**https://javascript.info/regexp-methods**

**Character groups**

**Character set [xyz] —** if target position matches any of these characters, return true:

var regex = /[bt]ear/;

console.log(regex.test('tear')); // returns true

**S**pecial characters lose their special meaning inside this set, except for caret (^) which gains new meaning:

**Negated character set [^xyz] —** if target position contains any of these characters, return false:

var regex /[^bt]ear/;

console.log(regex.test('tear'));  
// returns false  
console.log(regex.test('fear'));  
// return true

**Ranges [a-z] — if target position contains characters within this range, return true.**  For example: **[a-h]** will match all the letters from a to h. Ranges can also be digits like **[0-9]** or capital letters like **[A-Z]**.

var regex = /[a-z]ear/;

console.log(regex.test('fear'));  
// returns true

console.log(regex.test('tear'));  
// returns true

**Meta-characters**

These have special meanings:

* **\d** — Match any digit ( same as [0-9] ).
* **\w** — Match any word character (any letter, digit, and underscore). (Same as [a-zA-Z0–9\_] ) i.e alphanumeric character.
* **\s**— Match a whitespace character (spaces, tabs etc).
* **\t**— Match a tab character only.
* **\b**— Find a match at beginning or ending of a word. Also known as word boundary.
* **.** — (period) Matches any character except for newline.
* **\D** — Match any non digit character (same as [^0–9]).
* **\W** — Match any non word character (Same as [^a-zA-Z0–9\_] ).
* **\S** — Match a non whitespace character.

var str = "Give 100%!";  
var patt1 = /\w/g;

returns these items: G,i,v,e,1,0,0

**Quantifiers: —**Quantifiers are symbols that have a special meaning in a regular expression.

* **+** — Matches the preceding expression 1 or more times.

var regex = /\d+/;  
console.log(regex.test('8'));  
// true

console.log(regex.test('88899'));  
// true

console.log(regex.test('8888845'));  
// true

* **\***—Matches the preceding expression 0 or more times.

var regex = /go\*d/;

console.log(regex.test('gd'));  
// true

console.log(regex.test('god'));  
// true

console.log(regex.test('good'));  
// true

console.log(regex.test('goood'));  
// true

* **?**— Matches the preceding expression 0 or 1 time, that is preceding pattern is optional.

var regex = /goo?d/;

console.log(regex.test('god'));  
// true

console.log(regex.test('good'));  
// true

console.log(regex.test('goood'));  
// false

* **^**— Matches the beginning of the string, the regular expression that follows it should be at the start of the test string. i.e the caret (^) matches the start of string.

var regex = /^g/;

console.log(regex.test('good'));  
// true

console.log(regex.test('bad'));  
// false

console.log(regex.test('tag'));  
// false

* **$** — Matches the end of the string, that is the regular expression that precedes it should be at the end of the test string. The dollar ($) sign matches the end of the string.

var regex = /.com$/;

console.log(regex.test('test@testmail.com'));  
// true

console.log(regex.test('test@testmail'));  
// false

* **{N}** — Matches exactly N occurrences of the preceding regular expression.

var regex = /go{2}d/;

console.log(regex.test('good'));  
// true

console.log(regex.test('god'));  
// false

* **{N,}** — Matches at least N occurrences of the preceding regular expression.

var regex = /go{2,}d/;

console.log(regex.test('good'));  
// true

console.log(regex.test('goood'));  
// true

console.log(regex.test('gooood'));  
// true

* **{N,M}** — Matches at least N occurrences and at most M occurrences of the preceding regular expression (where M > N).

var regex = /go{1,2}d/;

console.log(regex.test('god'));  
// true

console.log(regex.test('good'));  
// true

console.log(regex.test('goood'));  
// false

**Alternation X|Y**— Matches either X or Y. For example:

var regex = /(green|red) apple/;

console.log(regex.test('green apple'));  
// true  
console.log(regex.test('red apple'));  
// true  
console.log(regex.test('blue apple'));  
// false

**Note**— If you want to use any special character as a part of the expression, say for example you want to match literal + or ., then you have to escape them with backslash ( \ ).

For example:

var regex = /a+b/; // This won't work

var regex = /a\+b/; // This will work

console.log(regex.test('a+b')); // true

**Advanced**

**(x)** — Matches x and remembers the match. These are called capturing groups. This is also used to create sub expressions within a regular expression. For example :-

var regex = /(foo)bar\1/;  
console.log(regex.test('foobarfoo'));  
// true

console.log(regex.test('foobar'));  
// false

\1 remembers and uses that match from first subexpression within parentheses.

**(?:x)**— Matches x and does not remember the match. These are called non capturing groups. Here \1 won’t work, it will match the literal \1.

var regex = /(?:foo)bar\1/;  
console.log(regex.test('foobarfoo'));  
// false

console.log(regex.test('foobar'));  
// false

console.log(regex.test('foobar\1'));  
// true

**x(?=y)** — Matches x only if x is followed by y. Also called positive look ahead. For example:

var regex = /Red(?=Apple)/;

console.log(regex.test('RedApple'));  
// true

In the above example, match will occur only if Redis followed by Apple.