Assertions

Assertions include boundaries, which indicate the beginnings and endings of lines and words, and other patterns indicating in some way that a match is possible (including look-ahead, look-behind, and conditional expressions).

const text = 'A quick fox';

const regexpLastWord = /\w+$/;

console.log(text.match(regexpLastWord));

// Expected output: Array ["fox"]

const regexpWords = /\b\w+\b/g;

console.log(text.match(regexpWords));

// Expected output: Array ["A", "quick", "fox"]

const regexpFoxQuality = /\w+(?= fox)/;

console.log(text.match(regexpFoxQuality));

// Expected output: Array ["quick"]

[**Types**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Assertions#types)

[**Boundary-type assertions**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Assertions#boundary-type_assertions)

| **Characters** | **Meaning** |
| --- | --- |
| ^ | Matches the 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 first "A" in "An A".  **Note:** This character has a different meaning when it appears at the start of a [character class](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Character_classes). |
| $ | Matches the 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". |
| \b | Matches a word boundary. This is the position where a word character is not followed or preceded by another word-character, such as between a letter and a space. Note that a matched word boundary is not included in the match. In other words, the length of a matched word boundary is zero.  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.   To match a backspace character ([\b]), see [Character Classes](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Character_classes). |
| \B | Matches a non-word boundary. This is a position where the previous and next character are of the same type: Either both must be words, or both must be non-words, for example between two letters or between two spaces. The beginning and end of a string are considered non-words. Same as the matched word boundary, the matched non-word boundary is also not included in the match. For example, /\Bon/ matches "on" in "at noon", and /ye\B/ matches "ye" in "possibly yesterday". |

[**Other assertions**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Assertions#other_assertions)

**Note:** The ? character may also be used as a quantifier.

| **Characters** | **Meaning** |
| --- | --- |
| x(?=y) | **Lookahead assertion:**Matches "x" only if "x" is followed by "y". 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) | **Negative lookahead assertion:**Matches "x" only if "x" is not followed by "y". For example, /\d+(?!\.)/ matches a number only if it is not followed by a decimal point. /\d+(?!\.)/.exec('3.141') matches "141" but not "3". |
| (?<=y)x | **Lookbehind assertion:**Matches "x" only if "x" is preceded by "y". For example, /(?<=Jack)Sprat/ matches "Sprat" only if it is preceded by "Jack". /(?<=Jack|Tom)Sprat/ matches "Sprat" only if it is preceded by "Jack" or "Tom". However, neither "Jack" nor "Tom" is part of the match results. |
| (?<!y)x | **Negative lookbehind assertion:**Matches "x" only if "x" is not preceded by "y". For example, /(?<!-)\d+/ matches a number only if it is not preceded by a minus sign. /(?<!-)\d+/.exec('3') matches "3". /(?<!-)\d+/.exec('-3') match is not found because the number is preceded by the minus sign. |

[**Examples**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Assertions#examples)

[**General boundary-type overview example**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Assertions#general_boundary-type_overview_example)

JS

// Using Regex boundaries to fix buggy string.

buggyMultiline = `tey, ihe light-greon apple

tangs on ihe greon traa`;

// 1) Use ^ to fix the matching at the beginning of the string, and right after newline.

buggyMultiline = buggyMultiline.replace(/^t/gim, "h");

console.log(1, buggyMultiline); // fix 'tey' => 'hey' and 'tangs' => 'hangs' but do not touch 'traa'.

// 2) Use $ to fix matching at the end of the text.

buggyMultiline = buggyMultiline.replace(/aa$/gim, "ee.");

console.log(2, buggyMultiline); // fix 'traa' => 'tree.'.

// 3) Use \b to match characters right on border between a word and a space.

buggyMultiline = buggyMultiline.replace(/\bi/gim, "t");

console.log(3, buggyMultiline); // fix 'ihe' => 'the' but do not touch 'light'.

// 4) Use \B to match characters inside borders of an entity.

fixedMultiline = buggyMultiline.replace(/\Bo/gim, "e");

console.log(4, fixedMultiline); // fix 'greon' => 'green' but do not touch 'on'.

[**Matching the beginning of input using a ^ control character**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Assertions#matching_the_beginning_of_input_using_a_control_character)

Use ^ for matching at the beginning of input. In this example, we can get the fruits that start with 'A' by a /^A/ regex. For selecting appropriate fruits we can use the [filter](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter) method with an [arrow](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions) function.

JS

const fruits = ["Apple", "Watermelon", "Orange", "Avocado", "Strawberry"];

// Select fruits started with 'A' by /^A/ Regex.

// Here '^' control symbol used only in one role: Matching beginning of an input.

const fruitsStartsWithA = fruits.filter((fruit) => /^A/.test(fruit));

console.log(fruitsStartsWithA); // [ 'Apple', 'Avocado' ]

In the second example ^ is used both for matching at the beginning of input and for creating negated or complemented character class when used within [character classes](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Character_classes).

JS

const fruits = ["Apple", "Watermelon", "Orange", "Avocado", "Strawberry"];

// Selecting fruits that do not start by 'A' with a /^[^A]/ regex.

// In this example, two meanings of '^' control symbol are represented:

// 1) Matching beginning of the input

// 2) A negated or complemented character class: [^A]

// That is, it matches anything that is not enclosed in the brackets.

const fruitsStartsWithNotA = fruits.filter((fruit) => /^[^A]/.test(fruit));

console.log(fruitsStartsWithNotA); // [ 'Watermelon', 'Orange', 'Strawberry' ]

[**Matching a word boundary**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Assertions#matching_a_word_boundary)

JS

const fruitsWithDescription = ["Red apple", "Orange orange", "Green Avocado"];

// Select descriptions that contains 'en' or 'ed' words endings:

const enEdSelection = fruitsWithDescription.filter((descr) =>

/(en|ed)\b/.test(descr),

);

console.log(enEdSelection); // [ 'Red apple', 'Green Avocado' ]

[**Lookahead assertion**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Assertions#lookahead_assertion)

JSCopy to Clipboard

// JS Lookahead assertion x(?=y)

const regex = /First(?= test)/g;

console.log("First test".match(regex)); // [ 'First' ]

console.log("First peach".match(regex)); // null

console.log("This is a First test in a year.".match(regex)); // [ 'First' ]

console.log("This is a First peach in a month.".match(regex)); // null

[**Basic negative lookahead assertion**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Assertions#basic_negative_lookahead_assertion)

For example, /\d+(?!\.)/ matches a number only if it is not followed by a decimal point. /\d+(?!\.)/.exec('3.141') matches "141" but not "3.

JS

console.log(/\d+(?!\.)/g.exec("3.141")); // [ '141', index: 2, input: '3.141' ]

[**Different meaning of '?!' combination usage in assertions and character classes**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Assertions#different_meaning_of_!_combination_usage_in_assertions_and_character_classes)

The ?! combination has different meanings in assertions like /x(?!y)/ and [character classes](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Character_classes) like [^?!].

JS

const orangeNotLemon =

"Do you want to have an orange? Yes, I do not want to have a lemon!";

// Different meaning of '?!' combination usage in Assertions /x(?!y)/ and Ranges /[^?!]/

const selectNotLemonRegex = /[^?!]+have(?! a lemon)[^?!]+[?!]/gi;

console.log(orangeNotLemon.match(selectNotLemonRegex)); // [ 'Do you want to have an orange?' ]

const selectNotOrangeRegex = /[^?!]+have(?! an orange)[^?!]+[?!]/gi;

console.log(orangeNotLemon.match(selectNotOrangeRegex)); // [ ' Yes, I do not want to have a lemon!' ]