

DESARROLLO WEB EN ENTORNO CLIENTE TEMA3: UTILIZACIÓN DE LOS OBJETOS PREDEFINIDOS DE JAVASCRIPT

Índice

- Funciones predefinidas del lenguaje
- Funciones del usuario
- Objetos definidos por el usuario
- Objetos nativos de JavaScript

1. Funciones predefinidas del lenguaje

3

- JavaScript cuenta con una serie de funciones integradas en el lenguaje.
- Dichas funciones se pueden utilizar sin conocer todas las instrucciones que ejecuta.
- Simplemente se debe conocer el nombre de la función y el resultado que se obtiene al utilizarla.

1. Funciones predefinidas del lenguaje

4

- Las siguientes son algunas de las principales funciones predefinidas de JavaScript:

Funciones Predefinidas	
<code>escape()</code>	<code>Number()</code>
<code>eval()</code>	<code>String()</code>
<code>isFinite()</code>	<code>parseInt()</code>
<code>isNaN()</code>	<code>parseFloat()</code>

1. Funciones predefinidas del lenguaje

5

- `escape()`: recibe como argumento una cadena de caracteres y devuelve esa misma cadena sustituida con su codificación en ASCII.
- `unescape()` sería la función contraria de decodificación.

```
<script type="text/javascript">
  var input = prompt("Introduce una cadena"); //Probar ?
  var inputCodificado = escape(input);
  alert("Cadena codificada: " + inputCodificado);
</script>
```

1. Funciones predefinidas del lenguaje

6

- `eval()`: convierte una cadena que pasamos como argumento en código JavaScript ejecutable.

```
<script type="text/javascript">  
  var input = prompt("Introduce una operación numérica");//2+3  
  var resultado = eval(input);  
  alert ("El resultado de la operación es: " + resultado);  
</script>
```

1. Funciones predefinidas del lenguaje

7

- *isFinite()*: verifica si el número que pasamos como argumento es o no un número finito.



¿Cuál es el rango de valores admitidos en JavaScript?

1. Funciones predefinidas del lenguaje

8

- *isNaN()*: comprueba si el valor que pasamos como argumento es un de tipo numérico.

```
<script type="text/javascript">
  var input = prompt("Introduce un valor numérico: ");
  if (isNaN(input)){
    alert("El dato ingresado no es numérico.");
  }else{
    alert("El dato ingresado es numérico.");
  }
</script>
```


1. Funciones predefinidas del lenguaje

9

- *String()*: convierte el objeto pasado como argumento en una cadena que represente el valor de dicho objeto.

```
<script type="text/javascript">  
  var fecha = new Date()  
  var fechaString = String(fecha)  
  alert("La fecha actual es: "+fechaString);  
</script>
```

1. Funciones predefinidas del lenguaje

10

- *Number()*: convierte el objeto pasado como argumento en un número que represente el valor de dicho objeto.



¿Qué devuelve la función si el parámetro es un objeto de tipo Date?

1. Funciones predefinidas del lenguaje

11

- *parseInt()*: convierte la cadena que pasamos como argumento en un valor numérico de tipo entero.
- Si no se especifica la base, por defecto es base decimal.

```
<script type="text/javascript">  
  var input = prompt("Introduce un valor: "); //55 , 5u, u5  
  var inputParsed = parseInt(input);  
  alert("parseInt("+input+"): "+inputParsed);  
</script>
```

1. Funciones predefinidas del lenguaje

12

- *parseFloat()*: convierte la cadena que pasamos como argumento en un valor numérico de tipo flotante.

```
<script type="text/javascript">
  var input = prompt("Introduce un valor: ");
  var inputParsed = parseFloat(input);
  alert("parseFloat("+input+"): " + inputParsed);
</script>
```

1. Funciones predefinidas del lenguaje

13



A.3.1. Crear un script que muestre la codificación de todas las vocales con tilde.

2. Funciones del usuario

14

- Es posible crear funciones personalizadas diferentes a las funciones predefinidas por el lenguaje.
- Con estas funciones se pueden realizar las tareas que queramos.
- Una tarea se realiza mediante un grupo de instrucciones relacionadas a las cuales debemos dar un nombre.

2. Funciones del usuario

15

- Definición de funciones:
 - ❑ El mejor lugar para definir las funciones es dentro de las etiquetas HTML `<head>` y `</head>`.
 - ❑ El motivo es que el navegador carga siempre primero todo lo que se encuentra entre estas etiquetas.
 - ❑ La definición de una función consta de cinco partes:
 - La palabra clave *function*.
 - El nombre de la función.
 - Los argumentos utilizados.
 - El grupo de instrucciones.
 - La palabra clave *return*.

2. Funciones del usuario

16

- Definición de funciones – Sintaxis:

```
function nombreFunción ([argumentos]){  
    grupo_de_instrucciones;  
    [return valor;]  
}
```

```
<script type="text/javascript">  
    function aplicarIVA(valorProducto, IVA){  
        var productoConIVA = valorProducto * IVA;  
        alert("El precio del producto, aplicando el IVA del " + IVA + " es: " + productoConIVA);  
    }  
</script>
```


2. Funciones del usuario

17

- Invocación de funciones:
 - ❑ Una vez definida la función es necesaria llamarla para que el navegador ejecute el grupo de instrucciones.
 - ❑ Se invoca usando su nombre seguido del paréntesis.
 - ❑ Si tiene argumentos, se deben especificar en el mismo orden en el que se han definido en la función.

2. Funciones del usuario

18

- Invocar una función desde JavaScript:

```
<html><head>  
  <title>Invocar función desde JavaScript</title>  
  <script type="text/javascript">  
    function mi_funcion([args]){  
      //instrucciones  
    }  
  </script></head>  
<body>  
  <script type="text/javascript">  
    mi_funcion([args]);  
  </script>  
</body></html>
```

2. Funciones del usuario

19

- Invocar una función desde HTML:

```
<html>
  <head>
    <title>Invocar función desde JavaScript</title>
    <script type="text/javascript">
      function mi_funcion([args]){
        //instrucciones
      }
    </script>
  </head>
  <body onload="mi_funcion([args])"></body>
</html>
```

2. Funciones del usuario

20



A.3.2. Invocar al método `aplicarIVA(300,1,18)`; de las dos formas indicadas anteriormente.

3. Objetos definidos por el usuario

21

- JavaScript proporciona una serie objetos predefinidos (punto siguiente), sin embargo es posible crear nuevos objetos definidos por el usuario.
- Cada uno de estos objetos puede tener sus propios métodos y propiedades.
- La creación de nuevos objetos resulta útil en el desarrollo de aplicaciones avanzadas.

3. Objetos definidos por el usuario

22

- Declaración e inicialización de los objetos:
 - ❑ Un objeto es una entidad que posee unas propiedades que lo caracterizan y unos métodos que actúan sobre estas propiedades.
 - ❑ Su sintaxis es la siguiente:

```
function mi_objeto (valor_1, valor_2, valor_x){  
    this.propiedad_1 = valor_1;  
    this.propiedad_2 = valor_2;  
    this.propiedad_x = valor_x;  
}
```

```
function Coche(marca_in, modelo_in, anyo_in){  
    this.marca = marca_in;  
    this.modelo = modelo_in;  
    this.anyo = anyo_in;  
}
```

3. Objetos definidos por el usuario

23

- Una vez declarado el nuevo tipo de objeto se pueden crear instancias mediante la palabra clave *new*:

```
var coches = new Array(4);
coches[0] = new Coche("Ferrari", "Scaglietti", "2010");
coches[1] = new Coche("BMW", "Z4", "2010");
coches[2] = new Coche("Seat", "Toledo", "1999");
coches[3] = new Coche("Fiat", "500", "1995");
for(i=0; i<coches.length; i++){
    document.write("Marca: " + coches[i].marca + " - Modelo: " + coches[i].modelo +
        " - Año de fabricación: " + coches[i].anyo + "<br>");
}
```



3. Objetos definidos por el usuario

24

- Es posible añadir otras propiedades a cada instancia del objeto, por ejemplo:

```
function Coche (marca_in, modelo_in, anyo_in){  
  this.marca = marca_in;  
  this.modelo = modelo_in;  
  this.anyo = anyo_in;  
}  
var mi_coche = new coche("Pegeout", "206cc", "2003");  
mi_coche.color = "azul";
```




3. Objetos definidos por el usuario

25

- Dentro de la definición de los objetos, podemos incluir funciones que acceden a las propiedades → *métodos del objeto*.
- Para ello, incluir una función que realice las intruscciones que queramos ejecutar y ,en la definición del objeto, añadimos el nombre de la función a traves de la palabra clave *this*.

```
function imprimirDatos(){  
    document.write("<br>Marca: "+ this.marca);  
    document.write("<br>Modelo: "+ this.modelo);  
    document.write("<br>Año: "+ this.anyo);  
}  
function Coche (marca_in, modelo_in, anyo_in){  
    this.marca = marca_in;  
    this.modelo = modelo_in;  
    this.anyo = anyo_in;  
    this.imprimirDatos=imprimirDatos;  
}  
var miCoche = new Coche("Pegeout", "206cc", "2003");  
miCoche.imprimirDatos();  
</script>
```



2. Funciones del usuario

26



A.3.3. Crear un script que defina un objeto llamado *ProductoAlimenticio*. Este objeto debe presentar las propiedades *código*, *nombre* y *precio*, además del método *imprimirDatos*, el cual escribe por pantalla los valores de sus propiedades. Posteriormente, crear 3 instancias de este objeto y guárdelas en un *array*. Con la ayuda de un bucle usar el método *imprimirDatos* para mostrar por pantalla los valores de los tres objetos.

4. Objetos nativos de JavaScript

27

- Objetos representan estructuras que permiten agrupar dentro de esta tanto datos, como funcionalidad.
- JavaScript proporciona una serie de objetos definidos nativamente que no dependen del navegador (Ej. String, Date, Math).
- Para crear un objeto se utiliza la palabra clave *new*. Ejemplo:
 - ❑ `var mi_objeto = new Object();`

4. Objetos nativos de JavaScript

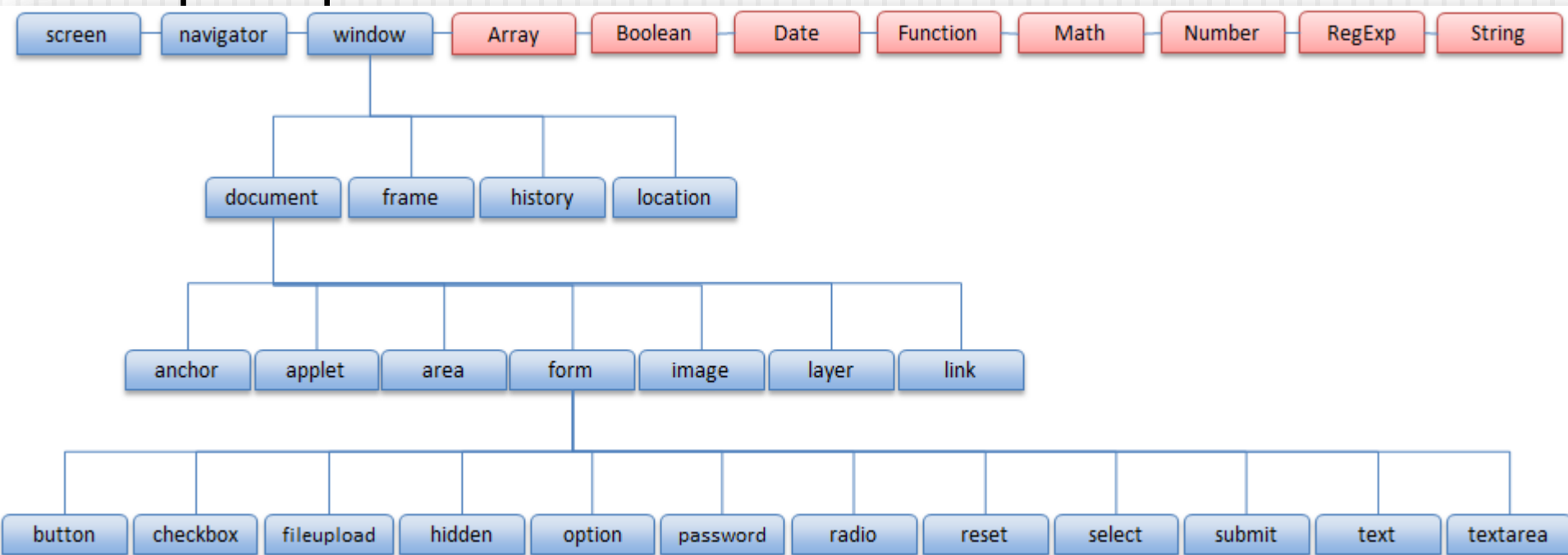
28

- En JavaScript se accede a las propiedades y a los métodos de los objetos mediante el operador punto (“.”):
 - ❑ *mi_objeto.nombre_propiedad;*
 - ❑ *mi_objeto.nombre_función(parámetros);*

4. Objetos nativos de JavaScript

29

- Los objetos de JavaScript se ordenan de modo jerárquico:



4. Objetos nativos de JavaScript

30

➤ Date:

- ❑ Permite realizar controles relacionados con el tiempo en las aplicaciones web.
- ❑ Cuenta con una serie de métodos divididos en tres subconjuntos:
 - Métodos de lectura.
 - Métodos de escritura.
 - Métodos de conversión.

4. Objetos nativos de JavaScript

31

□ [Link](#)

Métodos

<code>getDate()</code>	<code>getTime()</code>	<code>getUTCMonth()</code>	<code>setMonth()</code>	<code>setUTCMonth()</code>
<code>getDay()</code>	<code>getTimezoneOffset()</code>	<code>getUTCSeconds()</code>	<code>setSeconds()</code>	<code>setUTCSeconds()</code>
<code>getFullYear()</code>	<code>getUTCDate()</code>	<code>parse()</code>	<code>setTime()</code>	<code>toDateStr()</code>
<code>getHours()</code>	<code>getUTCDay()</code>	<code>setDate()</code>	<code>setUTCDate()</code>	<code>toLocaleDateString()</code>
<code>getMilliseconds()</code>	<code>getUTCFullYear()</code>	<code>setFullYear()</code>	<code>setUTCFullYear()</code>	<code>toLocaleTimeString()</code>
<code>getMinutes()</code>	<code>getUTCHours()</code>	<code>setHours()</code>	<code>setUTCHours()</code>	<code>toLocaleStr()</code>
<code>getMonth()</code>	<code>getUTCMilliseconds()</code>	<code>setMilliseconds()</code>	<code>setUTCMilliseconds()</code>	<code>toTimeString()</code>
<code>getSeconds()</code>	<code>getUTCMinutes()</code>	<code>setMinutes()</code>	<code>setUTCMinutes()</code>	<code>toUTCStr()</code>


4. Objetos nativos de JavaScript

32

```
<script type="text/javascript">
  var fechaActual= new Date();
  var fechaMaya= new Date(2017,11,21);
  alert("La fecha actual es: " + fechaActual);
  alert("El calendario Maya termina el: " + fechaActual);
  var tiempoRestante=fechaMaya-fechaActual;
  alert("Quedan " + tiempoRestante + " milisegundos para que termine el calendario Maya");
</script>
```


4. Objetos nativos de JavaScript

33



A.3.4. Calcular los milisegundos hasta su próximo cumpleaños y muéstralos por pantalla.



A.3.5. Calcular el tiempo desde que se carga el documento hasta que el usuario pulsa un botón en la pantalla.

4. Objetos nativos de JavaScript

34

➤ Math:

- ❑ Permite realizar operaciones matemáticas complejas en JavaScript.

4. Objetos nativos de JavaScript

35

□ [Link](#)

Métodos		
<code>abs()</code>	<code>exp()</code>	<code>random()</code>
<code>acos()</code>	<code>floor()</code>	<code>round()</code>
<code>asin()</code>	<code>log()</code>	<code>sin()</code>
<code>atan()</code>	<code>max()</code>	<code>sqrt()</code>
<code>ceil()</code>	<code>min()</code>	<code>tan()</code>
<code>cos()</code>	<code>pow()</code>	

Propiedades
<code>E</code>
<code>LN2</code>
<code>LN10</code>
<code>LOG2E</code>
<code>LOG10E</code>
<code>PI</code>
<code>SQRT1_2</code>
<code>SQRT2</code>

4. Objetos nativos de JavaScript

36

```
<script type="text/javascript">  
  var r= prompt("Ingresa el radio de un círculo en cms:");  
  var area= Math.PI * Math.pow(r,2);  
  
  alert("El área del círculo es de: " + area + " centímetros cuadrados");  
</script>
```

4. Objetos nativos de JavaScript

37



A.3.6. En el ejemplo anterior, hemos usado la clase *Math* para calcular el área del círculo. Realiza una aplicación similar pero que calcule el área de un triángulo en el cual la base y la altura las proporcione el usuario.

4. Objetos nativos de JavaScript

38

➤ Number:

- ❑ Permite realizar tareas relacionadas con tipos de datos numéricos.

Métodos

`toExponential()`

`toFixed()`

`toPrecision()`

Propiedades

`MAX_VALUE`

`MIN_VALUE`

`NaN`

`NEGATIVE_INFINITY`

`POSITIVE_INFINITY`

4. Objetos nativos de JavaScript

39

```
<script type="text/javascript">
  alert("Propiedad MAX_VALUE: " + Number.MAX_VALUE);
  alert("Propiedad MIN_VALUE: " + Number.MIN_VALUE);
  alert("Propiedad NaN: " + Number.NaN);
  alert("Propiedad NEGATIVE_INFINITY: " + Number.NEGATIVE_INFINITY);
  alert("Propiedad POSITIVE_INFINITY: " + Number.POSITIVE_INFINITY);
  var n1= new Number(Math.PI);
  alert("Pi formateado: "+ n1.toPrecision(3));

</script>
```

4. Objetos nativos de JavaScript

40

➤ Array:


- ❑ Típicamente, colecciones ordenadas de datos homogéneas. Sin embargo, *en JavaScript, se puede tener datos de distinto tipo.*

```
var vacio=[];  
//var cars = ["Saab", "Volvo", "BMW"];  
var cars = [  
    "Saab",  
    "Volvo",  
    "BMW"  
];
```

- ❑ La propiedad *length* es editable, tanto para eliminar como para añadir nuevos elementos

4. Objetos nativos de JavaScript

41



A.3.7. Sobre el ejemplo presentado a continuación, añadir al arrays días, para completar la semana. Utilizar el propio array sin necesidad de crear uno nuevo. Aprovechar que la propiedad *length* es editable. Recorrer el array para imprimir todos los días de la semana.

```
var dias = [  
    "Lunes",  
    "Martes",  
    "Miércoles",  
    "Jueves",  
    "Viernes",  
];  
console.log("Dia "+dias[6]);|
```

4. Objetos nativos de JavaScript

42

❏ Métodos: [Link](#)



```
<script>
var fruits = ["Apple", "Banana"];

console.log(fruits.length);
//2

var first = fruits[0];
console.log(first);
// Apple

var last = fruits[fruits.length - 1];
console.log(last);
// Banana

fruits.forEach(function (item, index, array) {
    console.log(item, index);
});
// Apple 0
// Banana 1

var newLength = fruits.push("Orange");
console.log(newLength);
// ["Apple", "Banana", "Orange"]

var last = fruits.pop(); // remove Orange (from the end)
console.log(fruits.length);
// ["Apple", "Banana"];
```

4. Objetos nativos de JavaScript

43

```
var first = fruits.shift(); // remove Apple from the front
console.log(fruits.length);
// ["Banana"];

var newLength = fruits.unshift("Strawberry") // add to the front
console.log(fruits.length);
// ["Strawberry", "Banana"];

fruits.push("Mango");
console.log(fruits.length);
// ["Strawberry", "Banana", "Mango"]

var pos = fruits.indexOf("Banana");
console.log(pos);
// 1

var removedItem = fruits.splice(pos, 1); // this is how to remove an item
console.log(fruits.length);
// ["Strawberry", "Mango"]


var shallowCopy = fruits.slice(); // this is how to make a copy
// ["Strawberry", "Mango"]

</script>
```




4. Objetos nativos de JavaScript

44



A.3.8. Partiendo del *array* *dias*, el cual contiene todos los literales de la semana, dividir el *array* en dos partes, laborables y festivos.



A.3.9. Ordenar los alumnos por orden alfabético (función *Array.sort()*) y mostrar el contenido como cadena de texto (*Array.toString()*). También invertir el orden (*Array.reverse()*) y mostrarlo de nuevo.

Sanchez – Julian

López – Luis

Cuesta – José

4. Objetos nativos de JavaScript

45

➤ String:

❑ Permite manipular las cadenas de texto.

Métodos			
<code>anchor()</code>	<code>fixed()</code>	<code>link()</code>	<code>strike()</code>
<code>big()</code>	<code>fontcolor()</code>	<code>match()</code>	<code>sub()</code>
<code>blink()</code>	<code>fontsize()</code>	<code>replace()</code>	<code>substr()</code>
<code>bold()</code>	<code>fromCharCode()</code>	<code>search()</code>	<code>substring()</code>
<code>charAt()</code>	<code>indexOf()</code>	<code>slice()</code>	<code>sup()</code>
<code>charCodeAt()</code>	<code>italics()</code>	<code>small()</code>	<code>toLowerCase()</code>
<code>concat()</code>	<code>lastIndexOf()</code>	<code>split()</code>	<code>toUpperCase()</code>

Propiedades
<code>Length</code>

4. Objetos nativos de JavaScript

46

```
<script type="text/javascript">
  var texto= new String("Prueba de texto");
  document.write("Número de letras de la cadena de texto: " + texto.length + "</br>");
  document.write("Cursive: " + texto.italics() + "</br>");
  document.write("Negrita: " + texto.bold() + "</br>");
  document.write("Rojo: " + texto.fontcolor("#FF0000") + "</br>");
  document.write("Grande: " + texto.fontSize("20") + "</br>");
  document.write("Tachado: " + texto.strike() + "</br>");
</script>
```



4. Objetos nativos de JavaScript

47



A.3.10. A partir de las siguientes cadenas de texto ("lunes", "martes", "miercoles"), realizar las operaciones necesarias para obtener ("Lunes", "Martes", "Miercoles").

4. Objetos nativos de JavaScript

48

➤ Expresiones regulares:

- ❑ Permite, mediante un texto con una notación y sintaxis concreta, definir un patrón de caracteres o un carácter en particular.
- ❑ Permite comparar la expresión regular con textos con el fin de verificar que estos contienen el patrón definido.
- ❑ Utilización: desde operaciones de validación de datos a búsqueda de textos.

4. Objetos nativos de JavaScript

49

- ❑ Sintaxis: */patrón/modificadores;*

```
var patron=/[jc]ava/i;  
//java, cava, Java, etc
```

- ❑ Modificadores:

- ❑ **g** (global): Se aplica a toda la cadena en lugar de detenerse al encontrar la primera correspondencia correcta.
- ❑ **i** (insensible) a mayúsculas y minúsculas.
- ❑ **m** (multilínea): El patrón sigue buscando correspondencias a pesar de haber alcanzado el final de la línea.

4. Objetos nativos de JavaScript

50

❑ Propiedades:

<u>global</u>	Checks whether the "g" modifier is set
<u>ignoreCase</u>	Checks whether the "i" modifier is set
<u>lastIndex</u>	Specifies the index at which to start the next match
<u>multiline</u>	Checks whether the "m" modifier is set
<u>source</u>	Returns the text of the RegExp pattern

❑ Metacaracteres: [Link](#)

4. Objetos nativos de JavaScript

51

❏ Métodos:

Método	Descripción
<code>exec</code>	Un método <code>RegExp</code> que ejecuta una búsqueda por una coincidencia en una cadena. Devuelve un array de información.
<code>test</code>	Un método <code>RegExp</code> que verifica una coincidencia en una cadena. Devuelve <code>true</code> o <code>false</code> .
<code>match</code>	Un método <code>String</code> que ejecuta una búsqueda por una coincidencia en una cadena. Devuelve un array de información o <code>null</code> si no existe coincidencia alguna.
<code>search</code>	Un método <code>String</code> que verifica una coincidencia en una cadena. Devuelve el índice de la coincidencia, o <code>-1</code> si la búsqueda falla.
<code>replace</code>	Un método <code>String</code> que ejecuta una búsqueda por una coincidencia en una cadena, y reemplaza la subcadena encontrada con una subcadena de reemplazo.
<code>split</code>	Un método <code>String</code> que utiliza una expresión regular o una cadena fija para cortar una cadena y colocarlo en un array de subcadenas.

4. Objetos nativos de JavaScript

52

- ❑ Ej. Función que permite validar las direcciones de correo electrónico:

```
<script type="text/javascript">
  function validarCorreo(correo){
    patron=/^[a-z]+@{1}[a-z]+\.[a-z]{3}$/;
    exprReg= new RegExp(patron);
    if (correo.match(exprReg)) {
      console.log("Correo valido");
    }else{
      console.log("Correo con formato incorrecto");
    }
  }
  validarCorreo("micorreo@gmail.com");
  validarCorreo("micorreo-gmail.com");
  validarCorreo("micorreo@yahoo.es");
  validarCorreo("micorreo@gmail.com@yahoo.es");
</script>
```

4. Objetos nativos de JavaScript

53



A.3.11. Sobre el ejemplo de la página anterior modificarlo para que cumpla estas dos nuevas condiciones:

- En los caracteres antes de la @, debe permitir caracteres alfabéticos a-z, numéricos 0-9 y el guion bajo _.
- Además el dominio debe poder albergar caracteres de una longitud variable entre 2 y 4.