

```

poo.js:13
▼ {constructor: f} ⓘ
  ► constructor: f Viaje(origen,destino,dias,precio)
  ► [[Prototype]]: Object

poo.js:14
▼ {constructor: f, __defineGetter__: f, __definesetter__: f,
  hasOwnProperty: f, __lookupGetter__: f, ...} ⓘ
  ► constructor: f object()
  
```

Figura 5.12. Cadena de prototipos del objeto viaje1.

Como se observa, existe una cadena de prototipos a la que hacer referencia y que podría modificarse según los intereses.

■ 5.4. Objetos predefinidos del lenguaje

Los objetos predefinidos del lenguaje son aquellos objetos que ya incorpora el lenguaje y que se pueden utilizar libremente con cualquier intérprete de JavaScript. En realidad, no es un concepto que resulte novedoso, puesto que ya se ha trabajado intensamente con muchos de ellos, como **String** (parcialmente), **Array**, **Set** o **Map**.

Esta sección descubre algunos otros objetos predefinidos que resultan de enorme interés por ser de utilidad en una generalidad de casos.

■ ■ ■ 5.4.1. String

En la Unidad 2, al repasar los tipos de datos de JavaScript, se avanzaron algunos usos básicos de las cadenas de caracteres. En aquella ocasión se crearon como primitivas. Ahora, en cambio, se crearán de forma alternativa como objetos usando el constructor **String()**. Tanto las cadenas primitivas como las cadenas desde objeto se pueden usar indistintamente en la mayoría de las situaciones.

Las cadenas se pueden especificar encerrándolas entre comillas simples, ‘cadena’, dobles, "cadena", o invertidas, `cadena`, y su tratamiento será idéntico.

Las cadenas también se pueden comparar, tomando todas las precauciones (como se vio en la unidad anterior), teniendo en cuenta que el criterio por defecto de ordenación de los caracteres es el de Unicode.

En la Tabla 5.1 se recogen algunos de los métodos más frecuentemente utilizados con strings.

Tabla 5.1. Métodos útiles del objeto String

Método	Utilidad
charAt(<i>posición</i>)	Devuelve el carácter que ocupa esa <i>posición</i> .
charCodeAt(<i>posición</i>)	Variante que devuelve el código Unicode.

Método	Utilidad
<code>toUpperCase()</code>	Devuelve la cadena en mayúsculas.
<code>toLowerCase()</code>	Devuelve la cadena en minúsculas.
<code>indexOf(texto)</code>	Devuelve la posición del texto buscado.
<code>lastIndexOf(texto)</code>	Devuelve la posición de la última ocurrencia del texto.
<code>endsWith(textoABuscar)</code>	Devuelve <code>true</code> si el texto finaliza con lo buscado.
<code>startsWith(textoABuscar)</code>	Devuelve <code>true</code> si el texto empieza con lo buscado.
<code>replace(txtAnt,txtNuevo)</code>	Reemplaza el texto buscado por un nuevo texto.
<code>trim()</code>	Elimina los espacios en blanco del inicio y del final.
<code>slice(inicio,fin)</code>	Extrae la cadena desde la posición <i>inicio</i> hasta la posición <i>fin</i> , sin incluir <i>fin</i> .
<code>substr(inicio,n)</code>	Extrae <i>n</i> caracteres desde la posición <i>inicio</i> .
<code>split(delimitador)</code>	Rompe un <i>string</i> usando un carácter <i>delimitador</i> y construye un <i>array</i> con las piezas generadas.

Además, el objeto también dispone de tres métodos estáticos, como se muestra en la Tabla 5.2.

Tabla 5.2. Métodos estáticos del objeto String

Método	Utilidad
<code>String.fromCharCode(num1,...)</code>	Crea una cadena usando una secuencia de valores Unicode.
<code>String.fromCodePoint(num1,...)</code>	Crea una cadena utilizando la secuencia de puntos de código especificada.
<code>String.raw()</code>	Crea una cadena a partir de una plantilla literal sin formato: no escapa caracteres.

```
let cadena = " Bolonia ";
console.log("1." + cadena.charAt(1));
console.log("2." + cadena.toUpperCase());
console.log("3." + cadena.toLowerCase());
console.log("4." + cadena.indexOf("n"));
console.log("5." + cadena.lastIndexOf("o"));
console.log("6." + cadena.replace("B","C"));
console.log("7." + cadena.trim());
console.log("8." + cadena.slice(1,3));
console.log("9." + cadena.substr(1,3));
console.log("10." + cadena.split("o"));
```

1.8
2. BOLONIA
3. bolonia
4.5
5.4
6. Colonia
7.Bolonia
8.Bo
9.Bol
10. B,l,nia
>

Figura 5.13. Resultado de aplicar algunos de los métodos más comunes.



Nota técnica

Los **métodos estáticos** son aquellos que deben ser llamados sin instanciar su clase. Por ejemplo, para usar un método estático del prototipo **String()** no puede crearse un **objeto = new String()** y hacer luego **objeto.metodo()**, sino que hay que invocarlo directamente como **String.metodo()**.

5.4.2. Date

Los objetos **Date** representan un momento fijo en el tiempo que puede representarse en numerosos formatos. Una fecha en JavaScript se especifica como el número en milisegundos que han transcurrido desde el 1 de enero de 1970, UTC.

Importante

Es importante destacar que mientras el valor de la hora en el núcleo del objeto **Date** está en UTC, los métodos básicos para recibir la fecha y la hora o sus derivados trabajan todos en la zona horaria local.



Un objeto **Date** puede crearse sin parámetros, pero también indicando su lista completa (año, mes, día, hora, minutos, segundos, milisegundos) o un número parcial de ellos. Si se especifica uno solo, se interpreta que es el número de milisegundos transcurridos desde el 1 de enero de 1970. Lo mejor es ver algunos ejemplos:

```
let fechaSinParametros = new Date();
let fechaTodosParametros = new Date(2022,8,17,13,59,49,0);
let fechaTresParametros = new Date(2022,8,17);
let fechaUnParametro = new Date(1000);
```

```
Sun Jul 17 2022 19:30:37 GMT+0200 (hora de verano de Europa central)
Sat Sep 17 2022 13:59:49 GMT+0200 (hora de verano de Europa central)
Sat Sep 17 2022 00:00:00 GMT+0200 (hora de verano de Europa central)
Thu Jan 01 1970 01:00:01 GMT+0100 (hora estándar de Europa central)
```

Figura 5.14. Salidas en consola tras variar el número de parámetros del constructor de Date.

El número de parámetros del constructor es un aspecto útil del objeto, pero sin duda la gran cantidad de métodos disponibles permite hacer cualquier cálculo imaginable con fechas. Se repasan algunos de los más utilizados en la Tabla 5.3.

También existen algunas versiones de estos métodos, que se han obviado por claridad, pero que trabajan con UTC o GMT.

Por último, el objeto **Date** también dispone de un conjunto de métodos estáticos (Tabla 5.4).

Tabla 5.3. Métodos más usados del objeto Date

Método	Utilidad
<code>getDate()</code>	Devuelve el día del mes de la fecha (de 1 a 31).
<code>getDay()</code>	Obtiene el día de la semana de la fecha (de 0 a 6).
<code>getFullYear()</code>	Obtiene el año (con cuatro dígitos).
<code>getHours()</code>	Obtiene la hora de la fecha (número de 0 a 23).
<code>getMilliseconds()</code>	Obtiene los milisegundos en la fecha actual.
<code>getMinutes()</code>	Obtiene los minutos de la fecha.
<code>getMonth()</code>	Obtiene el número del mes sabiendo que enero es 0.
<code>getSeconds()</code>	Obtiene los segundos de la fecha.
<code>getTime()</code>	Obtiene el valor en milisegundos de la fecha.
<code> setDate(<i>día</i>)</code>	Modifica el día de la fecha.
<code>setFullYear(<i>año</i>)</code>	Modifica el año de la fecha.
<code>setHours(<i>hora</i>)</code>	Modifica la hora de la fecha.
<code>setMilliseconds(<i>milisegundos</i>)</code>	Modifica los milisegundos de la fecha.
<code>setMinutes(<i>minutos</i>)</code>	Modifica los minutos de la fecha.
<code>setMonth(<i>mes</i>)</code>	Modifica el mes de la fecha.
<code>setSeconds(<i>segundos</i>)</code>	Modifica los segundos de la fecha.
<code> setTime(<i>milisegundos</i>)</code>	Establece la fecha a partir de un valor en milisegundos.
<code>toDateString()</code>	Convierte la fecha a un formato más amigable.
<code>toLocaleString([<i>params</i>])</code>	Muestra la fecha en texto en formato local.
<code>toLocaleDateString()</code>	Muestra la fecha sin la hora en formato local.
<code>toTimeString()</code>	Muestra la hora sin la fecha en formato local.
<code>toString()</code>	Muestra la fecha en texto al estilo JavaScript.
<code> toJSON()</code>	Muestra la fecha en formato JSON.

Tabla 5.4. Métodos estáticos del objeto Date

Método	Utilidad
<code>Date.now()</code>	Devuelve el valor numérico correspondiente al actual número de milisegundos transcurridos desde el 1 de enero de 1970, 00:00:00 UTC, ignorando los segundos intercalares.
<code>Date.parse(<i>objeto</i>)</code>	Transforma la cadena que representa una fecha y retorna el número de milisegundos transcurridos desde el 1 de enero de 1970, 00:00:00 UTC, ignorando los segundos intercalares.
<code>Date.UTC(<i>año, mes, día, horas, minutos, segundos, milisegundos</i>)</code>	Acepta los mismos parámetros de la forma extendida del constructor (por ejemplo, del 2 al 7) y retorna el número de milisegundos transcurridos desde el 1 de enero de 1970, 00:00:00 UTC, ignorando los segundos intercalares.

Actividad resuelta 5.3

Fecha española

Haciendo uso de un objeto **Date**, crea un objeto de una clase que construyas para saludar al usuario tras iniciar sesión y que indique la fecha y la hora actuales.

Solución

```
class saludoHorario{
    usuario = "usuario";
    constructor(usuario) {
        this.usuario = usuario;
    }
    muestraSaludo() {
        let fecha = new Date();
        let dia = fecha.getDate() + "/" + fecha.getMonth() + "/" + fecha.getFullYear();
        let hora = fecha.getHours() + ":" + fecha.getMinutes() + ":" + fecha.getSeconds();
        console.log(`Bienvenido de nuevo ${this.usuario}.`);
        console.log(`Login registrado el ${dia} a las ${hora}.`);
    }
}
const saludar = new saludoHorario("David");
saludar.muestraSaludo();
```

5.4.3. Math

El objeto predefinido **Math** es otro que goza de mucha popularidad por la cantidad de operaciones matemáticas de cierta complejidad que resuelve. No es un objeto de función, no se puede editar y, además, todas sus propiedades y métodos son estáticos. Incluye algunas constantes matemáticas de uso común como las recogidas en la Tabla 5.5.

Tabla 5.5. Constantes matemáticas definidas en el objeto Math

Constante	Utilidad
Math.E	Constante de Euler, aproximadamente 2,718.
Math.LN10	Logaritmo natural de 10, aproximadamente 2,303.
Math.LN2	Logaritmo natural de 2, aproximadamente 0,693
Math.LOG10E	Logaritmo de E con base 10, aproximadamente 0,434
Math.LOG2E	Logaritmo de E con base 2, aproximadamente 1,443.
Math.PI	Ratio de la circunferencia con respecto a su diámetro, aproximadamente 3,14159.
Math.SQRT1_2	Raíz cuadrada de ½, aproximadamente 0,707.
Math.SQRT_2	Raíz cuadrada de 2, aproximadamente 1,414.



Recuerda

Mucha de la funcionalidad matemática que se aporta en esta sección tiene una precisión que depende de la implementación. Diferentes navegadores pueden dar un resultado distinto, e incluso el mismo motor de JavaScript en un sistema operativo o arquitectura diferente puede dar resultados distintos.

En la Tabla 5.6 se recogen algunos de los métodos más usados por la comunidad de programadores.

Tabla 5.6. Métodos más usados del objeto Math

Método	Utilidad
<code>Math.abs(<i>n</i>)</code>	Valor absoluto de <i>n</i> .
<code>Math.acos(<i>n</i>)</code>	Arcocoseno de <i>n</i> .
<code>Math.asin(<i>n</i>)</code>	Arcoseno de <i>n</i> .
<code>Math.atan(<i>n</i>)</code>	Arcotangente de <i>n</i> .
<code>Math.ceil(<i>n</i>)</code>	Redondea <i>n</i> (decimal) a su entero superior.
<code>Math.cos(<i>n</i>)</code>	Coseno de <i>n</i> .
<code>Math.exp(<i>n</i>)</code>	e^n .
<code>Math.floor(<i>n</i>)</code>	Redondea <i>n</i> (decimal) a su entero inferior.
<code>Math.log(<i>n</i>)</code>	Logaritmo decimal de <i>n</i> .
<code>Math.max(<i>a,b</i>)</code>	Devuelve el mayor de dos números, <i>a</i> y <i>b</i> .
<code>Math.min(<i>a,b</i>)</code>	Devuelve el menor de dos números, <i>a</i> y <i>b</i> .
<code>Math.pow(<i>a,b</i>)</code>	a^b .
<code>Math.random()</code>	Devuelve un número aleatorio entre 0 y 1.
<code>Math.round(<i>n</i>)</code>	Redondea <i>n</i> a su entero más próximo.
<code>Math.sin(<i>n</i>)</code>	Seno de <i>n</i> .
<code>Math.sqrt(<i>n</i>)</code>	Raíz cuadrada de <i>n</i> .
<code>Math.tan(<i>n</i>)</code>	Tangente de <i>n</i> .
<code>Math.trunc(<i>n</i>)</code>	Devuelve la parte entera de <i>n</i> , eliminando los decimales.

Importante

Se ha de tener en cuenta que las funciones trigonométricas devuelven los ángulos en radianes. La conversión de radianes a grados se puede hacer con `Math.PI / 180`. Y a la inversa, de grados a radianes: `Math.PI * 180`.



Actividad propuesta 5.4

Trabajando con métodos matemáticos

Crea una clase llamada **Trigonometría** y encapsula en su interior los métodos relacionados con esta área de las Matemáticas, de manera que puedas usar los métodos en español: **sin** por seno, **cos** por coseno, **tan** por tangente, **asin** por arcoseno, **acos** por arcocoseno y **atan** por arctangente. Finalmente, instancia un objeto de la clase y utiliza cada método sacando resultados por consola.

5.4.4. Boolean

El objeto **Boolean** es un objeto contenedor para un valor booleano, un valor lógico **true** o **false**. Por tanto, no se deben confundir los valores booleanos primitivos **true** y **false**, con los valores **true** y **false** del objeto **Boolean**.

El valor pasado a **Boolean** como primer parámetro se convierte en un valor booleano, si es necesario. Si el valor se omite o tiene uno de estos valores: **0**, **-0**, **null**, **false**, **NaN**, **undefined**, **""**, el objeto tiene un valor inicial de **false**. Para todos los demás valores se obtendrá inicialmente un valor de **true**.

```
let b1 = new Boolean(NaN);
let b2 = new Boolean(undefined);
let b3 = new Boolean("");
let b4 = new Boolean([]);
let b5 = new Boolean("false");
```

▶ Boolean {false}
▶ Boolean {false}
▶ Boolean {false}
▶ Boolean {true}
▶ Boolean {true}

Figura 5.15. Resultados tras ejecutar el constructor de Boolean con diferentes valores.

Cuidado también al considerar estas dos expresiones porque no devuelven lo mismo y es un error común pensar lo contrario:

```
let logico1 = false;
let logico2 = new Boolean(false);
console.log(logico1);
console.log(logico2);
if (logico1)
    console.log("logico1 entra");
if (logico2)
    console.log("logico2 entra");
```

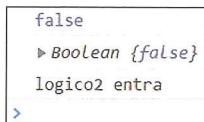


Figura 5.16. Un valor aparentemente false evaluándose como true.

La salida en la consola revela un comportamiento inesperado: aunque inicialmente ambas definiciones de las variables se muestran como **false**, no se evalúan como tal cuando se utilizan en expresiones condicionales. El objeto creado a partir de **Boolean** se ha evaluado como **true**.

La mejor regla que debe recordarse en relación con este objeto es no utilizar un objeto **Boolean** como sinónimo de un booleano primitivo.

5.4.5. Expresiones regulares

Las expresiones regulares, una utilidad que está presente en la mayoría de los lenguajes de programación, también cuentan con un objeto predefinido en JavaScript.

Conceptualmente una expresión regular es un patrón que se utiliza para buscar coincidencias en las cadenas de texto, de manera que puedan resolverse tareas frecuentes como la validación de datos.

Se pueden construir expresiones regulares de dos formas distintas:

1. Usando una expresión regular literal (las barras delimitan la expresión):

```
let erLiteral = /[0-9]/;
```

2. Llamando a la función constructora del objeto **RegExp**:

```
let erObjeto = new RegExp('[0-9]');
```

Por ejemplo, con las expresiones regulares anteriores serán validadas todas aquellas cadenas que contengan algún número, como por ejemplo “101 Dálmatas” o “875”, pero no cadenas como “Faro” o “xMssT_pol#xhN”.

La manera de comprobar si una cadena cumple con los criterios del patrón establecido en la expresión regular es a través del método **test()**, que recibe como parámetro la cadena a comprobar y devuelve **true** en caso de superar la validación, o **false** en caso contrario:

```
let erObjeto = new RegExp("[0-9]");
console.log(erObjeto.test("a"));
console.log(erObjeto.test("almamia"));
console.log(erObjeto.test("alma66Mia"));
console.log(erObjeto.test("987"));
```

Naturalmente, la salida de la pieza de código anterior será **false false true true**.

Debido a la altísima variabilidad de expresiones que se pueden indicar en una cadena de caracteres y las estrategias para validarlas, JavaScript ha desarrollado toda una sintaxis que facilita mucho su uso. A continuación se ven algunos de los elementos más comunes.

Modificador i

El símbolo i se indica cuando al validar letras del alfabeto no se desea distinguir entre mayúsculas y minúsculas:

```
let er = /a/;
console.log(er.test("pizza")); // Escribe true
console.log(er.test("TACO")); // Escribe false
let er2 = /a/i;
console.log(er2.test("pizza")); // Escribe true
console.log(er2.test("TACO")); // Escribe true
```

Modificador ^

El símbolo circunflejo, ^, fuerza que la cadena empiece por el carácter inmediatamente posterior:

```
let er = /^a/;
console.log(er.test("pizza")); // Escribe false
console.log(er.test("TACO")); // Escribe false
console.log(er.test("armario")); // Escribe true
```

Modificador \$

El símbolo del dólar, \$, fuerza a que la cadena termine por el carácter inmediatamente anterior:

```
let er = /pon$/;
console.log(er.test("ponderado")); // Escribe false
console.log(er.test("posicion")); // Escribe false
console.log(er.test("tapon")); // Escribe true
```

Modificador .

El símbolo del punto representa un carácter cualquiera:

```
let er = /ar.on/;
console.log(er.test("arcon")); // Escribe true
console.log(er.test("arpon")); // Escribe true
console.log(er.test("Aaron")); // Escribe false
```

Modificador []

Los símbolos de los corchetes establecen caracteres opcionales. La expresión la cumpliría cualquier cadena que contenga alguno de los elementos indicados entre corchetes:

```
let er = /[aeiou]/;
console.log(er.test("SOS")); // Escribe false
console.log(er.test("col")); // Escribe true
console.log(er.test("Pfff!")); // Escribe false
```

■■■ Modificador [[^]expresión]

El carácter circunflejo como primer elemento de unos corchetes indica un carácter no permitido. Por ejemplo, la siguiente expresión significa que solo se validarán las cadenas que no sean completamente numéricas:

```
let er = /^[^0-9]/;
console.log(er.test("cabo")); // Escribe true
console.log(er.test("526")); // Escribe false
console.log(er.test("bueno")); // Escribe true
console.log(er.test("p4sswOrd")); // Escribe true
```

■■■ Modificadores de cardinalidad

Se trata de símbolos que permiten configurar repeticiones de expresiones (Tabla 5.7).

Tabla 5.7. Símbolos usados para modificar la cardinalidad en una expresión regular

Método	Utilidad
<i>exp?</i>	Halla ninguna o una vez el elemento <i>exp</i> . Por ejemplo, /a?sa?/ coincide con «as» en «pecas» y «asa» en «casados».
<i>exp*</i>	Si se usa inmediatamente después de cualquiera de los cuantificadores *, +, ?, o {}, hace que el cuantificador no sea maximalista (es decir, que coincida con el mínimo número de veces), a diferencia del predeterminado, que es maximalista (coincide con el máximo número de veces).
<i>exp*</i>	Concuerda cero o más veces con el elemento <i>exp</i> . Por ejemplo, /ho*/ coincide con «muchoooo» en «Hace muchoooo calor» y «h» en «El vehículo está ardiendo», pero nada en «Para de ladear».
<i>exp+</i>	Encuentra una o más veces el elemento <i>exp</i> , equivalente a {1,}.
<i>exp+</i>	Por ejemplo, /r+/ coincide con la letra «r» en «Brody» y con todas las letras «r» en «Brrrrrr!».
<i>exp{n}</i>	Donde <i>n</i> es un número entero positivo, concuerda exactamente con <i>n</i> apariciones del elemento <i>exp</i> . Por ejemplo, /r{2}/ no coincide con la «r» de «Brody», pero coincide con todas las «r» de «carro» y las dos primeras «r» en «Brrrrrr!».
<i>exp{n,}</i>	Donde <i>n</i> es un número entero positivo, concuerda con al menos <i>n</i> apariciones del elemento <i>exp</i> . Por ejemplo, /a{2,}/ no coincide con las «a» en «maracaná», pero coincide con todas las «a» en «maamaa» y en «aaaaaaaaaaaaamelo».
<i>exp{m,n}</i>	Donde <i>n</i> es 0 o un número entero positivo, <i>m</i> es un número entero positivo y <i>m > n</i> coincide con al menos <i>n</i> y como máximo <i>m</i> apariciones del elemento <i>exp</i> . Por ejemplo, /e{1,3}/ no coincide con nada en «castaña», la «e» en «mesa», las dos «e» en «peero» y las tres primeras «e» en «eeeeeguro».

Actividad propuesta 5.5

Validación de etiquetas

Un conocido comercio para el que estás realizando su nuevo sitio web, te ha encargado que realices una primera validación de las referencias que aparecen en las etiquetas de sus productos. La etiqueta es de la forma 2022-xrFdS/25_9. Es decir, los 4 primeros caracteres deben ser numéricos, el siguiente debe ser un – los cinco siguientes serán combinaciones de letras minúsculas y mayúsculas, el siguiente carácter será la barra inclinada, los dos siguientes dos números, después un _ y por último un número.

Crea una clase llamada **Etiqueta**, que de momento tenga solo dos propiedades, nombre del artículo y referencia del artículo; y dos métodos, mostrar artículo y validar etiqueta. Luego crea un objeto y comprueba que se validan correctamente las etiquetas.

■■■ Modificador ()

Los símbolos de los paréntesis permiten agrupar expresiones, aumentando la complejidad del patrón. Por ejemplo, para validar una cadena del tipo «maria#5jorge#9», habría que indicarle que el patrón buscado es una cadena de cinco letras de la «a» a la «z», después el símbolo «#», después un número de «0» a «9», y después repetir lo anterior. Toda la expresión que va entre paréntesis es lo que debería repetirse:

```
let er = /([a-z]{5}#[0-9])\{2\}/;
console.log(er.test("maria#5jorge#9")); // Escribe true
console.log(er.test("maria#5jorge#")); // Escribe false
```

■■■ Modificador |

El símbolo de la barra vertical indica una opción, es decir, valida lo que está a su izquierda o lo que está a su derecha. Por ejemplo, para validar un teléfono móvil en España (nueve dígitos que deben comenzar por 6, 7 u 8) se puede recurrir a este código:

```
let er = /(6|7|8)([0-9]\{8\})/;
console.log(er.test("615833678")); // Escribe true
console.log(er.test("715833678")); // Escribe true
console.log(er.test("815833678")); // Escribe true
console.log(er.test("515833678")); // Escribe false
console.log(er.test("915833678")); // Escribe false
console.log(er.test("61583678")); // Escribe false
```

■■■ Modificadores abreviados

Se trata de un conjunto de símbolos a los que precede la barra invertida, que funcionan muy bien con Unicode y permiten escribir expresiones de una forma más ágil (Tabla 5.8).

Tabla 5.8. Símbolos usados para simplificar expresiones regulares

Símbolo	Utilidad
\d	Cualquier dígito numérico.
\D	Cualquier carácter salvo los dígitos numéricos.
\s	Espacio en blanco.
\S	Cualquier carácter salvo el espacio en blanco.
\w	Cualquier carácter alfanumérico: [a-zA-Z0-9].
\W	Cualquier carácter que no sea alfanumérico: [^a-zA-Z0-9].
\0	Carácter nulo.
\n	Carácter de nueva línea.
\t	Carácter tabulador.
\`	El símbolo \.
\"	Comillas dobles.
\'	Comillas simples.
\c	Escapa el carácter c.
\ooo	Carácter Unicode empleando la notación octal.
\xff	Carácter ASCII empleando la notación hexadecimal.
\xffff	Carácter Unicode empleando la notación hexadecimal.

Actividad resuelta 5.4

Validación de correos electrónicos

Crea una función que haciendo uso de las expresiones regulares permita validar el formato de una dirección de correo electrónico. Recuerda que estas son las normas que validan un correo electrónico:

- El carácter @ es obligatorio; separa la primera parte (izquierda) de la segunda (derecha).
- La primera parte:
 - Acepta letras minúsculas y mayúsculas, caracteres numéricos y los caracteres especiales # * + & ‘ ! % @ ? { ^ } ”.
 - Acepta todos los caracteres punto (.) que se deseen, pero no puede ser ni el primer ni el último carácter, y tampoco pueden ir seguidos.
- La segunda parte acepta puntos, dígitos, guiones y letras.

Por ejemplo, hola@tu.casa.net es válido, pero mi.email.140dominio.com no lo es.

Solución

```
function validaEmail(email) {
    var regExp = /^[^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/;
    if (regExp.test(email)) {
        console.log("Formato de email correcto");
    } else {
        console.log("Formato de email incorrecto");
    }
}
validaEmail("hola@tu.casa.net");
validaEmail("mi.email.140dominio.com");
```

Método exec()

El método `exec()` se utiliza para realizar una búsqueda sobre las coincidencias de una expresión regular en una cadena específica, devolviendo un array en caso de éxito o `null` en caso contrario. Básicamente, se trata de una alternativa más potente a `test()` y que muestra más información de las coincidencias.

La siguiente expresión:

```
let exreg = /sendero\s(arenoso).+?noche/ig;
let res = exreg.exec('El sendero arenoso de noche puede ser peligroso');
```

busca «sendero arenoso» seguido de «noche», ignorando los caracteres que encuentre en medio y, además, ignora mayúsculas y minúsculas. La información que contiene el objeto devuelto se observa en la Figura 5.17.

```
(2) ['sendero arenoso de noche', 'arenoso', index: 3, input: 'El
▼ sendero arenoso de noche puede ser peligroso', groups: undefined]
  i
  0: "sendero arenoso de noche"
  1: "arenoso"
  groups: undefined
  index: 3
  input: "El sendero arenoso de noche puede ser peligroso"
  length: 2
  ▶ [[Prototype]]: Array(0)
```

Figura 5.17. Toda la información proporcionada por la salida de `exec()`.

La salida se interpreta de este modo:

- El elemento con índice cero es el primer texto encontrado que cumple la expresión regular.
- La propiedad `index` es un número que indica la primera posición en la que se encontró el texto.
- La propiedad `input` almacena el texto original donde se realizó la búsqueda.
- Los índices mayores que cero indican las coincidencias con las subcadenas buscadas en agrupaciones con paréntesis.