

Regular Expressions for client-side JavaScript a free online quick reference by [VisiBone](#)

// Regular expressions are **patterns**
// used to match portions of strings

/  **/** // All the code here checks itself
assert(/hello/.test("hello world"));
// using Edward Hieatt's JsUnit
// from www.jsunit.net

// The **test** method says whether there's a match anywhere.
assert(!/i/.test("courage")); // there is no "i" in courage
assert(/our/.test("courage")); // there is "our" in courage

// The **search** method says how many characters precede.
assert("courage".search(/our/) == 1); // 1 letter before "our"
assert("courage".search(/i/) == -1); // -1 means no match

// The **replace** method changes a matched substring
assert("recieve".replace(/ie/, "ei") == "receive");

character matching one-for-one:

A ... Z **a ... z** **0 ... 9** **** 
alphanumeric punctuation

\u0000 ... \uFFFF
Unicode hexadecimal

\x00 ... \xFF
ASCII hexadecimal

\cA ... \cZ
control characters

\0 ... \7
\00 ... \77
\000 ... \377
ASCII Octal

\0 **[\b]** **\t** **\v** **\f** **\r** **\n**
NUL back space TAB VT FF CR new
line
\x00 **\x08** **\x09** **\x0B** **\x0C** **\x0D** **\x0A**

Character Classes (sets of matchables)

\d **\D** // **/\d/** matches any decimal digit
digit non-digit
assert(/**\d**/.test("9"));
// same as **/[0123456789]/** or **/[0-9]/**
assert(/**[0123456789]**/.test("9"));
assert(/**[0-9]**/.test("9"));

\s **\S** // **/\s/** matches space, tab, terminators
space non-space
assert(/**\S\S\S\S\S\S**/.test("to be"));
// any invisible "white space" character
assert(/**\s**/.test(" "));
assert(/**[\t\n\u000B\f\r]**/.test(" "));

\w **\W** // **/\w/** is a letter, number, underscore
assert(/**\w\w\W\w\w\W\w\w**/
test("21-Mav/02"));

I hope you find these excerpts of the VisiBone JavaScript references very useful.

See also the [JavaScript Card and Foldouts](#)

Here is the syntax for a very powerful and very cryptic string pattern matching scheme in the **client-side JavaScript** of web browsers.

You can use it to validate form entry, parse URLs, and many other things.

The information here forms a page of the **JavaScript Card**:



and is one of the set of three **JavaScript Foldouts**:



[Feedback form below!](#)

or email stein@visibone.com.

**word non-
char word**



one of



one not of



range

```
assert(/\w/.test('X'));  
assert(/[0-9A-Za-z_]/.test('X'));
```

```
var vowel = /[aeiou]/;  
assertEquals(2, 'story'.search(vowel));  
assert(!vowel.test('mfgR'));
```

```
var nonvowel = /^[^aeiou]/;  
assert(nonvowel.test('our'));  
assert(!nonvowel.test('eye'));
```

```
assert(/[a-z]/.test('Story'));  
assert('$8ea'.replace(/[0-9]/, 'X') == '$Xea');  
assert(/[^\a-zA-Z]/.test('Yes?'));
```

characters inside class [brackets]



alphanumeric



**punc-
tuation**



Unicode hexadecimal



ASCII hexadecimal



control characters



ASCII Octal



NUL **back** **TAB** **VT** **FF** **CR** **new**
space
line
\x00 **\x08** **\x09** **\x0B** **\x0C** **\x0D** **\x0A**



digit **non-
digit** **space** **non-
space** **word
char** **non-
word**

```
// use digit, word or space in or outside class [ brackets ]  
assert(/^[\d\s\(\)\-\+\,]*$/i.test('(+20) 2 0900 0700'));  
assert(/^[\d\s\(\)\-\+\,]*$/i.test('714/921-5424'));  
assert(!/^[\d\s\(\)\-\+\,]*$/i.test('1-866-4MORTAL'));
```

// Slashed and Confused:

Backspace: `/[\b]/` or `'\b'` Not: `/\b/` (word boundary)

Vertical tab: `/\v/` or `/[\v]/` Not: `'\v'` (same as 'v')

**// Think of regular expressions like Lego® building blocks.
// They make sense once you envision how all the pieces fit.**



either - or

```
assert(/jpg|jpeg/.test('pic.jpg'));  
assert(/jpg|jpeg/.test('pic.jpeg'));  
assert(!/jpg|jpeg/.test('pic.gif'));
```




any character
(except newline 🐛)

```
assert(/g.t/.test('get')); get
assert(/g.t/.test('night')); night
assert(!(/g.t/.test('goat'))); goat
assert((/g.t/.test('g\\t')));
assert(!(/g.t/.test('g\\nt')));
```

Repeaters:



optional (0 or 1)

```
assert(/to?t/.test('tt'));
assert(/to?t/.test('tot'));
assert(!/to?t/.test('toot'));
assert(!/to?t/.test('tooot'));
```



any (0 or more)

```
assert(/to*t/.test('tt'));
assert(/to*t/.test('tot'));
assert(/to*t/.test('toot'));
assert(/to*t/.test('tooot'));
```



etc. (1 or more)

```
assert(!/to+t/.test('tt'));
assert(/to+t/.test('tot'));
assert(/to+t/.test('toot'));
assert(/to+t/.test('tooot'));
```



exactly (n)

```
assert(!/to{2}t/.test('tt'));
assert(!/to{2}t/.test('tot'));
assert(/to{2}t/.test('toot'));
assert(!/to{2}t/.test('tooot'));
```



min (n or more)

```
assert(!/to{2,}t/.test('tt'));
assert(!/to{2,}t/.test('tot'));
assert(/to{2,}t/.test('toot'));
assert(/to{2,}t/.test('tooot'));
```



range (n to m)

```
assert(!/to{1,2}t/.test('tt'));
assert(/to{1,2}t/.test('tot'));
assert(/to{1,2}t/.test('toot'));
assert(!/to{1,2}t/.test('tooot'));
```

a subexpression groups or captures contents



// **groups** characters for a repeater:

```
assert(/friend(ship)?/.test('friend'));
assert(/friend(ship)?/.test('friendship'));
```

// or **captures** submatched characters for later recall:

```
assert("abc".replace(/(a)(b)(c)/, '$1.$2.$3') == "a.b.c");
assert("in 206BC is".replace(/(\d+)BC/, "<i>$1</i> BC")
== "in <i>206</i> BC is");
```

// only plain () parens capture, not (? :) nor (? =) nor (? !)



a passive subexpression
only groups its contents

```
assert("abc".replace(/(a)(b)(c)/, '$1.$2') == "a.b");
assert("abc".replace(/(a)(?:b)(c)/, '$1.$2') == "a.c");
```

Flags:



```
assert('aardvark'.replace(/a/, 'o') == 'oardvark');
assert('aardvark'.replace(/a/g, 'o') == 'oordvork');
```



```
assert('Aardvark'.replace(/a/, 'e') == 'Aerdvark');
assert('Aardvark'.replace(/a/i, 'e') == 'eardvark');
assert('Aardvark'.replace(/a/gi, 'e') == 'eerdverk');
```



```
assert('ten\nton\ntin'.replace(/n$/, 'x'))
```

 **/m**

```
== 'ten\nton\ntix');
assert('ten\nton\ntin'.replace(/n$/m,'x'))
== 'tex\nton\ntin');
assert('ten\nton\ntin'.replace(/n$/gm,'x'))
== 'tex\ntox\ntix');
```

Anchors (matches between characters)



starts with

```
assert(/^a/.test('apple'));
assert(/^a/.test('alpha'));
assert(!/^a/.test('baker'));
```



ends with

```
assert(/e$/.test('apple'));
assert(!/e$/.test('baker'));
assert(/e$/.test('charlie'));
```



word boundary



not a word boundary, inside a word

// "words" are consecutive letters, numbers or underscores
assert(/I am/.test("I am. ")); // same as what matches \w
assert(/\bI\b\b ba\bB\b/.test("I am. "));



look-ahead equals
match on what comes next

```
assert("abc".replace(/ab(c)/,'-')v'-');
assert("abc".replace(/ab(?=c)/,'-') == "-c");
```



look-ahead not-equals
match on what isn't next

```
assert("abc".replace(/ab[ ^d]/,'-') == "-");
assert("abc".replace(/ab(?!d)/,'-') == "-c");
```

// Most regular expressions **find** something in a string
// of text, or **change** the text into something more useful.

simple methods:

b = r.test(s);

```
assert(/a/.test("apple"));
assert(/e/.test("apple"));
assert(!/x/.test("apple"));
```

n = s.search(r);

```
assert("apple".search(/a/) == 0);
assert("apple".search(/pp/) == 1);
assert("apple".search(/l/) == 3);
assert("apple".search(/e/) == 4);
assert("apple".search(/x/) == -1);
```

s = s.replace(r,s);

```
assert("alan".replace(/a/,"A") == "Alan");
assert("alan".replace(/a/,"e") == "elan");
assert("alan".replace(/a/g,"e") == "elen");
```

replace()'s special symbols:

"\$` \$& \$(' \$ _ \$1 \$2 ... \$+ \$\$"

// symbols for any regular expression

```
assert("abcd".replace(/bc,$`) == "aad"); // leftContext
assert("abcd".replace(/bc,$&') == "abcd"); // lastMatch
assert("abcd".replace(/bc,$'') == "add"); // rightContext
assert("abcd".replace(/bc,$_) == "aabcd"); // input
assert("abcd".replace(/bc,$$) == "a$d"); // (literal $)
```

// symbols recalling subexpression-matched characters:

```
assert("abcd".replace(/(b)(c)/,$1) == "abd"); // $1
assert("abcd".replace(/(b)(c)/,$2) == "acd"); // $2
assert("abcd".replace(/(b)(c)/,$+) == "acd"); // lastParen
```

1 **2** ...

```
var s = "the the first man to to see";
var r = /\b(\w+) \1\b/g;
assertEquals(s.replace(r,$1 [$1]),
  "the [the] first man to [to] see");
submatch recur
insist that what matched a subexpression appear again
```

advanced methods:

a = s.split(rd);

Delimiter breakdown

```
var s = "to be the *first* -- that is the idea"; // will parse by
var a = s.split(/\W+/); // any series of non-word characters
assert(a.join() == "to,be,the,first,that,is,the,idea");
var a = s.split(/\W+/,4); // split can also limit array length
assert(a.join() == "to,be,the,first");
var a = "to be first".split(/(\W+)/); // someday split() may
assert(a.join() != "to ,be ,first"); // include submatches
```

a = s.match(rg);

Global breakdown

```
var sBingo = "Calling b7 i20 n33, anybody win yet?";
// extract bingo codes: one bingo-letter plus 1 or 2 digits
var ag = sBingo.match(/[bingo]\d{1,2}/g);
assert(ag.length == 3); // a match on a g-option expression
assert(ag[0] == 'b7'); // returns an array of the
assert(ag[1] == 'i20'); // substrings that matched
assert(ag[2] == 'n33');
assert(ag.input == sBingo);
```

a = s.match(rp);

Parenthetical breakdown

```
// (subexpression 1: bingo-letter) (subexpression 2: digits)
var ap = ag[2].match(/([bingo])(\d{1,2})/);
assert(ap.length == 3); // non-g match array very different:
assert(ap[0] == 'n33'); // 0: entire matched string
assert(ap[1] == 'n'); // 1: 1st subexpression's match
assert(ap[2] == '33'); // 2: 2nd subexpression's match
assert(ap.index == 0);
assert(ap.input == ag[2]);
```

a = r.exec(s);

Parenthetical and global breakdown

```
// Without the g option, r.exec(s) is identical to s.match(rp).
// With the g option, something very unusual happens.
// Each call to exec() matches the entire expression once
// (as if there were no g option) and breaks it down
// parenthetically (as s.match(rp)) but it picks up where the
// last call left off. So repeatedly calling exec() until null
// gets every match the g-option would normally have hit.
RegExp.lastIndex = 0; // (☹️ wise to do before first exec)
var r = /[([bingo])(\d{1,2})/g;
var a = r.exec(sBingo); // first exec breaks down first match
```

```

assert(a.join() == "b7,b,7");
a=r.exec(sBingo); // next exec breaks down next match
assert(a.join() == "i20,i,20");
a=r.exec(sBingo); // and so on until...
assert(a.join() == "n33,n,33");
a=r.exec(sBingo); // ...there are no more matches
assert(a == null);

// Here's exec() flexing its muscle in a 2D loop:
var s="";
RegExp.lastIndex=0; // (less grief: always start with this)
while ((a=r.exec(sBingo)) != null) {
  for (var i=1; i < a.length; i++) { // (skipping a[0])
    s+=a[i]+'';
  }
  s+='/';
}
assertEquals(s,"b 7 / i 20 / n 33 / ");

```

// Most regular expressions are somewhere
// between dirt simple and fiendishly complex.

r = new RegExp(s);
r = new RegExp(s,sFlags);
s = r.source;

Constructors

☛ don't
forget
to use new

```

var r1=/e/g; // Three ways to
var r2=new RegExp("e","g"); // make a regular
var r3=eval("/e/g"); // expression
assert(r1 != r2); // ☛ Compares by reference
assert("meme".replace(r1,"i") == "mimi");
assert("meme".replace(r2,"i") == "mimi");
assert("meme".replace(r3,"i") == "mimi");
var r=new RegExp("\\d\\w"); // Double backslashes, since
assert(r.test("3D")); // strings use them for escaping too
var r=new RegExp("\\\\"); // Quadruple literal backslashes
assert(r.test("\\u005C")); // string-doubling, regexp-doubling
assert(r1.source == "e");
assert(r2.source == "e");

```

s = RegExp.leftContext;
s = RegExp.lastMatch;
s = RegExp.rightContext;
s = RegExp.input;
n = RegExp.index;

side effects
of the most
recent regular
expression
(that made
a match)

```

/and/.test('to-and-fro');
assert(RegExp.leftContext == 'to-');
assert(RegExp.lastMatch == 'and');
assert(RegExp.rightContext == '-fro');

```

VisiBone also makes
several printed web
color references.

[Posters & Charts](#)



[Laminated Cards](#)



that match the
"VisiBone2"
swatch collection
in Adobe
Illustrator and
Photoshop.

Plus two varieties
of [Mouse Pads](#).

And a chart with
1068 non-web-safe
colors:

```
assert(RegExp.input == 'to-and-fro');
assert(RegExp.index == 3);
```

```
s = RegExp.$1;
s = RegExp.$2;
:
s = RegExp.lastParen;
```

*side effects of
subexpressions
of the most recent
regular expression
(that made
a match)*

```
/^(\w+)-(\w+)-(\w+)$/ .test('to-and-fro');
assertEquals(RegExp.$1, 'to');
assertEquals(RegExp.$2, 'and');
assertEquals(RegExp.$3, 'fro');
assertEquals(RegExp.lastParen, RegExp.$3);
```

```
// Most punctuation symbols have some special meaning in
// regular expressions. To search for those characters in
// your target, you should precede them with a slash.
// Here are those characters all slashified:
assert(/\!\$\%\(\)\*\+\.\,\:\;\=\?\[\]\^\{\}\|\/
.test("!$()*+./:;=<?[]^{| }/");
// The following characters don't (yet) need to be slashified:
assert(/#%&,-;<>@_~/
.test("#%&,-;<>@_~"));
// But they work fine if they are:
assert(/#\%\&\,\-\;\<\>\@\_\~/
.test("#%&,-;<>@_~"));
// Similarly, quotes don't need to be slashified but can be:
assert(/\".test('')); assert(/\".test(''));
assert(/'/.test('')); assert(/'/.test('');
```

```
// Six ways to specify a newline in a regular expression:
assertEquals(3, "abc\n".search(/\n/));
assertEquals(3, "abc\n".search(/\cJ/));
assertEquals(3, "abc\n".search(/\x0A/));
assertEquals(3, "abc\n".search(/\u000A/));
assertEquals(3, "abc\n".search(/\12/));
assertEquals(3, "abc\n".search(/\012/));
```



Feedback welcome!

Thank you, and good luck building!

◆ Bob Stein, VisiBone, stein@visibone.com

Your Feedback Welcome!

Would love your help making the most useful references for web designers in the world.

Feedback, thoughts, questions

Your **name**

Your **email** address (optional, always private, for our correspondence only)

☒ interested in VisiBone **announcements** (only about 3 messages a **year**, see [archive](#))

Submit

[COLOR/FONT](#) [HTML/CSS](#) [JAVASCRIPT](#) [MAP](#) [SQL](#) [PHP](#) [prices](#) [store](#) [shipping](#) [SHOPPING CART](#)
◆◆ [PRODUCTS](#) ◆◆ [about](#) [contact](#) [articles](#) [tnar](#) [blog](#) [HOME](#)

[Color Lab](#) [Swatches](#) [Color Codes](#) [Color Wheel](#) [Color Blindness](#) [CMYK](#) [Popups](#) [Font Survey](#)
[HTML Tags](#) [Style Sheet](#) [Characters](#) [JavaScript](#) [Regular Expressions](#) [Country Codes](#) [Typing Tutor](#)

◆◆◆ FREE ONLINE SERVICES ◆◆◆

◆ Wednesday, 16-May-2018 17:29:45 EDT VisiBone