Pattern	Description	Example	
Character classes			
.	This matches any character, except newline or another unicode line terminator, such as (\n, \r, \u2028 or \u2029).	/f.o/ matches "fao", "feo", and "foo"	
\w	This matches any alphanumeric character, including the underscore. It is equivalent to [a-zA-Z0-9_].	/\w/ matches "f" in "foo"	
\W	This matches any single nonword character. It is equivalent to [^a-zA-Z0-9_].	/\W/ matches "%"in "100%"	
\d	This matches any single digit. It is equivalent to [0-9].	/\d/ matches "1" in "100"	
\D	This matches any non digit. It is equivalent to [^0-9].	/\D/ matches "R" in "R2-D2"	
\s	This matches any single space character. It is equivalent to [\t\r\n\v\f].	/\s/ matches " " in "foo bar"	
\S	This matches any single nonspace character. It is equivalent to [^ \t\r\n\v\f].	/\S/ matches "foo" in "foo bar"	
Literals			
Alphanumeric	These match themselves literally.	/javascript book/ matches "javascript book" in "javascript book"	
\0	This matches a NUL character.		
\n	This matches a newline character.		
\f	This matches a form feed character.		
\r	This matches a carriage return character.		
\t	This matches a tab character.		
\v	This matches a vertical tab character.		
[\b]	This matches a backspace character.		
\xxx	This matches the ASCII character, expressed by the xxx octal number.	/112/ matches the "J" character	
\xdd	This matches the ASCII character, expressed by the dd hex number.	/x4A/ matches the "J" character	
\uxxxx	This matches the ASCII character, expressed by the xxxx UNICODE.	/u0237/ matches the "J" character	
\	This indicates whether the next character is special and is not to be interpreted literally.	/\^/ matches "^" in "char ^"	

Character sets			
[xyz]	This matches any one character enclosed in the character set. You can use a hyphen to denote the range. For example, /[a-z]/ matches any letter in the alphabet and matches /[0-9]/ to any single digit.	/[ao]/ matches "a" in "bar"	
[^xyz]	This matches any one character, which is not enclosed in the character set.	/[^ao]/ matches "b" in "bar"	
Boundaries			
^	This matches the beginning of an input. If the multiline flag is set to true, it also matches immediately after the (\n) line break character.	/^ The/ matches "The" in "The stars", but not "In The stars".	
$	This matches the end of an input. If the multiline flag is set to true, it also matches immediately before the (\n) line break character.	/and$/ matches "and" in "land", but not "and the bar".	
\b	This matches any word boundary (test characters must exist at the beginning or at the end of a word within the string).	/va\b/ matches "va" in "this is a java script book", but not "this is a javascript book".	
\B	This matches any non-word boundary.	/va\B/ matches "va" in "this is a JavaScript book", but not "this is a JavaScript book".	
Grouping, alternation, and back reference			
(x)	This groups characters together to create a clause, that is, it matches x and remembers the match. These are called capturing parentheses.	/(foo)/ matches and remembers "foo" in "foo bar".	
()	Parenthesis also serves to capture the desired subpattern within a pattern.	/(\d\d)\/(\d\d)\/(\d\d\d\d)/ matches "12", "12", and "2000" in "12/12/2000".	
(?:x)	This matches x but does not capture it. In other words, no numbered references are created for the items within the parenthesis. These are called non-capturing parentheses.	/(?:foo)/ matches, but does not remember "foo" in "foo bar".	
|	Alternation combines clauses into one regular expression, and then matches any of the individual clauses. x|y matches either x or y. It is similar to the "OR" statement.	/morning|night/ matches "morning" in "good morning" and matches "night" in "good night".	

()\n	"\n" (where n is a number from 1-9) when added to the end of a regular expression pattern, allows you to back reference a subpattern within the pattern, so, the value of the subpattern is remembered and used as part of the matching.	/(no)\1/ matches "nono" in "nono". "\1" is replaced with the value of the first subpattern within the pattern, or (no), to form the final pattern.	
Quantifiers			
{n}	This matches exactly n occurrences of a regular expression.	/\d{5}/ matches "12345" (five digits) in "1234567890".	
{n,}	This matches n or more occurrences of a regular expression.	/\d{5,}/ matches "1234567890" (minimum of five digits) in "1234567890".	
{n,m}	This matches n to m number of occurrences of a regular expression.	/\d{5,7}/ matches "1234567" (minimum of five digits and a maximum of seven digits) in "1234567890".	
*	This matches zero or more occurrences and is equivalent to {0,}.	/fo*/ matches "foo" in "foo" and matches "foooooooo" in "foooooooooled".	
+	This matches one or more occurrences and is equivalent to {1,}.	/o+/ matches "oo" in "foo".	
?	This matches zero or one occurrences and is equivalent to {0,1}.	/fo?/ matches "fo" in "foo" and matches "f" in "fairy".	
+? *?	"?" can also be used following one of the *, +, ?, or {} quantifiers to make the later match nongreedy, or the minimum number of times versus the default maximum.	/\d{2,4}?/ matches "12" in the "12345" string, instead of "1234" due to "?" at the end of the quantifier nongreedy.	
x(?=y)	Positive lookahead: It matches x only if it's followed by y. Note that y is not included as part of the match, acting only as a required condition.	/Java(?=Script|Hut)/ matches "Java" in "JavaScript" or "JavaHut", but not "JavaLand".	
x(?!y)	Negative lookahead: It matches x only if it's not followed by y. Note that y is not included as part of the match, acting only as a required condition.	/^\d+(?! years)/ matches "5" in "5 days" or "5 books", but not "5 years".	
JavaScript regular expressions methods			
String.match(reg exp)	This executes a search for a match within a string, based on a regular expression.	var myString = "today is 12-12-2000"; var matches = myString.match(/\d{4}/); //returns array ["2000"]	
RegExp.exec(string)	This executes a search for a match in its string parameter. Unlike String.match, the parameter entered should be a string, not a regular expression pattern.	var pattern = /\d{4}/; pattern.exec("today is 12-12-2000"); //returns array ["2000"]	

String.replace(reg exp, replacement text)	This searches and replaces the regular expression portion (match) with the replaced text instead.	var phn = "(201) 123-4567";
		var phnFrmttd = phone.replace(/[\(\)-\s]/g, "");
		//returns 2011234567
		(removed ( ) - and blank space)

String.split (reg exp)	This breaks up a string into an array of substrings, based on a regular expression or fixed string.	var oldstring = "1,2, 3, 4, 5";
		var newstring = oldstring.split(/\s*,\s*/);
		//returns the array ["1","2","3","4","5"]

String.search(reg exp)	This tests for a match in a string. It returns the index of the match, or -1, if it's not found.	var myString = "today is 12-12-2000";
		myString.search(/\d{4}/);
		//returns 15 - index of 2000

RegExp.test(string)	This tests whether the given string matches the Regexp, and returns true if it's matching, and false, if not.	var pattern = /\d{4}/;
		pattern.test("today is 12-12-2000");
		//returns true

.	Any character except newline
a	The character a
ab	The string ab
a|b	a or b
a*	0 or more a's
\	Escapes a special character
0 or more	
+	1 or more
?	0 or 1
{2}	Exactly 2
{2, 5}	Between 2 and 5
{2,}	2 or more
(...)	Capturing group
(?:...)	Non-capturing group
\Y	Match the Y'th captured group
[ab-d]	One character of: a, b, c, d

[^ab-d]	One character except: a, b, c, d
[\b]	Backspace character
\d	One digit
\D	One non-digit
\s	One whitespace
\S	One non-whitespace
\w	One word character
\W	One non-word character
^	Start of string
$	End of string
\b	Word boundary
\B	Non-word boundary
(?=...)	Positive lookahead
(?!...)	Negative lookahead
g	Global Match
i	Ignore case
m	^ and $ match start and end of line
\n	Newline
\r	Carriage return
\t	Tab
\0	Null character
\YYY	Octal character YYY
\xYY	Hexadecimal character YY
\uYYYY	Hexadecimal character YYYY
\cY	Control character Y
$$	Inserts $
$&	Insert entire match
$`	Insert preceding string
$'	Insert following string
$Y	Insert Y'th captured group

	Credit card numbers	\d{4}[-,]?\d{4}[-,]?\d{4}[-,]\d{4}	
	Email addresses	[a-zA-Z0-9.+-_]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,63}	
	img tags which don't contain an **alt** attribute and which are contained within an opening and closing tag of the same type, on a single line	<([a-zA-Z][a-zA-Z0-9]?).*>.*<[iI][mM][gG] (?![^>]*alt=).*>.*</\1>	