Objetos y Arrays en JavaScript

1. INTRODUCCIÓN

Utilizando **JavaScript**, puedes hacer que tus páginas web sean más interactivas, ejecutando código en el ordenador del usuario (parte cliente) y no en el servidor web.

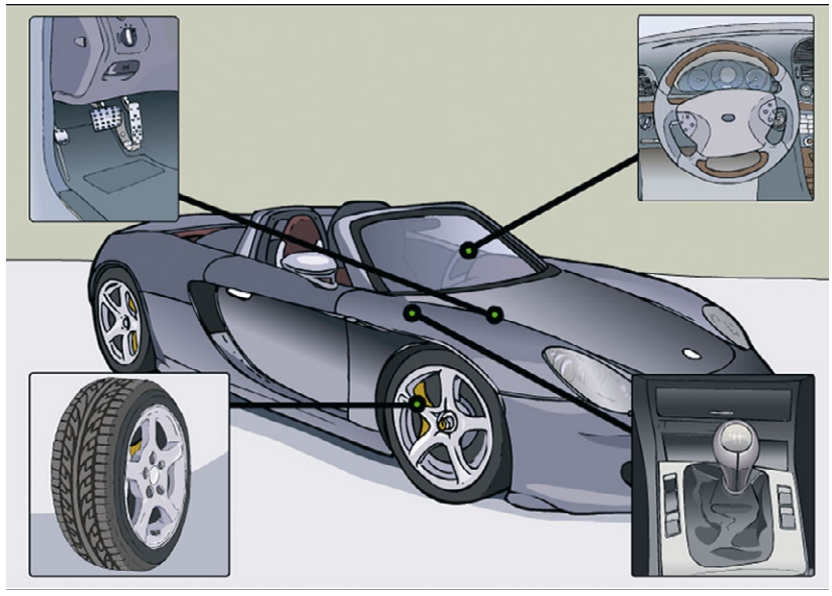
Para ello, el navegador proporciona lo que se conoce como los "*objetos del navegador*".

Se trata de una interfaz de programación que nos permite acceder y modificar el contenido de un documento HTML. Para ello, el navegador representa cualquiera elemento de la página (y también del propio navegador, como después veremos) como un **objeto**. El diseñador utiliza el lenguaje **JavaScript** para acceder a esos objetos y conseguir el propósito que persigue.

Un **objeto** es una entidad en la que se pueden diferenciar tres componentes:

- **Propiedades**: permiten acceder a características que definen la apariencia visual de los objetos u otros detalles de importancia.
- **Métodos**: son funciones que establecen el comportamiento de los objetos.
- **Manejadores de evento**: permiten establecer qué debe realizarse cuando ocurre cierta circunstancia.

Esto es una forma sencilla de describir los elementos de una página web, permitiendo acceder a ellos desde el código. Además, es una forma similar a la que utilizarías para describir un objeto en la realidad.



Por ejemplo, piensa cómo describirías un *coche*. Podrías indicar que tiene 4 ruedas, carrocería, motor, capacidad del maletero, etc. Estas serían las **propiedades** del coche.

Indicarías que puedes acelerar para conseguir cierta velocidad y frenar para disminuir dicha velocidad; que puedes girar y seguir recto, etc. Estos serían los **métodos** del coche.

Finalmente podrías indicar que cada cierto tiempo es necesario reponer gasolina, que el usuario pulsa en el pedal del acelerador o del freno, etc. Estos serían los **manejadores de eventos**.

La misma idea se aplica a los objetos del navegador. Así, cuando ejecutas **Internet Explorer**, **Firefox** o cualquier otro navegador, se crea un objeto **window** que representa la ventana donde se carga dicha aplicación.

Podrás acceder a algunas de las propiedades de este objeto, como puede ser el texto que aparece en la barra de estado; podrás utilizar alguno de sus métodos, como **open**, que permite abrir una nueva ventana; o podrás establecer qué debe ocurrir cuando se cierra la ventana a través del manejador de eventos **unload**.

No todos los objetos del navegador tienen los tres componentes que hemos mencionado. En muchos casos solo se dispone de sus propiedades y de algún método.

Y, si esto no fuera suficiente, también tienes la posibilidad de crear tus propios objetos.

Sin embargo, nosotros nos vamos a centrar en los objetos del navegador, accediendo a sus propiedades, utilizando sus métodos y configurando sus manejadores de evento.

En JavaScript existen otros tipos de objetos que permiten realizar tareas específicas. Entre estos objetos podrás encontrar el objeto **Date**, que permite manejar fechas; **Math**, que proporciona funciones matemáticas; y **String**, que permite trabajar con cadenas de caracteres.

2. LA JERARQUÍA DE OBJETOS

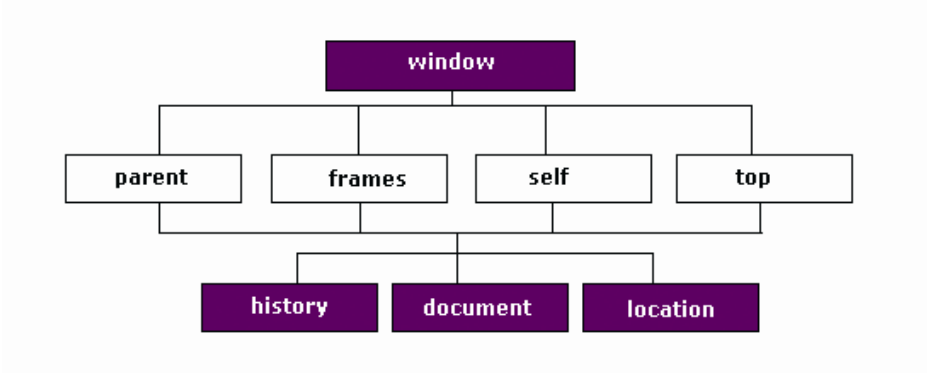
Los navegadores proporcionan objetos distribuidos a través de una jerarquía de objetos en la que en la parte superior aparece el objeto **window**.

Esta jerarquía es una forma de dividir los objetos que puedes utilizar a través de un lenguaje de guiones como JavaScript. Cada uno de estos objetos tiene sus propias propiedades, métodos y eventos.

Sin embargo, a diferencia de otros lenguajes orientados a objetos, se trata de una jerarquía de **composición**.

Esto quiere decir que no representa relaciones de "*herencia*" entre los objetos, sino que los objetos situados en un nivel inferior son propiedades de los objetos situados en un nivel superior.

Observa lo que esto significa: una propiedad de un objeto puede ser, a su vez, otro objeto. Esta es la situación que puedes ver en la figura adjunta.



Por ejemplo, el objeto **window** tiene entre sus propiedades a los objetos **history**, **document** y **location**, que son los que normalmente se utilizan en el código JavaScript.

Estos objetos se crean cuando la página web se carga en el navegador. Lo que quiere decir que se puede hacer referencia a un determinado objeto solo después de que ese objeto se muestre en la página web. Si se hace antes, JavaScript producirá un error indicando que el objeto no tiene propiedades.

De forma resumida podemos indicar que:

- El objeto **window** es el objeto situado en el nivel superior de la jerarquía. Contiene propiedades que se aplican a toda la ventana del navegador. Por ejemplo, la barra de estado es una propiedad de este objeto.

  Sin embargo, debes tener en cuenta que, cuando se utilizan **frames**, es posible tener más de un objeto **window** en la misma página web.

- El objeto **history** contiene propiedades de las direcciones web que ha visitado el usuario. Si la página web utiliza frames, cada uno de estos tiene su propia lista historial.

- El objeto **document** es el más importante dentro de la jerarquía ya que contiene propiedades de la página web que está cargada en el navegador: hipervínculos, imágenes, texto, formularios, etc.

  Cuando se utilizan frames, cada uno de estos tiene su propio objeto **document**.

- El objeto **location** permite acceder a las propiedades de la dirección web actual, es decir, de la dirección de la página que está cargada en un momento dado.

Estudiaremos todos estos objetos.

3. PROPIEDADES Y EVENTOS

Ya hay suficiente teoría. Vamos a ver todo esto en el código JavaScript. Para ello, utilizaremos el objeto **window**, programando una de sus propiedades, que es la barra de estado.

Como se ha comentado, una de las propiedades del objeto **window** es la barra de estado. La barra de estado se encuentra en la parte inferior del navegador y sirve para informar al usuario de algunos acontecimientos: que la página se ha cargado, adónde apunta un determinado hipervínculo, etc.

La barra de estado es una propiedad directa del objeto **window** y tiene el nombre **status**.

La forma de acceder a las propiedades de un objeto es a través de la sintaxis **objeto.propiedad**. Es decir, utilizamos el punto (.) para separar el nombre del objeto del nombre de la propiedad.

Por lo tanto, si deseamos acceder a la propiedad **status** del objeto **window**, deberemos hacerlo con la sintaxis **window.status**

Estudia el siguiente código JavaScript:

```
<script type="text/javascript">
<!--
function ShowMessage()
{
 window.status = "Tenga en cuenta que puede tardar un poco.";
 return true;
}
//-->
</script>
```

Por ahora lo único que hemos hecho es definir una función que muestra un mensaje en la barra de estado utilizando la propiedad **status** del objeto **window**.

Observa cómo se ha utilizado la sintaxis del punto para establecer esta propiedad.

Lo lógico es mostrar el mensaje en la barra de estado cuando ocurra algo. Decir esto es exactamente lo mismo que indicar que se debe programar un **manejador de evento**.

Así, *"cuando ocurra algo"* es lo mismo que decir *"cuando suceda tal evento"*. Por ejemplo, vamos a hacer que el mensaje se muestre cuando el usuario sitúe su ratón por encima de un hipervínculo.

```
<a href="butterfly.avi" onmouseover="return
ShowMessage();">Pulsa aquí para ver un bonito vídeo.</a>
```

Aunque incluir los manejadores de evento directamente en el código HTML es muy sencillo, no es la mejor forma de hacerlo.

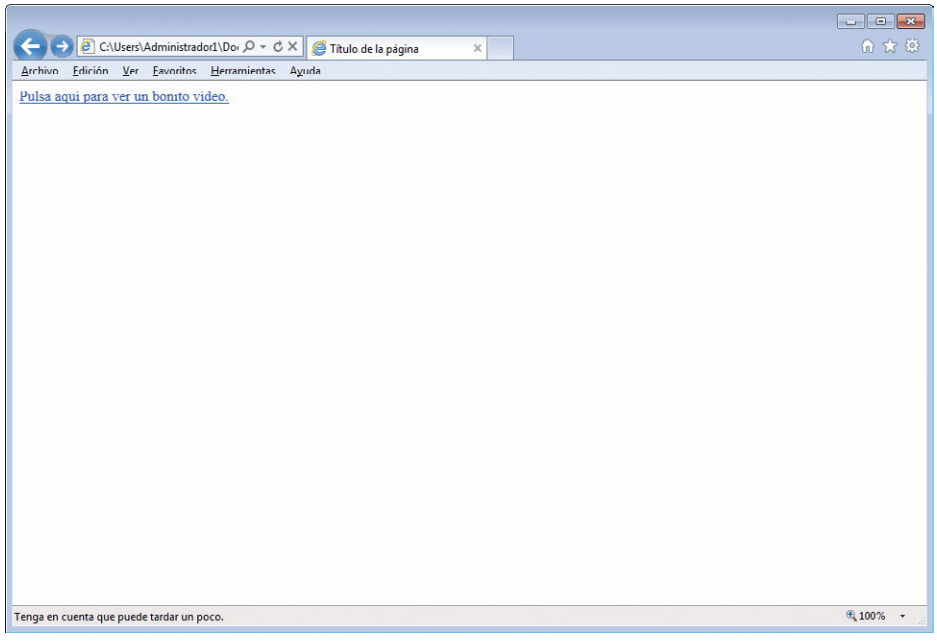
Veremos que es conveniente escribir los manejadores de evento en el código JavaScript, sin mezclarlo con el código de las etiquetas HTML.

Vayamos por partes: **onmouseover** es el nombre de un manejador de evento que coincide con el evento que sucede al situar el ratón por encima del hipervínculo (**onmouseover** significa *"cuando el ratón está por encima"*).

Con esta línea de código estamos indicando qué debe suceder en ese momento. En este caso, llamamos a la función **ShowMessage**.

Por lo tanto, la idea es que se ejecute la función **ShowMessage** cuando ocurra el evento de situar el ratón por encima del hipervínculo. Esto se consigue con el manejador de evento **onmouseover**.

En este ejemplo, el mensaje se mantiene en la barra de estado tras la primera vez que se muestra. Podrías cambiar el mensaje o eliminarlo (cadena vacía) en el evento **onmouseout**, es decir, cuando el puntero del ratón sale de la superficie del hipervínculo.



Existe un manejador de evento asociado con cada evento reconocido en JavaScript. Los nombres de dichos manejadores se construyen de la siguiente forma: *on + nombre_evento*.

Así, tenemos: **onabort**, **onblur**, **onclick**, **onchange**, **onerror**, **onfocus**, **onload**, **onmouseout**, **onmouseover**, **onselect**, **onsubmit** y **onunload**.

Seguramente ahora entenderás mejor los conceptos de propiedad y de evento.

En este capítulo hemos visto cómo utilizar la propiedad **status** del objeto window y el evento **onmouseover** de un hipervínculo.

A continuación se describen algunos eventos soportados en JavaScript:

abort: ocurre cuando el usuario detiene la carga de una imagen (por ejemplo, pulsando en el botón *Detener* del navegador).

blur: ocurre cuando se pierde el foco. Esto puede ocurrir con una ventana cuando deja de ser la ventana activa o en un campo de formulario, cuando deja de ser el elemento en el que se está trabajando.

click: ocurre cuando el usuario hace clic con el ratón en un hipervínculo, en un área de un mapa de imagen o en un campo de formulario.

change: ocurre cuando el usuario cambia el valor de un campo de formulario.

error: ocurre si existe un error al cargar una imagen.

focus: ocurre cuando el usuario da el foco a una ventana o a un campo de formulario (es el evento contrario a blur). Cuando se da el foco a un objeto, lo pierde otro.

load: ocurre cuando la página o imagen ha acabado de cargarse en la ventana del navegador.

mouseout: ocurre cuando el usuario mueve el puntero del ratón desde el interior de un hipervínculo o área de un mapa de imagen hacia fuera de este.

mouseover: ocurre cuando el usuario mueve el puntero del ratón por encima de un hipervínculo o área de un mapa de imagen (es el evento contrario a mouseout).

select: ocurre cuando el usuario selecciona un campo de formulario.

submit: ocurre cuando el usuario pulsa el botón *Enviar* en un formulario. De esta forma, toda la información es enviada al servidor.

unload: ocurre cuando el usuario abandona una página.

4. MÉTODOS

Ya hemos visto cómo utilizar las propiedades y cómo programar manejadores de evento. Nos falta utilizar el último componente de un objeto: los **métodos**.

Uno de los métodos del objeto **window** es **open**. Con este método puedes crear una nueva ventana e incluir el contenido que desees en la misma.

Sin embargo, debes tener en cuenta que la nueva ventana creada con este método no mantiene ninguna relación con la ventana donde está situado el código que la ha creado.

Veamos el código del ejemplo:

```
<!DOCTYPE HTML>
<html>
<head>
<title>Creando nuevas ventanas</title>
<script type="text/javascript">
<!--
var nuevaVentana = null;
function AbrirVentana()
{
   nuevaVentana = window.open("", "nueva",
                              "width=100,height=100");
}
function CerrarVentana()
{
  if (nuevaVentana)
  {
    nuevaVentana.close();
    nuevaVentana = null;
  }
}
//-->
</script>
</head>
<body>
<form>
<input type="button" value="Abrir ventana"
 onclick="AbrirVentana();">
<input type="button" value="Cerrar ventana"
 onclick="CerrarVentana();">
</form>
</body>
</html>
```

En esta página web se ha creado un formulario con dos botones: uno para abrir una nueva ventana y otro para cerrarla.

Definimos una función de nombre **AbrirVentana**, la cual utiliza el método **open** del objeto **window** para crear una ventana nueva .

Observa que un método tiene la misma sintaxis que una función, es decir, un nombre y, opcionalmente entre paréntesis, la lista de parámetros.

En definitiva, los objetos son una forma de unir datos y funciones en una única entidad. Así, las propiedades actúan como variables (datos) y los métodos como funciones.

El método **open** utiliza tres parámetros:

window.open("dirección", "nombre" [, "características"])

El último parámetro es opcional, por lo que aquí aparece entre corchetes:

- **Dirección** es una cadena de texto que indica el documento que se cargará en la ventana nueva. Puede tener como valor una dirección de una página en la World Wide Web, en el disco duro o en cualquier otro lugar.

  Puedes establecerla de forma absoluta o de forma relativa.

- **Nombre** es una cadena de texto que permite identificar a la nueva ventana. Sin embargo, este nombre no se utiliza en el código JavaScript, sino en el HTML. Por ejemplo, podrías utilizarlo para el atributo **TARGET** de los hipervínculos.

- **Características** es una cadena de texto en la que se indican las propiedades de la nueva ventana: si debe aparecer la barra de herramientas, la de dirección, la anchura y la altura, etc.

  Todas las características deben aparecer entre comas, pero sin espacios en blanco entre ellas.

A continuación puedes encontrar algunas de las características más utilizadas a la hora de crear una ventana:

Característica	Tipo de datos	Descripción
toolbar	yes/no	Representa la barra de herramientas del navegador, con los botones Atrás/Adelante.
location	yes/no	Representa la barra de direcciones.
directories	yes/no	Representa la barra con botones como Novedades.
status	yes/no	Representa la barra de estado del navegador.

menubar	yes/no	Representa la barra de menús del navegador.
scrollbars	yes/no	Indica si deben aparecer las barras de desplazamiento cuando sea necesario.
resizable	yes/no	Indica si el usuario puede cambiar el tamaño de la ventana.
width	entero	Indica la anchura de la ventana en píxeles.
height	entero	Indica la altura de la ventana en píxeles.

Como puedes ver, la mayoría son valores booleanos o **yes/no**, esto es, pueden tener el valor yes (activado) o no (desactivado). Por ejemplo:

```
window.open("", "",
            "toolbar,status=no,width=500,heigh=200")
```

indica que se muestre la barra de herramientas, que no lo haga la barra de estado y establece las dimensiones de la ventana. Observa cómo es lo mismo toolbar=yes que simplemente toolbar. Cuando no se da un valor a una característica booleana, quiere decir que está estableciendo el valor yes (activado).

Además, observa cómo se asigna a una variable de nombre **nuevaVentana** el resultado de ejecutar el método **open**. De esta forma podremos referirnos a la nueva ventana en el código JavaScript.

Esta variable debe ser global ya que, si se declara en el interior de la función, no podría utilizarse en el resto del código del *script*.

Con la función **CerrarVentana** lo que hacemos es cerrar la ventana solo en el caso de que siga existiendo. Debes tener en cuenta que se puede producir un error al intentar crear una nueva ventana, por lo que, en este caso, la variable seguiría teniendo el valor **null**.

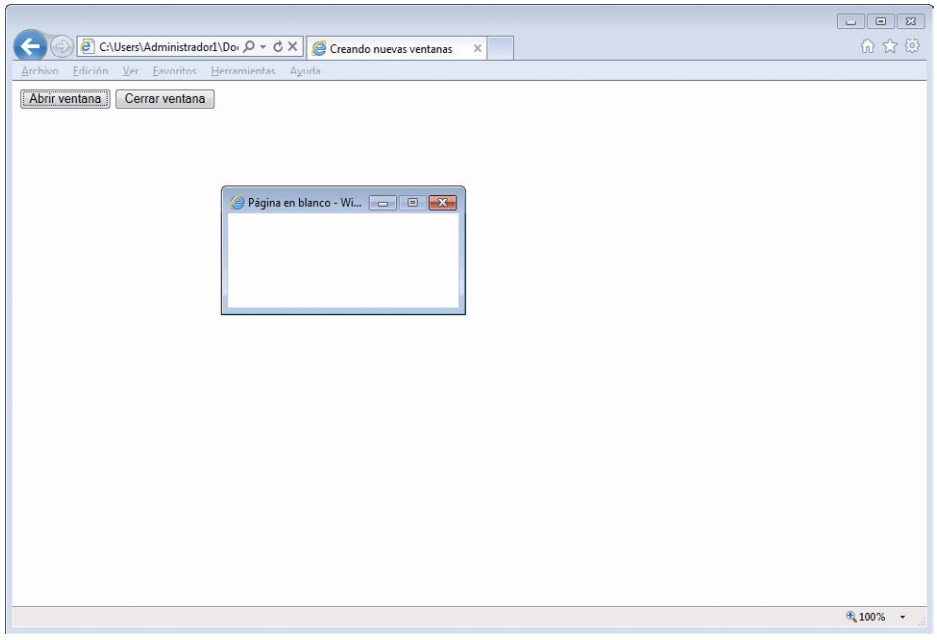
Después se utiliza el método **close**, que no necesita ningún parámetro, y finalmente se vuelve a establecer al valor **null** para que no se intente cerrar una ventana que no existe.

Observa cómo se utiliza la variable **nuevaVentana** para indicar sobre qué ventana se está ejecutando el método **close**.

Esta es la única forma que existe de referirnos a una ventana en particular. Por ello, la variable debe ser global y no local.

Nos queda por ver cómo asociar este código a la pulsación de dichos botones. Como ya te imaginarás, deberemos utilizar un manejador de evento de estos botones:

```
<input type="button" value="Abrir ventana"
 onclick="AbrirVentana();">
<input type="button" value="Cerrar ventana"
 onclick="CerrarVentana();">
```



Por lo tanto, debes pensar en un método como en una función de un objeto que permite realizar alguna acción determinada. Es la parte del objeto que nos va a permitir establecer su comportamiento.

Ya habías utilizado algún que otro método anteriormente.

Por ejemplo, el método **write** del objeto **document**.

Por otra parte, los cuadros de diálogo **alert**, **confirm** y **prompt** también son métodos del objeto **window**.

Cuando se tiene un único objeto **window**, no es necesario indicarlo expresamente, por lo que es equivalente **`window.alert()`** que simplemente **`alert()`**.

Observa cómo la notación para utilizar una propiedad o un método es la misma. JavaScript distingue entre uno u otro caso según el nombre que aparece tras el punto.

Finalmente, fíjate que has utilizado un método para crear un nuevo objeto **window**. El nuevo objeto tendrá sus propios métodos, propiedades y eventos.

5. ARRAYS

Otro concepto que debemos conocer para poder escribir código en JavaScript es el de **array**.

Aunque los tipos básicos de JavaScript son muy útiles, en ocasiones desearás disponer de tipos más complejos como los objetos y los arrays.

De forma sencilla, un **array** es una colección indexada de elementos.

A diferencia de otros lenguajes de programación, en JavaScript cada uno de los elementos de un array puede ser de un tipo de datos distinto. Esto se debe a que no es necesario indicar el tipo de datos al declarar una variable.

Piensa en un array como en una lista de elementos. Puedes acceder a cada uno de esos elementos a partir de un **índice** que indica su posición en la lista.

Estudia este sencillo ejemplo:

```
<!DOCTYPE HTML>
<html>
<head>
<title>Trabajando con arrays</title>
</head>
<body>
<script type="text/javascript">
<!--
var a0 = 0;
var a1 = 1;
var a2 = 2;
var a3 = 3;
var a4 = 4;
var a5 = 5;
var suma = a0 + a1 + a2 + a3 + a4 + a5;
document.write(suma);
//-->
</script>
</body>
</html>
```

Así pues, simplemente se declaran **6** variables, estableciendo un valor numérico para cada una de ellas.

Seguidamente se realiza la suma aritmética de todos esos valores, se almacena en otra variable y se imprime el resultado en la página web.

Aunque el código es correcto, se podría utilizar un **array de 6 elementos** en lugar de utilizar 6 variables independientes. Esto, además de reducir el código, es una forma de indicar que los elementos están relacionados.

Fíjate cómo se haría...

```
var a = new Array(6);
a[0] = 0;
a[1] = 1;
a[2] = 2;
a[3] = 3;
a[4] = 4;
a[5] = 5;
var suma = a[0] + a[1] + a[2] + a[3] + a[4] + a[5];
document.write(suma);
```

En la primera línea vemos cómo crear un array. Para ello, se utiliza la palabra clave **new**, el nombre **Array** y, entre paréntesis, el número de elementos del array.

En este caso estamos creando un array de 6 elementos cuyos índices van del **0** al **5**. Y es que el primer elemento tiene el índice **0**, por lo que el sexto elemento tendrá el índice 5.

En las siguientes líneas se asigna el valor de cada uno de los elementos del array. Para acceder a esos elementos, se utiliza la notación **nombreArray[índice]**.

Es decir, el nombre del arrray y, entre corchetes, el índice. Por ejemplo, con **`a[0]`** estamos accediendo al primer elemento del array **a**.

Finalmente, en las dos líneas siguientes simplemente se calcula la suma aritmética y se imprime el resultado.

Vemos de nuevo que se accede a cada elemento del array. Si antes era para asignarles un valor, ahora es para obtenerlo.

Hay una forma simplificada de rellenar los elementos del array si conoces sus valores durante la declaración. Fíjate...

```
var a = [0, 1, 2, 3, 4, 5];
var suma = a[0] + a[1] + a[2] + a[3] + a[4] + a[5];
document.write(suma);
```

Ahora se utiliza una sintaxis más compacta, indicando, entre corchetes, los valores de cada uno de los elementos del array en el mismo momento de la declaración.

Es importante entender que los arrays son un componente del lenguaje JavaScript. Sin embargo, los vamos a utilizar principalmente cuando trabajemos con los objetos del navegador y del documento.

Así, veremos cómo algunas **propiedades** de los objetos que vamos a ir conociendo son realmente **arrays**.