

## EXPRESIONES REGULARES

Las expresiones regulares son unos patrones, constituidos por caracteres, dígitos numéricos, signos de puntuación y/o comodines, que se emplean, entre otras cosas, para determinar si una cadena se ciñe a un determinado formato. Así pues, podemos, por ejemplo, comprobar si una cadena empieza por la letra A mayúscula, a continuación tiene seis dígitos numéricos, luego un guion y dos letras comprendidas entre la c y la s minúsculas. Por supuesto, lo más probable es que usted no necesite nunca verificar si una cadena se ciñe a un patrón tan estrañalario como el que acabo de describir, pero es sólo un ejemplo. Sin embargo, seguro que alguna vez necesitará comprobar si una cadena coincide con un código postal. Los códigos postales en España y en otros países están formados por cinco dígitos numéricos, de modo que necesitaremos un patrón que determine que nuestra cadena contiene cinco dígitos numéricos. Esto lo podremos usar también para, por ejemplo, validar de forma compacta, cómoda y profesional direcciones de correo electrónico.

Básicamente se trata de crear un patrón y luego comparar una cadena con él, para determinar si coincide o no, así que, lo primero que tenemos que hacer es aprender a crear estos patrones según nuestras necesidades. Una expresión regular se crea como una secuencia de caracteres o comodines entre dos barras inclinadas (slashes). Por ejemplo, una expresión regular válida sería /A\d\d\d\d\d\d-[cs]/ (por cierto, esta expresión regular sería la que he mencionado en el párrafo anterior).

Como he dicho anteriormente, en una expresión regular puedo incluir caracteres, dígitos, signos de puntuación y comodines. A continuación, aparece una tabla con los comodines que se emplean en JavaScript. Esta tabla es un compendio

de dichos comodines. Échale un breve vistazo y, a continuación, paso a explicarle cómo se construyen y cómo funcionan las expresiones regulares que, en definitiva, son objetos (*RegExp*) y tienen sus propiedades y sus métodos, que aprenderemos en este Apéndice.

LISTADO DE COMODINES PARA CREAR LAS EXPRESIONES REGULARES	
Comodín	Coincidencias
\d	Un dígito del 0 al 9.
\D	Cualquier carácter que no sea un dígito del 0 al 9.
\w	Una letra, un dígito o un guión bajo (_).
\W	Cualquier carácter que no sea una letra, un dígito o un guión bajo.
.	Cualquier carácter excepto el de nueva línea (\n).
\s	Un espacio en blanco, una tabulación (\t), una nueva línea (\n) o un retorno de carro (\r).
\S	Cualquier carácter que no sea un espacio en blanco, ni una tabulación, ni una nueva línea ni un retorno de carro.
[]	Cualquier carácter incluido dentro de los corchetes.
[x1-x2]	Cualquier carácter incluido en el rango de letras o números entre x1 y x2.
[^]	Cualquier carácter no incluido en los corchetes.
[^x1-x2]	Cualquier carácter no comprendido en el rango entre x1 y x2.
\b	Coincide cuando un carácter (o conjunto de caracteres) se encuentra al principio o al final de una palabra.
\B	Coincide cuando un carácter o conjunto de caracteres NO se encuentran al principio o al final de una palabra.
?	Si el carácter que le precede aparece una vez, o ninguna, en la cadena.
*	Si el carácter precedente no aparece o si lo hace, al menos, una vez, pero no hay otro en su lugar.
+	Si el carácter precedente aparece una o más veces.
{n,}	Si el carácter precedente aparece n o más veces.
{n}	Si el carácter precedente aparece n veces.

LISTADO DE COMODINES PARA CREAR LAS EXPRESIONES REGULARES (Cont.)	
Comodín	Coincidencias
{n,m}	Si el carácter precedente aparece, al menos, n veces y, como mucho, m veces.
^	Si los caracteres siguientes aparecen al comienzo de la cadena.
\$	Si los caracteres siguientes aparecen al final de la cadena.
	Se usa para indicar coincidencia con cualquiera de dos patrones diferentes.

La tabla anterior se puede usar como resumen-recordatorio de los distintos comodines de las expresiones regulares. Vamos a explicar, mediante unos ejemplos, cómo debemos construir estas expresiones. Después, entraremos en detalles sobre cómo crear y usar una expresión regular.

## H.1 COMPORTAMIENTO DE LOS COMODINES

Lo primero que vamos a aclarar es que, como hemos dicho al principio de este Apéndice, en una expresión regular no sólo se incluyen comodines, sino que se pueden incluir caracteres que deberán aparecer en la cadena que sometamos a la expresión regular para que se produzca una coincidencia. Por ejemplo, supongamos la siguiente expresión:

/A2-4/

Si ahora sometemos una cadena cualquiera a esta expresión, JavaScript comprobará si, en cualquier parte de la cadena, se encuentra la secuencia de caracteres de la expresión, es decir, una A mayúscula, el dígito 2, un guión y el dígito 4. Por ejemplo, las siguientes cadenas sí coinciden con la expresión:

"A2-4"  
"xsxA2-4"

"A2-4 sdf3"  
"AsA2-4d6r"

Sin embargo, las cadenas que aparecen a continuación no coinciden (recuerde que JavaScript distingue entre mayúsculas y minúsculas):

"a2-4"  
"A4-4"

"A2--4"  
"X2-4"

Por supuesto, si el uso de expresiones regulares se limitara a buscar una secuencia de caracteres fija dentro de una cadena, no haría falta complicarse tanto la vida. El método `indexOf()` del objeto `String` cumple perfectamente este objetivo (vea el Capítulo 6 si tiene alguna duda al respecto). Y aquí es donde entran en juego los comodines, que representan la verdadera potencia y flexibilidad de las expresiones regulares.

### H.1.1 El comodín \d

Este comodín representa la presencia en la cadena de un dígito numérico (cualquiera del 0 al 9). Por ejemplo, supongamos la siguiente expresión regular:

```
/A\d-\d/
```

La expresión buscará que la cadena tenga una secuencia que incluya la letra A mayúscula, un dígito cualquiera del 0 al 9, un guión y otro dígito cualquiera del 0 al 9. Esto ya nos da cierta flexibilidad, ya que no determinamos cuáles deberán ser los dígitos. Estas cadenas coinciden con la expresión regular planteada:

"A2 - 4"	"alda6 - 7"
"A5 - 9dñf4"	"bd1A0 - 5eee4"

En cambio, las siguientes cadenas no coinciden con la expresión:

"a24"	"A2 - - 4"
"alfa4 - 4"	"a2 - 4aa3"

```
*****
```

### H.1.2 El comodín \D

Este comodín busca caracteres que no sean dígitos numéricos. Vamos a crear una expresión regular de ejemplo:

```
/\D\D9\D/
```

Esta expresión buscará una secuencia compuesta por dos caracteres que no sean dígitos, el dígito 9 y otro carácter más que no sea un dígito. Las siguientes cadenas coinciden con la expresión:

"a.9v"	"cadena con un 9 en medio"
"CADENA CON UN 9 EN MEDIO"	"453AS9_03"

Sin embargo, las cadenas que aparecen a continuación no coinciden:

"A99a"	"a.67"
"asd6 .94"	"cadena con un 9"

```
*****
```

### H.1.3 El comodín \w

Este comodín busca en la cadena un carácter que sea una letra, un dígito o un guión bajo (\_).

Vamos a considerar una expresión regular como la que aparece a continuación:

```
/\d\d\w\w\w/
```

Esta expresión buscará en la cadena una secuencia formada por dos dígitos (\d\d), dos letras, dígitos o guiones bajos (\w\w) y algo que no sea un dígito (w).

Como ve, en una expresión regular se pueden combinar diferentes comodines, para crearla de acuerdo a nuestras necesidades. Las siguientes cadenas coinciden con la expresión planteada (dado que estos ejemplos son un poco más complejos de seguir, he resaltado la parte de las cadenas que coincide con la expresión regular planteada):

"567as"	"record 45_A*"
"expresión 99_A"	"Cadena con un 34_en medio"

En cambio, las siguientes cadenas no coinciden con la expresión:

"Alfa"	"Esto es una cadena"
"Cadena con un 34 en madio"	"99xx8"

```
*****
```

### H.1.4 El comodín \W

Este comodín es contrario al anterior. Causa la coincidencia cuando el carácter que encuentra no es una letra, ni un dígito ni un guión bajo. Supongamos la siguiente expresión:

```
/\W\W1\W/
```

Esta expresión buscará una secuencia formada por, al menos, dos caracteres que no sean letras, dígitos o guiones bajos, el dígito 1 y otro carácter que no sea una letra, ni un dígito ni un guión bajo. Las siguientes cadenas coinciden con la expresión planteada:

"<-1<"  
"clave: \*\*1\*\*!"

"++1+"  
"alkdf++1+432 a"

En cambio, las siguientes cadenas no coinciden:

"número 3"  
"Manuel"  
\*\*\*\*\*

"cadena -1a"  
"Tlf (91) 000.00.00"  
\*\*\*\*\*

### H.1.5 El comodín . (punto)

Cuando se encuentra un punto en una expresión, coincide con cualquier carácter que no sea nueva línea (\n).

Suponga una expresión regular como la siguiente:

/...../

Esta expresión coincidirá cuando encuentre una secuencia de, al menos, cinco caracteres que no sean salto de línea. Las siguientes cadenas coinciden:

"alfabeto"  
"Tlf (91) 000.00.00"

"Teruel"  
"Extraño"

En cambio, las siguientes no coinciden:

"1212"  
"debo"  
\*\*\*\*\*

"raro"  
"!!!"

### H.1.6 El comodín \s

Este comodín provoca la coincidencia con cualquier carácter de espaciado. Puede ser un espacio en blanco, una tabulación (\t), un salto de línea (\n) o un retorno de carro (\r). Supongamos una expresión como la siguiente:

/\d\d\w\s\d/

Esta expresión coincidirá cuando encuentre dos dígitos seguidos, un carácter que puede ser un dígito, letra o guión bajo, un carácter de espaciado y otro dígito más. Las siguientes cadenas causarán coincidencia:

"alfa233 7"  
"dos 22a 3"

"clave: 45A 3"  
"929 3"

En cambio, las siguientes cadenas no coinciden:

"123454"  
"alfa"  
\*\*\*\*\*

"edre"  
"domingo"  
\*\*\*\*\*

### H.1.7 El comodín \s

Este comodín busca un carácter que no sea de espaciado, es decir, ni espacio en blanco, ni tabulación, ni nueva línea ni retorno de carro. Veamos un ejemplo:

/\s\s\s\s\s\s/

Esta expresión busca una secuencia de, al menos, seis caracteres que no sean de espaciado. Esto no quiere decir que, en toda la cadena, no deba existir ningún carácter de espaciado para causar una coincidencia, sino que debe haber, al menos, seis caracteres seguido que no sean de espaciado. Parece una diferencia sutil, pero es muy importante. Las siguientes cadenas causarán coincidencia:

"123456"  
"desterrado"

"alfabeto"  
"123456 er"

En cambio, las siguientes no causarán coincidencia:

"12345 6er"  
"algo nuevo"  
\*\*\*\*\*

"buen dia"  
"alfa 2"  
\*\*\*\*\*

### H.1.8 El comodín [] (rango)

Cuando se emplean corchetes en una expresión regular significa que, para causar coincidencia, donde están los corchetes debe haber un carácter que coincida con alguno de los que hay encerrados dentro. Dicho así quizás suene un poco críptico, pero lo aclararemos con unos ejemplos. Suponga la siguiente expresión regular:

`/ [+-]\d\d\d/`

Esta expresión causará coincidencia cuando encuentre una secuencia de cuatro caracteres, de los cuales, el primero deberá ser un signo más (+) o menos (-), es decir, alguno de los que aparecen encerrados entre los corchetes. Los otros tres caracteres deberán ser dígitos del 0 al 9. Las siguientes cadenas causarán la coincidencia:

"+156"	"-861"
"Este valor es -1598"	"El horno está a +2000
grados"	

En cambio, las siguientes cadenas no causarán coincidencia:

"+15"	"-20"
"30 * 10"	"El horno está a 2000
grados"	

Así pues, como acabamos de ver, se busca en la cadena un carácter que esté entre los corchetes. Pero si la lista de posibles caracteres es muy larga y éstos son consecutivos, podemos especificar un rango. Por ejemplo, suponga la expresión:

`/ [a-n]\d\d\d/`

Esta expresión causará la coincidencia cuando encuentre una secuencia formada por cuatro caracteres de los que el primero debe ser una letra comprendida entre la a y la n (**minúsculas**). Los otros tres caracteres deben ser dígitos numéricos. Las siguientes cadenas causarán coincidencia:

"b562"	"Zona g3456"
"alfil23039sjd"	"dalkld232424sdd"

En cambio, las siguientes no coincidirán:

"B562"	"Zona z3456"
"El momento 3452"	"AR35"

Además, para un mismo carácter de una cadena, podemos incluir en la expresión regular más de un rango. Por ejemplo, suponga que quiere buscar la coincidencia cuando se encuentre una letra, entre la a y la z, tanto si es mayúscula como minúscula.

Vamos a crear una expresión regular que busque una secuencia de tres letras seguidas, con indiferencia de que sean mayúsculas o minúsculas, y dos dígitos. La expresión sería la siguiente:

`/ [A-Za-z][A-Za-z]\d\d/`

Esta expresión causaría coincidencia con, por ejemplo, las siguientes cadenas:

"Bx86"	"bx23"
"bx223wq"	"Esto es BX1123"

En cambio, las siguientes cadenas no causarán coincidencia:

"D871"	"R-234"
"fs 3"	"Domingo 31"

El rango encerrado entre los corchetes puede ser de dígitos numéricos, como en la siguiente expresión:

`/ [1-4]Z/`

Esta expresión buscará, en la cadena a examinar, una secuencia formada por un dígito del 1 al 4 y la letra Z mayúscula. Las siguientes cadenas coincidirán:

"2Z"	"A34ZQ"
"1Z"	"23Zwer23"

En cambio, las siguientes no coincidirán:

"5Z"	"Z2"
"3z"	"12345-Z"

Por último, debo comentar que en un conjunto de corchetes se pueden incluir un rango y un valor. Suponga una expresión como la siguiente:

`/ [a-dr]\d\d/`

Esta expresión buscará una secuencia formada por tres caracteres. El primero deberá ser una letra minúscula entre la a y la d o bien la r. Los otros dos son dígitos numéricos. Las siguientes cadenas provocarán la coincidencia:

"b34"	"r263"
"alfa345"	"vivir1203"

En cambio, las siguientes cadenas no coincidirán:

```
"n23"
"alfa3"
*****
"A89"
"Decid 33"
*****
```

### H.1.9 El comodín [^] (fuera de rango)

Este comodín es complementario del anterior. Se emplea para buscar un carácter que **no** esté en el rango especificado. Suponga la siguiente expresión:

```
/[^A-Z]\w/
```

Se buscará en las cadenas examinadas una secuencia de dos caracteres de los cuales el primero no debe ser una letra mayúscula, aunque sí podrá ser una minúscula, un dígito o cualquier otro. El segundo carácter no deberá ser una letra, un dígito ni un guion bajo. Las siguientes cadenas causan coincidencia:

```
"a*"
"9-A"
"4-"
" Lobo gris"
```

Sin embargo, las siguientes no coincidirán:

```
"A9"
"4b"
*****
" a9"
" Lobo"
```

### H.1.10 El comodín \b

Este comodín busca en la cadena una palabra que comience o acabe con la secuencia de caracteres indicados, dependiendo de dónde se sitúe el propio comodín. Por ejemplo, vamos a suponer que tenemos una expresión regular como la que aparece a continuación:

```
/\bado/
```

En esta expresión vemos que se buscarán palabras que comiencen por *ado*.

Las siguientes cadenas provocarán la coincidencia:

```
"Esto es un adorno"
" Lomo adobado"
"Yo adoro JavaScript"
"Estoy adormecido"
```

Sin embargo, las siguientes cadenas no coinciden:

```
"Esto es bonito"
"Alfa 345"
```

```
"Pollo asado"
"Domingo soleado"
```

Si el comodín \b se sitúa detrás de la secuencia de caracteres, la expresión buscará en la cadena una palabra que acabe en dicha secuencia.

Vamos a suponer que tenemos una expresión regular como la que aparece reflejada a continuación:

```
/ido\b/
```

Se buscará en la cadena alguna palabra que finalice con la secuencia *ido*. Las siguientes cadenas provocarán la coincidencia:

```
"Estoy dormido"
"He cosido la ropa"
```

```
"No ha venido Juan"
"Lo he conseguido"
```

Mientras que las siguientes no coincidirán:

```
"Estás idolatrado"
"Ésta no está"
```

```
"Sean bienvenidos"
"Cualquier otra"
```

Por último, permítame puntualizar que este comodín se puede poner al principio y al final de una secuencia de caracteres. Suponga una expresión regular como la siguiente:

```
/\bado\b/
```

Esta expresión buscará en la cadena una palabra que empiece y acabe con la misma secuencia. Por ejemplo, si en la cadena aparece la palabra "adobado", se producirá una coincidencia, pero no así con "adobo" o "idolatrado".

### H.1.11 El comodín \B

Este comodín es complementario del anterior. Busca que no exista en la cadena ninguna palabra que empiece o acabe con la secuencia de caracteres especificada. Supongamos la siguiente expresión:

```
/\Bado/
```

Se buscará que no haya en la cadena ninguna palabra que empiece por *ado*. Las siguientes cadenas provocarán la coincidencia:

"Esto es bonito"  
"Alfa 345"

"Pollo asado"  
"Domingo soleado"

En cambio, las siguientes no coincidirán:

"Esto es un adorno"  
"Lomo adobado"

"Yo adoro JavaScript"  
"Estoy adormecido"

Al igual que el comodín anterior, éste admite también identificar el final de las palabras, o ambos extremos, el principio y el final. Es decir, podemos construir una expresión regular para que se busquen cadenas en las que, por ejemplo, no haya ninguna palabra que acabe con la secuencia de caracteres especificada. Un ejemplo sería la expresión /ado\b/. Por último, hay que indicar que podemos dejar que en la cadena no haya palabras que empiecen ni acaben con la secuencia especificada. Esta expresión podría ser, por ejemplo, /\Bado\b/. Por supuesto, la secuencia no tiene, necesariamente, que estar constituida, como en nuestros ejemplos, por tres letras. Puede estar formada por cualquier conjunto de caracteres, aunque, al ser éste y el anterior comodines que se fijan en las palabras, lo normal es que esos caracteres sean letras. Sin embargo, podría tratarse de números u otros caracteres.

## H.1.12 El comodín ?

Este comodín provoca la coincidencia si el carácter que le precede no aparece en la cadena o si aparece una sola vez. Supongamos la siguiente expresión:

/\d\d\d-\?\d\d/

Esta expresión buscará una secuencia de cinco caracteres que deberán ser dígitos. Además, podrá aparecer o no aparecer un guion. Si aparece, sólo podrá haber uno y deberá estar situado entre el tercer dígito y el cuarto. Como ve, cada vez se va sofisticando más el uso de expresiones regulares: las siguientes cadenas coincidirán con el patrón:

"12345"  
"123423-323-23"

"123-45"  
"al-fa234-65qv"

En cambio, las siguientes no coincidirán:

"23-345"  
"123--45"  
/\*\*\*\*\*

## H.1.13 El comodín \*

Este comodín busca que en la cadena no aparezca el carácter que le precede, o que aparezca cualquier número de veces, pero que no haya ningún otro en su lugar. Por ejemplo, supongamos la siguiente expresión:

/e-\*mail/

Se buscará en la cadena la secuencia *e-mail*. Opcionalmente, entre la *e* y la *m* podrá haber uno o varios guiones, pero ningún otro signo.

Las siguientes cadenas causarán coincidencia:

"Dime tu e-mail"  
"Dime tu e--mail"

"Dime tu email"  
"Dime tu e---mail"

Sin embargo, no coincidirán las siguientes:

"Dime tu e.mail"  
"Dime tu e mail"  
\*\*\*\*\*

"Dime tu e:mail"  
"Dime tu e -mail"  
\*\*\*\*\*

## H.1.14 El comodín +

Este comodín provoca la coincidencia si el carácter que le precede aparece una o más veces, pero no coincide si no aparece dicho carácter en la secuencia. Por ejemplo, supongamos la siguiente expresión:

/\d\d\d\d+\d\d/

Esta expresión busca una secuencia constituida por cinco dígitos. Entre el tercero y el cuarto deberá haber, al menos, un guion, aunque podrá haber varios. Las siguientes cadenas provocarán la coincidencia:

"Esto es 123-34"  
"Aquí hay dos guiones 84398--345"

"675---34"  
"2409-98238dsj"

En cambio, las siguientes no coincidirán:

"123456"  
"123 45"  
\*\*\*\*\*

### H.1.15 El comodín {n}

Este comodín provoca la coincidencia si el carácter que le precede aparece, exactamente, tantas veces seguidas como indica el número entre las llaves. Suponga la siguiente expresión regular:

```
/co{2}r/
```

Esto indica que se buscará una secuencia formada por la letra c, dos letras o y la r (minúsculas, todas ellas). Es, por lo tanto, una forma de simplificación cuando se buscan muchos caracteres iguales seguidos. Sería equivalente a la expresión:

```
/coor/
```

Por supuesto, no tiene mucho interés usar este sistema cuando se trata, como en este ejemplo, de sustituir sólo dos letras.

### H.1.16 El comodín {n,}

Este comodín es una variante un poco más sofisticada del anterior. Provoca la coincidencia si el carácter que le precede se repite tantas veces seguidas como indica el número que aparece entre las llaves o más.

Por ejemplo, suponga la siguiente expresión:

```
/co{2,}r/
```

Esta expresión coincidiría con cadenas como "coordinar" o "cooordinar", pero no con "cordinar".

```
*****
```

### H.1.17 El comodín {n,m}

Este comodín provoca la coincidencia cuando el carácter que le precede aparece un número de veces comprendido entre n y m. Vea la siguiente expresión:

```
/\d{3,6}/
```

Se busca una secuencia constituida por entre tres y seis dígitos. Coincidirán:

"123"  
"Límite: 1300 metros."

"Secuencia 12345"  
"123456789"

En cambio, no coincidirán las siguientes:

"12"  
"Límite: 13 metros."  
\*\*\*\*\*

"12-03-76"  
"1-23+as"  
\*\*\*\*\*

### H.1.18 El comodín ^

Este comodín provoca la coincidencia si el carácter o conjunto de caracteres que le siguen aparecen al principio de la cadena. Por ejemplo, suponga la siguiente expresión regular:

```
/^Hoy/
```

Esta expresión coincidirá con cualquier cadena que empiece por la secuencia Hoy, independientemente de lo que haya después en la misma cadena (incluso otra secuencia Hoy).

### H.1.19 El comodín \$

Este comodín provocará la coincidencia siempre que la cadena termine con el carácter o secuencia de caracteres que le preceden. Por ejemplo, imagine la siguiente expresión regular:

```
/hoy$/
```

Esta expresión coincidirá con cualquier cadena que termine con la secuencia hoy, sea lo que sea que tenga delante, o aunque no tenga nada.

### H.1.20 Coincidencias múltiples ()

Es muy posible que usted necesite una expresión regular en la que se puedan mezclar criterios ambiguos de coincidencia de las cadenas.

Suponga, por ejemplo, que tiene que crear una expresión para validar los números de serie de los artículos de un fabricante concreto. Estos números siguen un patrón tal que están constituidos por dos letras, un guion y seis dígitos. Las letras pueden ser sn o bien ns. En todo caso, siempre será una de estas dos combinaciones. La expresión regular que usted creará debe admitir cualquiera de las combinaciones especificadas, pero no debe admitir ninguna otra letra. Por

ejemplo, unos números de serie válidos serían así: *sn-283932*, o *ns-728393*, pero no podría ser, por ejemplo, *za-324987*.

Para crear la expresión regular, lo del guión y los seis dígitos no tiene mayor problema. Podríamos hacer algo como lo siguiente:

`/-\d{6}/`

Además, podemos mejorarlo, añadiendo el signo \$ para indicarle a JavaScript que la cadena debe terminar ahí:

`/-\d{6}\$/`

El problema son las letras. Podemos poner algo así:

`/^((sn)|(ns))-\d{6}\$/`

Veamos qué es lo que hemos hecho con las letras. En primer lugar hemos agrupado las dos secuencias válidas mediante paréntesis, para indicar que no es que sea válida cualquier combinación de la *s* y la *n*. Por ejemplo, las combinaciones *ss* o *nn* no son válidas. Después hemos usado el signo | para indicarle a JavaScript que puede valernos cualquiera de las combinaciones indicadas, es decir, *sn* o bien *ns*. La expresión regular, por lo tanto, espera encontrar en la cadena una de estas secuencias, luego el signo | indica que es válida cualquiera de las dos secuencias: la que aparece a la izquierda del signo o la que aparece a la derecha, pero alguna de las dos tiene que estar en la cadena para que se produzca la coincidencia. No confunda este signo con el operador lógico || (or) de los condicionales.

Además, hemos encerrado toda esa parte de la expresión regular en otro par de paréntesis, para que, sea cual sea la secuencia que se halle en la cadena, deba cumplirse también el resto de la expresión. También hemos añadido el comodín ^ para indicarle a JavaScript que la cadena debe empezar, forzosamente, con una de las secuencias especificadas. La expresión no espera encontrar los paréntesis en la cadena, si no que “sabe” que tiene la misión de agrupar y “jerarquizar” los distintos elementos. El concepto es similar a la precedencia de operadores que vimos en el Capítulo 2 del libro. No se preocupe demasiado si no entiende ahora completamente la mecánica de este último ejemplo. Lo cierto es que es más complejo de lo que parece. Enseguida aprenderemos a crear y usar las expresiones regulares y entonces lo verá todo claro.

## H.1.21 Caracteres especiales

Anteriormente hemos visto, en algunos ejemplos, que se puede incluir un carácter concreto en una expresión regular, si queremos que las cadenas que coincidan incluyan ese carácter. Por ejemplo, podemos crear una expresión como la siguiente:

`/A\d\d\d/`

Veamos cómo funciona: lo que hace esta expresión es que, para que una cadena coincida, sea necesario que contenga, al menos, una secuencia de cuatro caracteres, de los cuales el primero debe ser una A mayúscula y los otros tres deben ser dígitos.

Así pues, la cadena "ALFA1546" coincidirá, pero la cadena "ALFA 1546" no coincidirá, porque entre la A mayúscula y el primer dígito hay un espacio que no se encuentra en la expresión regular.

Hasta aquí, todo perfecto. Pero la pregunta es la siguiente: ¿qué ocurre si queremos crear una expresión que detecte coincidencia con, por ejemplo, un punto? Es decir, yo quiero una expresión regular que me detecte coincidencia cuando en la cadena se encuentren, por ejemplo, tres dígitos, un punto y otros dos dígitos. Suponga que creo una expresión como la siguiente:

`/\d\d\d.\d\d/`

Esta expresión coincidiría con la cadena "123.45"… y también con, por ejemplo, "123R45". Esto se debe a que el carácter punto es uno de los comodines de expresiones regulares, que detecta, como dijimos antes, coincidencia con cualquier carácter que no sea el salto de línea.

Si lo que yo quiero es que la expresión regular busque, específicamente, coincidencia con el carácter punto, deberá usar una secuencia de escape, que funciona, exactamente, igual que las que vimos en el Capítulo 2 del libro. Es lo que se llama **escapar un carácter** (recuerde esta expresión, porque es muy habitual). La expresión regular correcta quedaría, por lo tanto, así:

`/\d\d\d.\d\d/`

Esta expresión detectaría la coincidencia con "123.45", pero no con "123R45". Siempre que queramos incluir en una expresión regular un carácter que coincida con un comodín, deberemos escaparlo, para que se comporte como tal.

## H.2 CREAR Y USAR LAS EXPRESIONES REGULARES

Una expresión regular es un objeto `RegExp`, del mismo modo que, por ejemplo, una cadena de texto es un objeto `String`. Podemos crear una expresión regular como una variable, o bien mediante el constructor `RegExp()`. El resultado es el mismo. Por ejemplo, si vamos a crear una expresión regular para buscar en una cadena una secuencia de tres dígitos, podemos hacerlo así:

```
var expresion_1 = /\d\d\d/;
```

o bien del siguiente modo:

```
var expresion_1 = new RegExp (/d\d\d/);
```

De cualquiera de los dos modos queda creado un objeto `RegExp`, de nombre `expresion_1`, que contiene una expresión regular.

Ahora nos falta usarla para comprobar cadenas, es decir, verificar si coinciden o no con el patrón establecido en la expresión. Para ello, los objetos `RegExp` cuentan con un método llamado `test()`. Este método recibe como argumento la cadena que queremos comprobar (o la variable que la contiene) y devuelve un valor booleano: `true` si hay coincidencia o `false`, en caso contrario.

Para entender cómo funciona, veamos el código `expresiones_1.htm`. El listado es el siguiente:

```
<html>
  <head>
    <title>
      Página con JavaScript.
    </title>
    <script language="javascript">
      <!--
        //Se crea una expresión regular
        //mediante el correspondiente constructor.
        var expresion_1 = new RegExp (/d\d\d/);

        //Entramos en un bucle que nos
        //pedirá una cadena.
        do
        {
          cadena = prompt ("Cadena","");
          //Comprobamos si la cadena coincide.
          if (expresion_1.test(cadena))
      </pre>

```

```
{
  alert("Coincide");
} else {
  alert ("NO coincide");
}
//Se sale del bucle al pulsar
//cancelar en el prompt.

} while (cadena != null);
</-->
</script>
</head>
<body>
</body>
</html>
```

Este código crea una expresión regular con la plantilla `\d\d\d`, es decir, que buscará en la cadena una secuencia de tres dígitos seguidos (observe la sintaxis de la primera línea resaltada). A continuación se le pide al usuario que teclee una cadena. Después se verifica, mediante el método `test()`, si la cadena coincide o no con la expresión (la segunda línea resaltada). Cuando el usuario pulsa el botón [CANCELAR] se sale del bucle y se termina la ejecución del script.

Este código funciona bastante bien, pero es poco flexible, ya que, si queremos cambiar la expresión regular tenemos que modificar el código. A continuación tenemos otra muestra de código que le permite al usuario teclear una expresión regular y una cadena para verificar con dicha expresión. Se llama `expresiones_2.htm`.

```
<html>
  <head>
    <title>
      Página con JavaScript.
    </title>
    <script language="javascript">
      <!--
        //Entramos en un bucle que nos
        //pedirá una expresión y una cadena.

        do
        {
          expresion_2 = new RegExp
            (prompt("Introduzca una expresión",""));
          cadena = prompt ("Cadena","");
      </pre>

```

```

//Comprobamos si la cadena coincide.
if (expresion_2.test(cadena))
{
    alert("Coincide");
} else {
    alert ("NO coincide");
}

//Se sale del bucle al pulsar
//cancelar en el prompt.

} while (cadena != null);
//-->

</script>
</head>
<body>
</body>
</html>

```

Fíjese en la sintaxis de la instrucción que pide la expresión. Es importante que, al igual que cuando teclea una cadena como respuesta a un prompt no introduce las comillas, cuando teclee una cadena que va a ser una expresión regular no introduzca los slashes delimitadores. Es decir, si va a introducir una cadena que busque una secuencia de tres dígitos, intodúzcala, exactamente, como aparece en la figura H.1. Si no, no le funcionará.

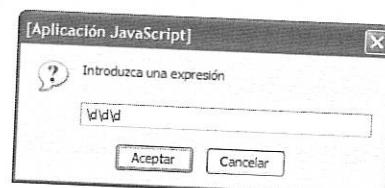


Figura H.1

### H.3 INDICADORES

Como complemento final, a las expresiones regulares se les puede añadir, opcionalmente, un **indicador**. Un indicador actúa modificando ligeramente el comportamiento de la expresión regular. Los dos indicadores que existen son *i* y *g*. A una expresión se le puede añadir un indicador, o los dos, o ninguno.

Los indicadores se incluyen detrás del slash de cierre cuando la expresión se crea como una variable y como segundo argumento si se emplea el constructor.

Por ejemplo, si vamos a crear la expresión regular `/A\d\d\d/` y queremos añadirle los dos indicadores, lo podremos hacer así:

```
var expresion = /A\d\d\dig;
```

o así:

```
var indicadores = "ig";
var expresion = new RegExp (/A\d\d\d/, indicadores);
```

El indicador *i* hace que la expresión ignore las mayúsculas y las minúsculas a la hora de determinar si una cadena coincide o no. Por ejemplo, suponga una expresión regular como la siguiente:

```
/A\d\d\d/i
```

Esta expresión coincidirá con la cadena "A123" y también con "a123".

El indicador *g* se usa para hacer lo que se llama una comprobación global. Su funcionamiento se estudia en el Capítulo 6 del libro, con los métodos `match()` y `replace()` del objeto `String`.

### H.4 COMPROBANDO EXPRESIONES REGULARES

En este apartado incluyo un código para la comprobación de las expresiones regulares, que también se encuentra, naturalmente, en el CD que acompaña al libro. Si usted todavía no conoce las posibilidades que ofrece JavaScript para la gestión de formularios, no podrá entender el funcionamiento hasta que haya leído el texto del correspondiente Capítulo. No obstante, puede usarlo para comprobar cualquier expresión regular. El código se llama `comprobar_expresiones.htm`.

```

<html>
<head>
<title>
    Página con JavaScript
</title>
<script language="JavaScript">
<!--
    function comprobar(formulario)
    {
        var expresion = formulario.patron.value

```

```

        var indicadores =
formulario.banderas.options[formulario.banderas.selectedIndex
].value

        var expresion_1 = new RegExp(expresion,
indicadores)

if(expresion_1.test(formulario.test_string.value))
{
    alert("Hay coincidencia.")
} else {
    alert("NO hay coincidencia.")
}
//-->
</script>
</head>
<body>
<form>
<b>
    Introduzca la expresión y los
indicadores:
</b>
<br>
<input type="text" name="patron">
<select name="banderas">
    <option value="" selected>Ninguno</option>
    <option value="i">i</option>
    <option value="g">g</option>
    <option value="ig">ig</option>
</select>
<br>
<b>
    Introduzca la cadena a comprobar:
</b>
<br>
<input type="text" name="test_string">
<input type="button" value="Comprobar cadena"
onClick="comprobar(this.form)">
</form>
</body>
</html>

```

Ejecútelo y compruebe su funcionamiento con diversas expresiones regulares, incluyendo la que le propuse en el ejemplo del apartado H.1.20.

## Apéndice I

## USO DE COOKIES

En este Apéndice voy a indicarle el método para permitir o denegar a las páginas por las que navegue el uso de cookies. Le explicaré cómo hacerlo en las últimas versiones de los navegadores más extendidos. En versiones anteriores, el proceso es similar, aunque, quizás, deba recurrir a la información del fabricante.

Como ya se menciona en el texto del libro, las cookies sólo sirven para que la página guarde memoria de sus visitas, preferencias, etc. No son códigos maliciosos, en principio. Eso no evita que alguien con conocimientos avanzados de programación pueda acceder a cookies de su ordenador y robarle información. En el Capítulo 11 del libro se explica cómo evitar esto. Sea confiado, pero no ingenuo. Admita las cookies pero, por si acaso, instale en su sistema un buen cortafuegos.

### I.1 EN MICROSOFT INTERNET EXPLORER

Para activar o desactivar las cookies en este navegador, deberá ir al panel de control. Pulse el botón [INICIO] en la barra de tareas y elija la opción [PANEL DE CONTROL]. Una vez dentro del panel de control, busque el ícono con la leyenda [CONEXIONES DE RED E INTERNET]. Haga clic en este enlace y, en el siguiente cuadro de diálogo que se abra, seleccione [OPCIONES DE INTERNET]. Haga doble clic en dicho ícono y se le abrirá un cuadro de diálogo con siete pestañas en la parte superior. Pulse la pestaña [PRIVACIDAD] y verá que el aspecto del cuadro de diálogo es similar al de la figura I.1. Este aspecto puede variar algo, dependiendo de la configuración que usted tenga, a priori, en su navegador.