**Groups and backreferences**

Groups group multiple patterns as a whole, and capturing groups provide extra sub match information when using a regular expression pattern to match against a string. Backreferences refer to a previously captured group in the same regular expression.

const aliceExcerpt = 'The Caterpillar and Alice looked at each other';

const regexpWithoutE = /\b[a-df-z]+\b/gi;

console.log(aliceExcerpt.match(regexpWithoutE));

// Expected output: Array ["and", "at"]

const imageDescription = 'This image has a resolution of 1440×900 pixels.';

const regexpSize = /([0-9]+)×([0-9]+)/;

const match = imageDescription.match(regexpSize);

console.log(`Width: ${match[1]} / Height: ${match[2]}.`);

// Expected output: "Width: 1440 / Height: 900."

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

| **Characters** | **Meaning** |
| --- | --- |
| (*x*) | **Capturing group:**Matches *x* and remembers the match. For example, /(foo)/ matches and remembers "foo" in "foo bar".  A regular expression may have multiple capturing groups. In results, matches to capturing groups typically in an array whose members are in the same order as the left parentheses in the capturing group. This is usually just the order of the capturing groups themselves. This becomes important when capturing groups are nested. Matches are accessed using the index of the result's elements ([1], …, [n]) or from the predefined RegExp object's properties ($1, …, $9).  Capturing groups have a performance penalty. If you don't need the matched substring to be recalled, prefer non-capturing parentheses (see below).  [String.prototype.match()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/match) won't return groups if the /.../g flag is set. However, you can still use [String.prototype.matchAll()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/matchAll) to get all matches. |
| (?<Name>x) | **Named capturing group:**Matches "x" and stores it on the groups property of the returned matches under the name specified by <Name>. The angle brackets (< and >) are required for group name.  For example, to extract the United States area code from a phone number, we could use /\((?<area>\d\d\d)\)/. The resulting number would appear under matches.groups.area. |
| (?:*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). |
| \*n* | 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". |
| \k<Name> | A back reference to the last substring matching the **Named capture group** specified by <Name>.  For example, /(?<title>\w+), yes \k<title>/ matches "Sir, yes Sir" in "Do you copy? Sir, yes Sir!".  **Note:** \k is used literally here to indicate the beginning of a back reference to a Named capture group. |

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

[**Using groups**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Groups_and_backreferences#using_groups)

JS

const personList = `First\_Name: John, Last\_Name: Doe

First\_Name: Jane, Last\_Name: Smith`;

const regexpNames = /First\_Name: (\w+), Last\_Name: (\w+)/gm;

for (const match of personList.matchAll(regexpNames)) {

console.log(`Hello ${match[1]} ${match[2]}`);

}

[**Using named groups**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Groups_and_backreferences#using_named_groups)

JS

const personList = `First\_Name: John, Last\_Name: Doe

First\_Name: Jane, Last\_Name: Smith`;

const regexpNames =

/First\_Name: (?<firstname>\w+), Last\_Name: (?<lastname>\w+)/gm;

for (const match of personList.matchAll(regexpNames)) {

console.log(`Hello ${match.groups.firstname} ${match.groups.lastname}`);

}

[**Using groups and back references**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Groups_and_backreferences#using_groups_and_back_references)

JS

const quote = `Single quote "'" and double quote '"'`;

const regexpQuotes = /(['"]).\*?\1/g;

for (const match of quote.matchAll(regexpQuotes)) {

console.log(match[0]);

}

[**Using groups and match indices**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Groups_and_backreferences#using_groups_and_match_indices)

By providing the d flag, the indices of each capturing group is returned. This is especially useful if you are correlating each matched group with the original text — for example, to provide compiler diagnostics.

JS

const code = `function add(x, y) {

return x + y;

}`;

const functionRegexp =

/(function\s+)(?<name>[$\_\p{ID\_Start}][$\u200c\u200d\p{ID\_Continue}]\*)/du;

const match = functionRegexp.exec(code);

const lines = code.split("\n");

lines.splice(

1,

0,

" ".repeat(match.indices[1][1] - match.indices[1][0]) +

"^".repeat(match.indices.groups.name[1] - match.indices.groups.name[0]),

);

console.log(lines.join("\n"));

// function add(x, y) {

// ^^^

// return x + y;

// }