Regular expression syntax cheatsheet

This page provides an overall cheat sheet of all the capabilities of `RegExp` syntax by aggregating the content of the articles in the `RegExp` guide. If you need more information on a specific topic, please follow the link on the corresponding heading to access the full article or head to the guide.

Character classes

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

Characters	Meaning
`.`	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". • 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. ES2018 added the `s` "dotAll" flag, which 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", and "3" in "3D".
`\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%".
`\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\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\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 Boundaries.
`\0`	Matches a NUL character. Do not follow this with another digit.
`\c`*X*	Matches a control character using caret notation, where "X" is a letter from A–Z (corresponding to codepoints `U+0001` – `U+001F`). For example, `/\cM/` matches "\r" in "\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).
`\`	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 that some characters like `:` , `-` , `@` , etc. neither have a special meaning when escaped nor when unescaped. Escape sequences like `\:` , `\-` , `\@` will be equivalent to their literal, unescaped character equivalents in regular expressions. However, in regular expressions with the unicode flag, these will cause an *invalid identity escape* error. This is done to ensure backward compatibility with existing code that uses new escape sequences like `\p` or `\k` . **Note:** To match this character literally, escape it with itself. In other words to search for `\` use `/\\/` .

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).

Boundary-type assertions

Characters	Meaning

`\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.
`\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

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.

Characters	Meaning
x|*y*	Matches either "x" or "y". For example, `/green|red/` matches "green" in "green apple" and "red" in "red apple".

`[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 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, `[^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.

($1, ..., $9).

`(?:x)`	**Non-capturing group:** Matches "x" but does not remember the match. The matched substring cannot be recalled from the resulting array's elements (`[1], ..., [n]`) or from the predefined `RegExp` object's properties (`$1, ..., $9`).

Quantifiers

Quantifiers indicate numbers of characters or expressions to match.

> **Note:** In the following, *item* refers not only to singular characters, but also includes character classes, Unicode property escapes, groups and ranges.

```
// Non-binary values
\p{UnicodePropertyValue}
\p{UnicodePropertyName=UnicodePropertyValue}

// Binary and non-binary values
\p{UnicodeBinaryPropertyName}
```