Carácter	Significado
****	Buscará coincidencias conforme a las siguientes reglas: Una barra invertida precediendo un caracter simple indica que éste debe ser interpretado como un caracter especial y no de forma literal. Por ejemplo, una 'b' sin barra invertida precediéndole coincidirá con cualquier 'b' minúscula en la cadena, sin embargo, una vez que le precede, no coincidirá con algún caracter en específico, representará el delimitador especial de palabras. Una barra invertida que precede a un caracter especial indica que éste deberá ser interpretado literalmente, esto es, como un caracter simple y no como un caracter especial. Por ejemplo, en el patrón '/a*/' el caracter '*' indica que se deberá buscar toda secuencia con cero o más 'a', por el contrario, el cambiar el patrón a '/a*/', el caracter especial es interpretado como un caracter simple, y cadenas como 'a*' harán coincidencia. No se olvide que al usar la barra invertida en el constructor RegExp ésta escapará cualquier caracter que preceda, dado que '\' es también el caracter de escape para cadenas de texto.
^	Coincide con el principio de la entrada. Si la bandera de miltilínea está activada, también coincidirá inmediatamente después de un salto de línea. Por ejemplo, /^A/ no coincide con la 'A' en "an A", pero sí con la 'A' en "An E". El caracter '^' tiene un significado diferente cuando aparece como el primer caracter en un patrón. Véase patrones complementarios para mayores detalles y ejemplos.
$	Busca el final de la entrada. Si la bandera de multilínea se establece en true, también buscará inmediatamente antes de un caracter de salto de línea. Por ejemplo, la expresión /r$/ no encontrará el caracter 'r' en la cadena "cenaremos", pero sí la encontrará en la cadena "cenar".
*****	Busca el caracter precedente 0 (cero) o más veces. Es equivalente a {0,}. Por ejemplo, la expresión /bo*/ encontrará la subcadena 'boooo' en la cadena "A ghost booooed" y el caracter 'b' en la cadena "A bird warbled", pero no encontrará nada en la cadena "A goat grunted".
+	Busca el caracter precedente 1 o más veces. Es equivalente a {1,}. Por ejemplo, la expresión /u+/ encontrará el caracter 'u' en la cadena "dulce" y todos los caracteres 'u' en la cadena "duuuuulce".
?	Busca el caracter precedente 0 (cero) o 1 (una) vez. Es equivalente a {0,1}. Por ejemplo, la expresión /e?le?/ encontrará la subcadena 'el' en la cadena "angel" y la subcadena 'le' en la cadena "angle" y también el caracter 'l' en la cadena "oslo". Si se utiliza inmediatamente después que cualquiera de los cuantificadores *, +, ?, o {}, hace que el cuantificador no sea expansivo (encontrando la menor cantidad posible de caracteres), en comparación con el valor predeterminado, que sí es expansivo (encontrando tantos caracteres como le sea posible). Por ejemplo, aplicando la expresión /\d+/ a la cadena "123abc" encuentra "123". Pero aplicando la expresión /\d+?/ a la misma cadena, encuentra solamente el caracter "1". Also used in lookahead assertions, as described in the x(?=y) and x(?!y) entries of this table.
.	(El punto decimal) coincide con cualquier caracter excepto un caracter de nueva linea. Por ejemplo , /.n/ coincide 'an' y 'on' en "nay, an apple is on the tree", pero no 'nay'.
(x)	Busca 'x' y recuerda la busqueda, como el siguiente ejemplo lo muestra. Los parentesis son llamados *capturando parentesis.* El '(foo)' y '(bar)' en el patron /(foo) (bar) \1 \2/ busca y recuerda las primeras dos palabras en el string "foo bar foo bar". El \1 y \2 en el patron coincida las dos últimas palabras de la cadena. Note that \1, \2, \n are used in the matching part of the regex. In the replacement part of a regex the syntax $1, $2, $n must be used, e.g.: 'bar foo'.replace(/(...) (...)/, '$2 $1').
(?:x)	Matches 'x' but does not remember the match. The parentheses are called *non-capturing parentheses*, and let you define subexpressions for regular expression operators to work with. Consider the sample expression /(?:foo){1,2}/. Without the non-capturing parentheses, the {1,2} characters would apply only to the last 'o' in 'foo'. With the non-capturing parentheses, the {1,2} applies to the entire word 'foo'.

x(?=y)	Matches 'x' only if 'x' is followed by 'y'. This is called a lookahead. 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.
x(?!y)	Matches 'x' only if 'x' is not followed by 'y'. This is called a negated lookahead. For example, /\d+(?!\.)/ matches a number only if it is not followed by a decimal point. The regular expression /\d+(?!\.)/.exec("3.141") matches '141' but not '3.141'.
x|y	Matches either 'x' or 'y'. For example, /green|red/ matches 'green' in "green apple" and 'red' in "red apple."
{n}	Matches exactly n occurrences of the preceding character. N must be a positive integer. For example, /a{2}/ doesn't match the 'a' in "candy," but it does match all of the a's in "caandy," and the first two a's in "caaandy."
{n,m}	Where n and m are positive integers. Matches at least n and at most m occurrences of the preceding character. When m is zero, it can be omitted. For example, /a{1,3}/ matches nothing in "cndy", the 'a' in "candy," the first two a's in "caandy," and the first three a's in "caaaaaaandy" Notice that when matching "caaaaaaandy", the match is "aaa", even though the original string had more a's in it.
[xyz]	Character set. This pattern type matches any one of the characters in the brackets, including escape sequences. Special characters like the dot(.) and asterisk (*) are not special inside a character set, so they don't need to be escaped. You can specify a range of characters by using a hyphen, as the following examples illustrate. The pattern [a-d], which performs the same match as [abcd], matches the 'b' in "brisket" and the 'c' in "city". The patterns /[a-z.]+/ and /[\w.]+/ match the entire string "test.i.ng".
[^xyz]	A negated or complemented character set. That is, it matches anything that is not enclosed in the brackets. You can specify a range of characters by using a hyphen. Everything that works in the normal character set also works here. For example, [^abc] is the same as [^a-c]. They initially match 'r' in "brisket" and 'h' in "chop."
[\b]	Matches a backspace (U+0008). You need to use square brackets if you want to match a literal backspace character. (Not to be confused with \b.)
\b	Matches a word boundary. A word boundary matches the position where a word character is not followed or preceeded by another word-character. Note that a matched word boundary is not included in the match. In other words, the length of a matched word boundary is zero. (Not to be confused with [\b].) 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.
\B	Matches a non-word boundary. This matches a position where the previous and next character are of the same type: Either both must be words, or both must be non-words. The beginning and end of a string are considered non-words. For example, /\B../ matches 'oo' in "noonday", and /y\B./ matches 'ye' in "possibly yesterday."
\d	Matches a digit character. Equivalent to [0-9]. For example, /\d/ or /[0-9]/ matches '2' in "B2 is the suite number."
\D	Matches any non-digit character. Equivalent to [^0-9]. For example, /\D/ or /[^0-9]/ matches 'B' in "B2 is the suite number."
\f	Matches a form feed (U+000C).
\n	Matches a line feed (U+000A).
\r	Matches a carriage return (U+000D).
\s	Matches a single white space character, including space, tab, form feed, line feed. Equivalent to [\f\n\r\t\v \u00a0\u1680\u180e\u2000\u2001\u2002\u2003\u2004 \u2005\u2006\u2007\u2008\u2009\u200a \u2028\u2029\u202f\u205f \u3000]. 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\u180e\u2000 \u2001\u2002\u2003\u2004 \u2005\u2006\u2007\u2008\u2009\u200a\u2028\u2029\u202f\u205f\u3000]. For example, /\S\w*/ matches 'foo' in "foo bar."
\t	Matches a tab (U+0009).
\v	Matches a vertical tab (U+000B).
\w	Matches any alphanumeric character 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 non-word character. Equivalent to [^A-Za-z0-9_]. For example, /\W/ or /[^A-Za-z0-9_]/ matches '%' in "50%."
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."