**Expresiones regulares para scraping**

**Creacion de expresiones regulares**

var re = new RegExp('ab+c');

var re = new RegExp('ab+c');

**Caracteres para expresiones regulares**

| **haracter** | **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. But a '\b' by itself doesn't match any character; it forms the special [word boundary character](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-word-boundary).  A backslash that precedes a special character indicates that the next character is not special and should be interpreted literally. For example, the pattern /a\*/ relies on the special character '\*' to match 0 or more a's. By contrast, the pattern /a\\*/ denotes the '\*'  as not special, enabling matches with strings like 'a\*'.  Do not forget to escape \ itself while using the RegExp("pattern") notation because \ is also an escape character in strings. |
| [^](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. See [complemented character sets](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-negated-character-set" \o "#special-negated-character-set) for details and an example. |
| [$](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-dollar) | Matches end of input. If the multiline flag is set to true, also matches immediately before a line break character.  For example, /t$/ 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-questionmark) | Matches the preceding expression 0 or 1 time. Equivalent to {0,1}.  For example, /e?le?/ matches the 'el' in "angel" and the 'le' in "angle" and also the 'l' in "oslo".  If used immediately after any of the quantifiers \*, +, ?, or {}, makes the quantifier non-greedy (matching the fewest possible characters), as opposed to the default, which is greedy (matching as many characters as possible). For example, applying /\d+/ to "123abc" matches "123". But applying /\d+?/ to that same string matches only the "1".  Also used in lookahead assertions, as described in the x(?=y) and x(?!y) entries of this table. |
| [.](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.  For example, /.n/ matches 'an' and 'on' in "nay, an apple is on the tree", but not 'nay'. |
| [(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 in the pattern match the string's last two words. Note that \1, \2, ..., \n are used in the matching part of the regex. 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. |
| [(?:x)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-non-capturing-parentheses) | Matches 'x' but does not remember the match. The parentheses are called *non-capturing parentheses*, and let you define subexpressions for regular expression operators to work with. Consider the sample expression /(?:foo){1,2}/. If the expression was /foo{1,2}/, the {1,2} characters would apply only to the last 'o' in 'foo'. With the non-capturing parentheses, the {1,2} applies to the entire word 'foo'. For more information, see [Using parentheses](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "Using_parentheses) below. |
| [x(?=y)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-lookahead) | Matches 'x' only if 'x' is followed by 'y'. This is called a lookahead.  For example, /Jack(?=Sprat)/ matches 'Jack' only if it is followed by 'Sprat'. /Jack(?=Sprat|Frost)/ matches 'Jack' only if it is followed by 'Sprat' or 'Frost'. However, neither 'Sprat' nor 'Frost' is part of the match results. |
| [x(?!y)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-negated-look-ahead) | Matches 'x' only if 'x' is not followed by 'y'. This is called a negated lookahead.  For example, /\d+(?!\.)/ matches a number only if it is not followed by a decimal point. The regular expression /\d+(?!\.)/.exec("3.141") matches '141' but not '3.141'. |
| [x|y](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-or) | Matches 'x', or 'y' (if there is no match for 'x').  For example, /green|red/ matches 'green' in "green apple" and 'red' in "red apple." The order of 'x' and 'y' matters. For example a\*|b matches the empty string in "b", but b|a\* matches "b" in the same string. |
| [{n}](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-quantifier) | Matches exactly n occurrences of the preceding expression. N must be a positive integer.  For example, /a{2}/ doesn't match the 'a' in "candy," but it does match all of the a's in "caandy," and the first two a's in "caaandy." |
| [{n,}](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-quantifier) | Matches at least n occurrences of the preceding expression. N must be a positive integer.  For example, /a{2,}/ will match "aa", "aaaa" and "aaaaa" but not "a" |
| [{n,m}](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-quantifier-range) | Where n and m are positive integers and n <= m. Matches at least n and at most m occurrences of the preceding expression. When m is omitted, it's treated as ∞.  For example, /a{1,3}/ matches nothing in "cndy", the 'a' in "candy," the first two a's in "caandy," and the first three a's in "caaaaaaandy". Notice that when matching "caaaaaaandy", the match is "aaa", even though the original string had more a's in it. |
| [[xyz]](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-character-set) | Character set. This pattern type matches any one of the characters in the brackets, including [escape sequences](https://developer.mozilla.org/en-US/docs/JavaScript/Guide/Values,_variables,_and_literals#Unicode_escape_sequences). Special characters like the dot(.) and asterisk (\*) are not special inside a character set, so they don't need to be escaped. You can specify a range of characters by using a hyphen, as the following examples illustrate.  The pattern [a-d], which performs the same match as [abcd], matches the 'b' in "brisket" and the 'c' in "city". The patterns /[a-z.]+/ and /[\w.]+/ match the entire string "test.i.ng". |
| [[^xyz]](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-negated-character-set) | A negated or complemented character set. That is, it matches anything that is not enclosed in the brackets. You can specify a range of characters by using a hyphen. Everything that works in the normal character set also works here.  For example, [^abc] is the same as [^a-c]. They initially match 'r' in "brisket" and 'h' in "chop." |
| [[\b]](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-backspace) | Matches a backspace (U+0008). You need to use square brackets if you want to match a literal backspace character. (Not to be confused with \b.) |
| [\b](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-word-boundary) | Matches a word boundary. A word boundary matches the position where a word character is not followed or preceded by another word-character. Note that a matched word boundary is not included in the match. In other words, the length of a matched word boundary is zero. (Not to be confused with [\b].)  Examples: /\bm/ matches the 'm' in "moon" ; /oo\b/ does not match the 'oo' in "moon", because 'oo' is followed by 'n' which is a word character; /oon\b/ matches the 'oon' in "moon", because 'oon' is the end of the string, thus not followed by a word character; /\w\b\w/ will never match anything, because a word character can never be followed by both a non-word and a word character.  **Note:** JavaScript's regular expression engine defines a [specific set of characters](http://www.ecma-international.org/ecma-262/5.1/#sec-15.10.2.6)to be "word" characters. Any character not in that set is considered a word break. This set of characters is fairly limited: it consists solely of the Roman alphabet in both upper- and lower-case, decimal digits, and the underscore character. Accented characters, such as "é" or "ü" are, unfortunately, treated as word breaks. |
| [\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:   * Before the first character of the word, if the first character is not a word character. * After the last character of the word, if the last character is not a word character. * Between two word characters * Between two non-word characters * The empty string   The beginning and end of a string are considered non-words.  For example, /\B../ matches 'oo' in "noonday", and /y\B./ matches 'ye' in "possibly yesterday." |
| [\c](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-control)*[X](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 single 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 single 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%." |
| [\](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-backreference)*[n](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-backreference)* | Where *n* is a positive integer, a back reference to the last substring matching the *n* parenthetical in the regular expression (counting left parentheses).  For example, /apple(,)\sorange\1/ matches 'apple, orange,' in "apple, orange, cherry, peach." |
| [\0](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-null) | Matches a NULL (U+0000) character. Do not follow this with another digit, because \0<digits> is an octal [escape sequence](https://developer.mozilla.org/en-US/docs/JavaScript/Guide/Values,_variables,_and_literals#Unicode_escape_sequences). Instead use \x00. |
| [\xhh](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-hex-escape) | Matches the character with the code hh (two hexadecimal digits) |
| [\uhhhh](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-unicode-escape) | Matches the character with the code hhhh (four hexadecimal digits). |
| [\u{hhhh}](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-unicode-escape-es6) | (only when u flag is set) Matches the character with the Unicode value hhhh (hexadecimal digits). |

Escaping user input that is to be treated as a literal string within a regular expression—that would otherwise be mistaken for a special character—can be accomplished by simple replacement:

function escapeRegExp(string) {

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

}

**Metodos**

| **haracter** | **Meaning** |
| --- | --- |
| [\](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#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. But a '\b' by itself doesn't match any character; it forms the special [word boundary character](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-word-boundary).  A backslash that precedes a special character indicates that the next character is not special and should be interpreted literally. For example, the pattern /a\*/ relies on the special character '\*' to match 0 or more a's. By contrast, the pattern /a\\*/ denotes the '\*'  as not special, enabling matches with strings like 'a\*'.  Do not forget to escape \ itself while using the RegExp("pattern") notation because \ is also an escape character in strings. |
| [^](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#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. See [complemented character sets](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "special-negated-character-set" \o "#special-negated-character-set) for details and an example. |
| [$](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-dollar) | Matches end of input. If the multiline flag is set to true, also matches immediately before a line break character.  For example, /t$/ 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#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#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#special-questionmark) | Matches the preceding expression 0 or 1 time. Equivalent to {0,1}.  For example, /e?le?/ matches the 'el' in "angel" and the 'le' in "angle" and also the 'l' in "oslo".  If used immediately after any of the quantifiers \*, +, ?, or {}, makes the quantifier non-greedy (matching the fewest possible characters), as opposed to the default, which is greedy (matching as many characters as possible). For example, applying /\d+/ to "123abc" matches "123". But applying /\d+?/ to that same string matches only the "1".  Also used in lookahead assertions, as described in the x(?=y) and x(?!y) entries of this table. |
| [.](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-dot) | (The decimal point) matches any single character except the newline character.  For example, /.n/ matches 'an' and 'on' in "nay, an apple is on the tree", but not 'nay'. |
| [(x)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#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 in the pattern match the string's last two words. Note that \1, \2, ..., \n are used in the matching part of the regex. 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. |
| [(?:x)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-non-capturing-parentheses) | Matches 'x' but does not remember the match. The parentheses are called non-capturing parentheses, and let you define subexpressions for regular expression operators to work with. Consider the sample expression /(?:foo){1,2}/. If the expression was /foo{1,2}/, the {1,2} characters would apply only to the last 'o' in 'foo'. With the non-capturing parentheses, the {1,2} applies to the entire word 'foo'. For more information, see [Using parentheses](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions" \l "Using_parentheses) below. |
| [x(?=y)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-lookahead) | Matches 'x' only if 'x' is followed by 'y'. This is called a lookahead.  For example, /Jack(?=Sprat)/ matches 'Jack' only if it is followed by 'Sprat'. /Jack(?=Sprat|Frost)/ matches 'Jack' only if it is followed by 'Sprat' or 'Frost'. However, neither 'Sprat' nor 'Frost' is part of the match results. |
| [x(?!y)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-negated-look-ahead) | Matches 'x' only if 'x' is not followed by 'y'. This is called a negated lookahead.  For example, /\d+(?!\.)/ matches a number only if it is not followed by a decimal point. The regular expression /\d+(?!\.)/.exec("3.141") matches '141' but not '3.141'. |
| [x|y](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-or) | Matches 'x', or 'y' (if there is no match for 'x').  For example, /green|red/ matches 'green' in "green apple" and 'red' in "red apple." The order of 'x' and 'y' matters. For example a\*|b matches the empty string in "b", but b|a\* matches "b" in the same string. |
| [{n}](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-quantifier) | Matches exactly n occurrences of the preceding expression. N must be a positive integer.  For example, /a{2}/ doesn't match the 'a' in "candy," but it does match all of the a's in "caandy," and the first two a's in "caaandy." |
| [{n,}](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-quantifier) | Matches at least n occurrences of the preceding expression. N must be a positive integer.  For example, /a{2,}/ will match "aa", "aaaa" and "aaaaa" but not "a" |
| [{n,m}](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-quantifier-range) | Where n and m are positive integers and n <= m. Matches at least n and at most m occurrences of the preceding expression. When m is omitted, it's treated as ∞.  For example, /a{1,3}/ matches nothing in "cndy", the 'a' in "candy," the first two a's in "caandy," and the first three a's in "caaaaaaandy". Notice that when matching "caaaaaaandy", the match is "aaa", even though the original string had more a's in it. |
| [[xyz]](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-character-set) | Character set. This pattern type matches any one of the characters in the brackets, including [escape sequences](https://developer.mozilla.org/en-US/docs/JavaScript/Guide/Values,_variables,_and_literals#Unicode_escape_sequences). Special characters like the dot(.) and asterisk (\*) are not special inside a character set, so they don't need to be escaped. You can specify a range of characters by using a hyphen, as the following examples illustrate.  The pattern [a-d], which performs the same match as [abcd], matches the 'b' in "brisket" and the 'c' in "city". The patterns /[a-z.]+/ and /[\w.]+/ match the entire string "test.i.ng". |
| [[^xyz]](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-negated-character-set) | A negated or complemented character set. That is, it matches anything that is not enclosed in the brackets. You can specify a range of characters by using a hyphen. Everything that works in the normal character set also works here.  For example, [^abc] is the same as [^a-c]. They initially match 'r' in "brisket" and 'h' in "chop." |
| [[\b]](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-backspace) | Matches a backspace (U+0008). You need to use square brackets if you want to match a literal backspace character. (Not to be confused with \b.) |
| [\b](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-word-boundary) | Matches a word boundary. A word boundary matches the position where a word character is not followed or preceded by another word-character. Note that a matched word boundary is not included in the match. In other words, the length of a matched word boundary is zero. (Not to be confused with [\b].)  Examples: /\bm/ matches the 'm' in "moon" ; /oo\b/ does not match the 'oo' in "moon", because 'oo' is followed by 'n' which is a word character; /oon\b/ matches the 'oon' in "moon", because 'oon' is the end of the string, thus not followed by a word character; /\w\b\w/ will never match anything, because a word character can never be followed by both a non-word and a word character.  **Note:** JavaScript's regular expression engine defines a [specific set of characters](http://www.ecma-international.org/ecma-262/5.1/#sec-15.10.2.6)to be "word" characters. Any character not in that set is considered a word break. This set of characters is fairly limited: it consists solely of the Roman alphabet in both upper- and lower-case, decimal digits, and the underscore character. Accented characters, such as "é" or "ü" are, unfortunately, treated as word breaks. |
| [\B](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-non-word-boundary) | Matches a non-word boundary. This matches the following cases:   * Before the first character of the word, if the first character is not a word character. * After the last character of the word, if the last character is not a word character. * Between two word characters * Between two non-word characters * The empty string   The beginning and end of a string are considered non-words.  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#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#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#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#special-form-feed) | Matches a form feed (U+000C). |
| [\n](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-line-feed) | Matches a line feed (U+000A). |
| [\r](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-carriage-return) | Matches a carriage return (U+000D). |
| [\s](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-white-space) | Matches a single 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#special-non-white-space) | Matches a single 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#special-tab) | Matches a tab (U+0009). |
| [\v](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-vertical-tab) | Matches a vertical tab (U+000B). |
| [\w](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#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#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%." |
| [\n](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-backreference) | Where n is a positive integer, a back reference to the last substring matching the n parenthetical in the regular expression (counting left parentheses).  For example, /apple(,)\sorange\1/ matches 'apple, orange,' in "apple, orange, cherry, peach." |
| [\0](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-null) | Matches a NULL (U+0000) character. Do not follow this with another digit, because \0<digits> is an octal [escape sequence](https://developer.mozilla.org/en-US/docs/JavaScript/Guide/Values,_variables,_and_literals#Unicode_escape_sequences). Instead use \x00. |
| [\xhh](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-hex-escape) | Matches the character with the code hh (two hexadecimal digits) |
| [\uhhhh](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-unicode-escape) | Matches the character with the code hhhh (four hexadecimal digits). |
| [\u{hhhh}](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#special-unicode-escape-es6) | (only when u flag is set) Matches the character with the Unicode value hhhh (hexadecimal digits). |

Escaping user input that is to be treated as a literal string within a regular expression—that would otherwise be mistaken for a special character—can be accomplished by simple replacement:

function escapeRegExp(string) {

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

}

**Ejemplos**

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

var str = 'John Smith';

var newstr = str.replace(re, '$2, $1');

console.log(newstr);

// "Smith, John"

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,

// the latter outputs Array [ "dbbd" ], while

// /d(b+)d/g.exec('cdbbdbsbz') outputs Array [ "dbbd", "bb" ].

// See below for further info (CTRL+F "The behavior associated with the".)