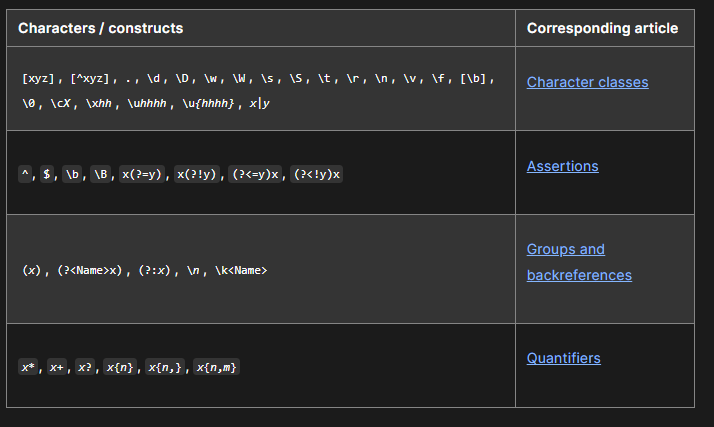
Character classes



Character classes distinguish kinds of characters such as, for example, distinguishing between letters and digits.

const chessStory = 'He played the King in a8 and she moved her Queen in c2.';

const regexpCoordinates = /\w\d/g;

console.log(chessStory.match(regexpCoordinates));

// Expected output: Array [ 'a8', 'c2']

const moods = 'happy 🙂, confused 😕, sad 😢';

const regexpEmoticons = /[\u{1F600}-\u{1F64F}]/gu;

console.log(moods.match(regexpEmoticons));

// Expected output: Array ['🙂', '😕', '😢']

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

| **Characters** | **Meaning** |
| --- | --- |
| [xyz] [a-c] | A character class. Matches any one of the enclosed characters. You can specify a range of characters by using a hyphen, but if the hyphen appears as the first or last character enclosed in the square brackets, it is taken as a literal hyphen to be included in the character class as a normal character.  For example, [abcd] is the same as [a-d]. They match the "b" in "brisket", and the "c" in "chop".  For example, [abcd-] and [-abcd] match the "b" in "brisket", the "c" in "chop", and the "-" (hyphen) in "non-profit".  For example, [\w-] is the same as [A-Za-z0-9\_-]. They both match the "b" in "brisket", the "c" in "chop", and the "n" in "non-profit". |
| [^xyz] [^a-c] | A negated or complemented character class. That is, it matches anything that is not enclosed in the brackets. You can specify a range of characters by using a hyphen, but if the hyphen appears as the first character after the ^ or the last character enclosed in the square brackets, it is taken as a literal hyphen to be included in the character class as a normal character. For example, [^abc] is the same as [^a-c]. They initially match "o" in "bacon" and "h" in "chop".  **Note:** The ^ character may also indicate the [beginning of input](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Assertions). |
| . | Has one of the following meanings:   * Matches any single character *except* line terminators: \n, \r, \u2028 or \u2029. For example, /.y/ matches "my" and "ay", but not "yes", in "yes make my day", as there is no character before "y" in "yes". * Inside a character class, the dot loses its special meaning and matches a literal dot.   Note that the m multiline flag doesn't change the dot behavior. So to match a pattern across multiple lines, the character class [^] can be used — it will match any character including newlines.  The s "dotAll" flag allows the dot to also match line terminators. |
| \d | Matches any digit (Arabic numeral). Equivalent to [0-9]. For example, /\d/ or /[0-9]/ matches "2" in "B2 is the suite number". |
| \D | Matches any character that is not a digit (Arabic numeral). Equivalent to [^0-9]. For example, /\D/ or /[^0-9]/ matches "B" in "B2 is the suite number". |
| \w | Matches any alphanumeric character from the basic Latin alphabet, including the underscore. Equivalent to [A-Za-z0-9\_]. For example, /\w/ matches "a" in "apple", "5" in "$5.28", "3" in "3D" and "m" in "Émanuel". |
| \W | Matches any character that is not a word character from the basic Latin alphabet. Equivalent to [^A-Za-z0-9\_]. For example, /\W/ or /[^A-Za-z0-9\_]/ matches "%" in "50%" and "É" in "Émanuel". |
| \s | Matches a single white space character, including space, tab, form feed, line feed, and other Unicode spaces. Equivalent to [\f\n\r\t\v\u0020\u00a0\u1680\u2000-\u200a\u2028\u2029\u202f\u205f\u3000\ufeff]. For example, /\s\w\*/ matches " bar" in "foo bar". |
| \S | Matches a single character other than white space. Equivalent to [^\f\n\r\t\v\u0020\u00a0\u1680\u2000-\u200a\u2028\u2029\u202f\u205f\u3000\ufeff]. For example, /\S\w\*/ matches "foo" in "foo bar". |
| \t | Matches a horizontal tab. |
| \r | Matches a carriage return. |
| \n | Matches a linefeed. |
| \v | Matches a vertical tab. |
| \f | Matches a form-feed. |
| [\b] | Matches a backspace. If you're looking for the word-boundary character (\b), see [Assertions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Assertions). |
| \0 | Matches a NUL character. Do not follow this with another digit. |
| \c*X* | Matches a control character using [caret notation](https://en.wikipedia.org/wiki/Caret_notation), where "X" is a letter from A–Z (corresponding to code points U+0001*–*U+001A). For example, /\cM\cJ/ matches "\r\n". |
| \x*hh* | Matches the character with the code *hh* (two hexadecimal digits). |
| \u*hhhh* | Matches a UTF-16 code-unit with the value *hhhh* (four hexadecimal digits). |
| \u*{hhhh}* or *\u{hhhhh}* | (Only when the u flag is set.) Matches the character with the Unicode value U+*hhhh* or U+*hhhhh* (hexadecimal digits). |
| \p{*UnicodeProperty*},  \P{*UnicodeProperty*} | Matches a character based on its [Unicode character properties](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Regular_expressions/Unicode_character_class_escape) (to match just, for example, emoji characters, or Japanese *katakana* characters, or Chinese/Japanese Han/Kanji characters, etc.). |
| \ | Indicates that the following character should be treated specially, or "escaped". It behaves one of two ways.   * For characters that are usually treated literally, indicates that the next character is special and not to be interpreted literally. For example, /b/ matches the character "b". By placing a backslash in front of "b", that is by using /\b/, the character becomes special to mean match a word boundary. * For characters that are usually treated specially, indicates that the next character is not special and should be interpreted literally. For example, "\*" is a special character that means 0 or more occurrences of the preceding character should be matched; for example, /a\*/ means match 0 or more "a"s. To match \* literally, precede it with a backslash; for example, /a\\*/ matches "a\*".   **Note:** To match this character literally, escape it with itself. In other words to search for \ use /\\/. |
| *x*|*y* | **Disjunction:**Matches either "x" or "y". Each component, separated by a pipe (|), is called an *alternative*. For example, /green|red/ matches "green" in "green apple" and "red" in "red apple".  **Note:** A disjunction is another way to specify "a set of choices", but it's not a character class. Disjunctions are not atoms — you need to use a [group](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Groups_and_backreferences) to make it part of a bigger pattern. [abc] is functionally equivalent to (?:a|b|c). |

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

[**Looking for a series of digits**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Character_classes#looking_for_a_series_of_digits)

JS

const randomData = "015 354 8787 687351 3512 8735";

const regexpFourDigits = /\b\d{4}\b/g;

// \b indicates a boundary (i.e. do not start matching in the middle of a word)

// \d{4} indicates a digit, four times

// \b indicates another boundary (i.e. do not end matching in the middle of a word)

console.table(randomData.match(regexpFourDigits));

// ['8787', '3512', '8735']

[**Looking for a word (from the latin alphabet) starting with A**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Character_classes#looking_for_a_word_from_the_latin_alphabet_starting_with_a)

JS

const aliceExcerpt =

"I'm sure I'm not Ada,' she said, 'for her hair goes in such long ringlets, and mine doesn't go in ringlets at all.";

const regexpWordStartingWithA = /\b[aA]\w+/g;

// \b indicates a boundary (i.e. do not start matching in the middle of a word)

// [aA] indicates the letter a or A

// \w+ indicates any character \*from the latin alphabet\*, multiple times

console.table(aliceExcerpt.match(regexpWordStartingWithA));

// ['Ada', 'and', 'at', 'all']

[**Looking for a word (from Unicode characters)**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Character_classes#looking_for_a_word_from_unicode_characters)

Instead of the Latin alphabet, we can use a range of Unicode characters to identify a word (thus being able to deal with text in other languages like Russian or Arabic). The "Basic Multilingual Plane" of Unicode contains most of the characters used around the world and we can use character classes and ranges to match words written with those characters.

JS

const nonEnglishText = "Приключения Алисы в Стране чудес";

const regexpBMPWord = /([\u0000-\u0019\u0021-\uFFFF])+/gu;

// BMP goes through U+0000 to U+FFFF but space is U+0020

console.table(nonEnglishText.match(regexpBMPWord));

["Приключения", "Алисы", "в", "Стране", "чудес"];

[**Counting vowels**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Character_classes#counting_vowels)

JS

const aliceExcerpt =

"There was a long silence after this, and Alice could only hear whispers now and then.";

const regexpVowels = /[AEIOUYaeiouy]/g;

console.log("Number of vowels:", aliceExcerpt.match(regexpVowels).length);

// Number of vowels: 26