Regular expressions

Hello. My name is Liudmila. In this video I want to talk about regular expressions.

A **regular expression**, **regex** or **regexp** is a sequence of [characters](https://en.wikipedia.org/wiki/Character_(computing)) that define a *search*[*pattern*](https://en.wikipedia.org/wiki/Pattern_matching). Usually such patterns are used by [string searching algorithms](https://en.wikipedia.org/wiki/String_searching_algorithm) for "find" or "find and replace" operations on [strings](https://en.wikipedia.org/wiki/String_(computer_science)), or for input validation.

Regular expressions originated in 1951, when mathematician [Stephen Cole Kleene](https://en.wikipedia.org/wiki/Stephen_Cole_Kleene) described [regular languages](https://en.wikipedia.org/wiki/Regular_language) using his mathematical notation called *regular events*. These arose in [theoretical computer science](https://en.wikipedia.org/wiki/Theoretical_computer_science), in the subfields of [automata theory](https://en.wikipedia.org/wiki/Automata_theory) and the description and classification of [formal languages](https://en.wikipedia.org/wiki/Formal_language).

Today, regexes are widely supported in programming languages, text processing programs, advanced text editors, and some other programs. Regex support is part of the [standard library](https://en.wikipedia.org/wiki/Standard_library) of many programming languages, including [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) and [Python](https://en.wikipedia.org/wiki/Python_(programming_language)), and is built into the syntax of others, including Perl and [ECMAScript](https://en.wikipedia.org/wiki/ECMAScript).

Regular expressions are patterns used to match character combinations in strings. In JavaScript, regular expressions are also objects.

# Creating a regular expression

There are two syntaxes that can be used to create a regular expression object.

The “long” syntax:

regexp = new RegExp("pattern", "flags”);

And the “short” one, using slashes "/":

regexp = /pattern/; // no flags

regexp = /pattern/gmi; // with flags g,m and i (to be covered soon);

Slashes /.../ tell JavaScript that we are creating a regular expression. They play the same role as quotes for strings. In both cases regexp becomes an instance of the built-in RegExp class.

The main difference between these two syntaxes is that pattern using slashes /.../ does not allow for expressions to be inserted (like string template literals with ${...}). They are fully static.

# What is that flag in regular expression?

Regular expressions have six optional flags that allow for functionality like global and case insensitive searching. These flags can be used separately or together in any order, and are included as part of the regular expression.

| **Regular expression flags** | |
| --- | --- |
| **Flag** | **Description** |
| g | Global search. With this flag the search looks for all matches, without it – only the first match is returned. |
| i | Case-insensitive search. With this flag the search is case-insensitive: no difference between A and a |
| m | Multi-line search. |
| s | Allows . to match newline characters. (Added in ES2018, not yet supported in Firefox)  Enables “dotall” mode, that allows a dot . to match newline character |
| u | "unicode"; treat a pattern as a sequence of unicode code points. Enables full unicode support. The flag enables correct processing of surrogate pairs. |
| y | Perform a "sticky" search that matches starting at the current position in the target string. |

Note that the flags are an integral part of a regular expression. They cannot be added or removed later.

For example, re = /\w+\s/g creates a regular expression that looks for one or more characters followed by a space, and it looks for this combination throughout the string.

let re = /\w+\s/g;

let str = 'fee fi fo fum';

let myArray = str.match(re);

console.log(myArray);

// ["fee ", "fi ", "fo "]

You could replace the line:

let re = /\w+\s/g;

with:

let re = new RegExp('\\w+\\s', 'g');

and get the same result.

A regular expression pattern is composed of simple characters, such as /abc/, or a combination of simple and special characters, such as /ab\*c/ or /Chapter (\d+)\.\d\*/. The last example includes parentheses which are used as a memory device.

Simple patterns are constructed of characters for which you want to find a direct match. For example, the pattern /abc/ matches character combinations in strings only when exactly the characters 'abc' occur together and in that order. Such a match would succeed in the strings "Hi, do you know your abc's?" and "The latest airplane designs evolved from slabcraft." In both cases the match is with the substring 'abc'. There is no match in the string 'Grab crab' because while it contains the substring 'ab c', it does not contain the exact substring 'abc'.

### **Using special characters**

When the search for a match requires something more than a direct match, such as finding one or more b's, or finding white space, you can include special characters in the pattern. For example, to match a single 'a' followed by zero or more 'b's followed by 'c', you'd use the pattern /ab\*c/: the \* after 'b' means "0 or more occurrences of the preceding item." In the string "cbbabbbbcdebc," the pattern matches the substring 'abbbbc'.

A complete list and description of the special characters that can be used in regular expressions can be finding at <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions>

Here are some examples:

**Special characters in regular expressions.**

| **Character** | **Meaning** |
| --- | --- |
| [\](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-backslash) | Matches according to the following rules:  A backslash that precedes a non-special character indicates that the next character is special and is not to be interpreted literally. For example, a 'b' without a preceding '\' generally matches lowercase 'b's wherever they occur — the character will be interpreted literally. But a sequence of '\b' doesn't match any character; it denotes a [word boundary](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-word-boundary). |
| [^](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-caret) | Matches beginning of input. If the multiline flag is set to true, also matches immediately after a line break character.  For example, /^A/ does not match the 'A' in "an A", but does match the 'A' in "An E".  The '^' has a different meaning when it appears as the first character in a character set pattern. |
| [$](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-dollar) | Matches end of input. If the multiline flag is set to true, also matches immediately before a line break character.  For example, /t$/ does not match the 't' in "eater", but does match it in "eat". |
| [\*](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-asterisk) | Matches the preceding expression 0 or more times. Equivalent to {0,}.  For example, /bo\*/ matches 'boooo' in "A ghost booooed" and 'b' in "A bird warbled" but nothing in "A goat grunted". |
| [+](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-plus) | Matches the preceding expression 1 or more times. Equivalent to {1,}.  For example, /a+/ matches the 'a' in "candy" and all the a's in "caaaaaaandy", but nothing in "cndy". |
| [.](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-dot) | (The decimal point) matches any single character except the newline character, by default.  For example, /.n/ matches 'an' and 'on' in "nay, an apple is on the tree", but not 'nay'.  If the s ("dotAll") flag is set to true, it also matches newline characters. |
| [(x)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-capturing-parentheses) | Matches 'x' and remembers the match, as the following example shows. The parentheses are called capturing parentheses.  The '(foo)' and '(bar)' in the pattern /(foo) (bar) \1 \2/ match and remember the first two words in the string "foo bar foo bar". The \1 and \2 denote the first and second parenthesized substring matches - foo and bar, matching the string's last two words. Note that \1, \2, ..., \n are used in the matching part of the regex, for more information, see [\n](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-backreference) below. In the replacement part of a regex the syntax $1, $2, ..., $n must be used, e.g.: 'bar foo'.replace(/(...) (...)/, '$2 $1'). $& means the whole matched string. |
| [\B](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-non-word-boundary) | Matches a non-word boundary. This matches the following cases:  For example, /\B../ matches 'oo' in "noonday", and /y\B./ matches 'ye' in "possibly yesterday." |
| [\cX](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-control) | Where X is a character ranging from A to Z. Matches a control character in a string.  For example, /\cM/ matches control-M (U+000D) in a string. |
| [\d](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-digit) | Matches a digit character. Equivalent to [0-9].  For example, /\d/ or /[0-9]/ matches '2' in "B2 is the suite number." |
| [\D](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-non-digit) | Matches a non-digit character. Equivalent to [^0-9].  For example, /\D/ or /[^0-9]/ matches 'B' in "B2 is the suite number." |
| [\f](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-form-feed) | Matches a form feed (U+000C). |
| [\n](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-line-feed) | Matches a line feed (U+000A). |
| [\r](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-carriage-return) | Matches a carriage return (U+000D). |
| [\s](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-white-space) | Matches a white space character, including space, tab, form feed, line feed. Equivalent to [ \f\n\r\t\v\u00a0\u1680\u2000-\u200a\u2028\u2029\u202f\u205f\u3000\ufeff].  For example, /\s\w\*/ matches ' bar' in "foo bar." |
| [\S](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-non-white-space) | Matches a character other than white space. Equivalent to [^ \f\n\r\t\v\u00a0\u1680\u2000-\u200a\u2028\u2029\u202f\u205f\u3000\ufeff].  For example, /\S\*/ matches 'foo' in "foo bar." |
| [\t](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-tab) | Matches a tab (U+0009). |
| [\v](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-vertical-tab) | Matches a vertical tab (U+000B). |
| [\w](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-word) | Matches any alphanumeric character including the underscore. Equivalent to [A-Za-z0-9\_].  For example, /\w/ matches 'a' in "apple," '5' in "$5.28," and '3' in "3D." |
| [\W](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-non-word) | Matches any non-word character. Equivalent to [^A-Za-z0-9\_].  For example, /\W/ or /[^A-Za-z0-9\_]/ matches '%' in "50%." |

Parentheses around any part of the regular expression pattern causes that part of the matched substring to be remembered. Once remembered, the substring can be recalled for other use, For example, the pattern /Chapter (\d+)\.\d\*/ illustrates additional escaped and special characters and indicates that part of the pattern should be remembered. It matches precisely the characters 'Chapter ' followed by one or more numeric characters (\d means any numeric character and + means 1 or more times), followed by a decimal point (which in itself is a special character; preceding the decimal point with \ means the pattern must look for the literal character '.'), followed by any numeric character 0 or more times (\d means numeric character, \* means 0 or more times). In addition, parentheses are used to remember the first matched numeric characters.

# Escaping

If you need to use any of the special characters literally (actually searching for a '\*', for instance), you must escape it by putting a backslash in front of it. For instance, to search for 'a' followed by '\*' followed by 'b', you'd use /a\\*b/—the backslash "escapes" the '\*', making it literal instead of special.

Similarly, if you're writing a regular expression literal and need to match a slash ('/'), you need to escape that (otherwise, it terminates the pattern). For instance, to search for the string "/example/" followed by one or more alphabetic characters, you'd use /\/example\/[a-z]+/i—the backslashes before each slash make them literal.

If escape strings are not already part of your pattern you can add them using [String.replace](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/replace" \o "The replace() method returns a new string with some or all matches of a pattern replaced by a replacement.):

function escapeRegExp(string) {

return string.replace(/[.\*+?^${}()|[\]\\]/g, '\\$&'); // $& means the whole matched string

}

# Working with regular expressions

Regular expressions are used with the RegExp methods test and exec and with the String methods match, replace, search, and split.

| **Methods that use regular expressions** | |
| --- | --- |
| **Method** | **Description** |
| [exec](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp/exec) | A RegExp method that executes a search for a match in a string. It returns an array of information or null on a mismatch. |
| [test](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp/test) | A RegExp method that tests for a match in a string. It returns true or false. |
| [match](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/match) | A String method that returns an array containing all of the matches, including capturing groups, or null if no match is found. |
| [matchAll](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/matchAll) | A String method that returns an iterator containing all of the matches, including capturing groups. |
| [search](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/search) | A String method that tests for a match in a string. It returns the index of the match, or -1 if the search fails. |
| [replace](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/replace) | A String method that executes a search for a match in a string, and replaces the matched substring with a replacement substring. |
| [split](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/split) | A String method that uses a regular expression or a fixed string to break a string into an array of substrings. |

When you want to know whether a pattern is found in a string, use the test or search method; for more information (but slower execution) use the exec or match methods. If you use exec or match and if the match succeeds, these methods return an array and update properties of the associated regular expression object and also of the predefined regular expression object, RegExp. If the match fails, the exec method returns null (which coerces to false).

In the following example, the script uses the exec method to find a match in a string.

var myRe = /d(b+)d/g;

var myArray = myRe.exec('cdbbdbsbz');

If you do not need to access the properties of the regular expression, an alternative way of creating myArray is with this script:

var myArray = /d(b+)d/g.exec('cdbbdbsbz');

// similar to "cdbbdbsbz".match(/d(b+)d/g); however,

// "cdbbdbsbz".match(/d(b+)d/g) outputs Array [ "dbbd" ], while

// /d(b+)d/g.exec('cdbbdbsbz') outputs Array [ 'dbbd', 'bb', index: 1, input: 'cdbbdbsbz' ].

With these script, the match succeeds and returns the array and updates the properties shown in the following table.

The topic of “regular expressios” is very extensive and in this video I tried to talk about the most important points. I hope you enjoyed it. Thanks for watching