**How To Match A String With Regex**

The most known usage of Regular Expressions is testing if a String matches a provided pattern by using the JavaScript test() function with a Regular Expression.

For example, checking a String if it is an Email can be done with this code snippet:

const isEmailValid = (email) =>

/^\w+([\.-]?\w+)\*@\w+([\.-]?\w+)\*(\.\w{2,3})+$/.test(email);

isEmailValid('test@email.de'); // true

isEmailValid('etsta'); // false

isEmailValid(''); // false

Copy

In this example, a function isEmailValid() is created which will use a really complex Regular Expression to check if a provided String email is a valid Email address.

As the Regular Expression /^\w+([.-]?\w+)*@\w+([.-]?\w+)*(.\w{2,3})+$/ looks very complex, it will be explained in detail:

* The first character / and the last character / delimit the Regular Expression pattern.
* ^: matches the beginning of the String.
* \w+: matches any word character (alphanumeric & underscore). The + indicates that there has to be at least 1 word.
* ([.-]?\w+)\*: this part is optional (\*) and will match any String that can (didn't have to) start with a . or - (only one of them) and followed for at least one word (+).
* @: will match the '@' character.
* \w+: matches any word character (alphanumeric & underscore). The + indicates that there has to be at least 1 word.
* ([.-]?\w+)\*:  this part is optional (\*) and will match any String that can (didn't have to) start with a . or - (only one of them) and followed for at least one word (+).
* (.\w{2,3})+: this part matches any String starting with a . and is followed by 2 or 3-word characters.
* $: Matches the end of the String.

**Closing Notes**

**Regular Expressions are powerful**! Using them in your JavaScript software project can enhance it and optimize quality and performance. However, using them too often in your project could make it less readable for "Regular Expression unfamiliar" software developers.

By fully understanding Regular Expression, you should be well-equipped to tackle complex JavaScript projects and build robust, high-quality software.

However, mastering Regular Expressions could take a lot of time and practice, so please don't get discouraged if you do not fully understand them immediately. Keep experimenting with them to further increase your skill.

I recommend either [regexr.com](https://regexr.com/?ref=paulsblog.dev) or [regex101.com](https://regex101.com/?ref=paulsblog.dev), which both have an easily understandable UI and will explain every Regex pattern in detail. Regexr also has developed [a Chrome extension](https://chrome.google.com/webstore/detail/regexr/igncokdengclanbghanhfichlnhlpihl?hl=de&ref=paulsblog.dev) that can be installed and used within the browser.

Hopefully, this article gave you a quick and neat overview of how to use Regular Expressions in your JavaScript project.

Do you have any questions regarding Regular Expressions? I would love to hear your feedback, your thoughts and answer all your questions. Please share everything in the comments.