JavaScript

Introducción

¿Qué es JavaScript?

JavaScript es un lenguaje de programación utilizado para crear pequeños programitas encargados de realizar acciones dentro del ámbito de una página web. Con JavaScript podemos crear efectos especiales en las páginas y definir interactividades con el usuario. El navegador del cliente es el encargado de interpretar las instrucciones JavaScript y ejecutarlas para realizar estos efectos e interactividades, de modo que el mayor recurso, y tal vez el único, con que cuenta este lenguaje es el propio navegador.

JavaScript es el siguiente paso, después del HTML, que puede dar un programador de la web que decida mejorar sus páginas y la potencia de sus proyectos. Es un lenguaje de programación bastante sencillo y pensado para hacer las cosas con rapidez, a veces con ligereza. Incluso las personas que no tengan una experiencia previa en la programación podrán aprender este lenguaje con facilidad y utilizarlo en toda su potencia con sólo un poco de práctica.

Entre las acciones típicas que se pueden realizar en JavaScript tenemos dos vertientes. Por un lado, los efectos especiales sobre páginas web, para crear contenidos dinámicos y elementos de la página que tengan movimiento, cambien de color o cualquier otro dinamismo. Por el otro, JavaScript nos permite ejecutar instrucciones como respuesta a las acciones del usuario, con lo que podemos crear páginas interactivas con programas como calculadoras, agendas, o tablas de cálculo.

Primeros Pasos

JavaScript se escribe en el documento HTML

Lo más importante y básico que podemos destacar en este momento es que la programación de JavaScript se realiza dentro del propio documento HTML. Es decir, el código JavaScript, en la mayoría de los casos, se mezcla con el propio código HTML para generar la página.

Esto quiere decir que debemos aprender a mezclar los dos lenguajes de programación y rápidamente veremos que, para que estos dos lenguajes puedan convivir sin problemas entre ellos, se han de incluir unos delimitadores que separan las etiquetas HTML de las instrucciones JavaScript. Estos delimitadores son las etiquetas <script> y </script>. Todo el código JavaScript que pongamos en la página ha de ser introducido entre estas dos etiquetas.

Maneras de ejecutar scripts de JavaScript

Existen **dos maneras fundamentales de ejecutar scripts** en la página. La primera de estas maneras se trata de ejecución directa de scripts, la segunda es una ejecución como respuesta a la acción de un usuario.

Ejecución directa

Es el método de ejecutar scripts más básico. En este caso se incluyen las instrucciones dentro de la etiqueta <SCRIPT>, tal como hemos comentado anteriormente. Cuando el navegador lee la página y encuentra un script va interpretando las líneas de código y las va ejecutando una después de otra. Llamamos a esta manera ejecución directa pues cuando se lee la página se ejecutan directamente los scripts.

Respuesta a un evento

Es la otra manera de ejecutar scripts, pero antes de verla debemos hablar sobre los eventos. Los eventos son acciones que realiza el usuario. Los programas como

Javascript están preparados para atrapar determinadas acciones realizadas, en este caso sobre la página, y realizar acciones como respuesta. De este modo se pueden realizar programas interactivos, ya que controlamos los movimientos del usuario y respondemos a ellos. Existen muchos tipos de eventos distintos, por ejemplo la pulsación de un botón, el movimiento del ratón o la selección de texto de la página.

Incluir ficheros externos de JavaScript

Otra manera de incluir scripts en páginas web, implementada a partir de Javascript 1.1, es incluir archivos externos donde se pueden colocar muchas funciones que se utilicen en la página. Los ficheros suelen tener extensión .js y se incluyen de esta manera.

```
<script type="text/javascript" src="archivo_externo.js"></script>
```

Sintaxis de JavaScript

El lenguaje Javascript tiene una sintaxis muy parecida a la de Java por estar basado en él. También es muy parecida a la del lenguaje C, de modo que si el lector conoce alguno de estos dos lenguajes se podrá manejar con facilidad con el código. De todos modos, en los siguientes capítulos vamos a describir toda la sintaxis con detenimiento, por lo que los novatos no tendrán ningún problema con ella.

Comentarios en JavaScript

Un comentario es una parte de código que no es interpretada por el navegador y cuya utilidad radica en facilitar la lectura al programador. El programador, a medida que desarrolla el script, va dejando frases o palabras sueltas, llamadas comentarios, que le ayudan a él o a cualquier otro a leer mas fácilmente el script a la hora de modificarlo o depurarlo.

Existen dos tipos de comentarios en el lenguaje. Uno de ellos, la doble barra, sirve para comentar una línea de código. El otro tipo es para comentar un bloque de líneas de código. Veamos unos ejemplos:

```
<script>

//Este es un comentario de una línea

/*Este comentario se puede extender por varias líneas.
Las que quieras*/

</script>
```

Mayúsculas y minúsculas

En Javascript se han de respetar las mayúsculas y las minúsculas. Si nos equivocamos al utilizarlas el navegador responderá con un mensaje de error, ya sea de sintaxis o de referencia indefinida.

Por poner un ejemplo, no es lo mismo la función alert() que la función Alert(). La primera muestra un texto en una caja de diálogo y la segunda (con la primera A mayúscula) simplemente no existe, a no ser que la definamos nosotros. Como se puede comprobar, para que la función la reconozca Javascript, se tiene que escribir toda en minúscula. Otro claro ejemplo lo veremos cuando tratemos con variables, puesto que los nombres que damos a las variables también son sensibles a las mayúsculas y minúsculas.

Variables en JavaScript

Una de las cosas más fundamentales en cualquier lenguaje de programación son las variables y los tipos de datos. Veremos qué son y cómo se trabaja con ellos en Javascript.

Concepto de variable

Una variable es un espacio en memoria donde se almacena un dato, un espacio donde podemos guardar cualquier tipo de información que necesitemos para realizar las acciones de nuestros programas.

Los nombres de las variables han de construirse con caracteres alfanuméricos y el carácter subrayado (_). Aparte de esta, hay una serie de reglas adicionales para construir nombres para variables. La más importante es que tienen que comenzar por un carácter alfabético o el subrayado.

También hay que evitar utilizar nombres reservados como variables, por ejemplo no podremos llamar a nuestra variable palabras como return, for, if, etc.

Declaración de variables

Declarar variables consiste en definir y de paso informar al sistema de que vas a utilizar una variable. Es una costumbre habitual en los lenguajes de programación el definir las variables que se van a usar en los programas y para ello, se siguen unas reglas estrictas. Pero Javascript se salta muchas reglas por ser un lenguaje un tanto libre a la hora de programar y uno de los casos en los que otorga un poco de libertad es a la hora de declarar las variables, ya que no estamos obligados a hacerlo, al contrario de lo que pasa en la mayoría de los lenguajes de programación.

Javascript cuenta con la palabra "var" que utilizaremos cuando queramos declarar una o varias variables. Como es lógico, se utiliza esa palabra para definir la variable antes de utilizarla.

```
var operando1;

var operando2;
```

También se puede asignar un valor a la variable cuando se está declarando:

```
var operando1 = 23;

var operando2 = 40;
```

También se permite declarar varias variables en la misma línea, siempre que se separen por comas.

```
var operando1, operando2;
```

Ámbito de variables en JavaScript

El ámbito de las variables en Javascript: qué son las variables locales y globales y cómo se trabaja con ellas en Javascript.

El ámbito de las variables es uno de los conceptos más importantes que deberemos conocer cuando trabajamos con variables, no sólo en Javascript, sino en la mayoría de los lenguajes de programación.

Concepto de ámbito de variables

Se le llama ámbito de las variables al lugar donde estas están disponibles. Por lo general, cuando declaramos una variable hacemos que esté disponible en el lugar donde se ha declarado, esto ocurre en todos los lenguajes de programación y como Javascript se define dentro de una página web, las variables que declaremos en la página estarán accesibles dentro de ella.

En Javascript no podremos acceder a variables que hayan sido definidas en otra página. Por tanto, la propia página donde se define es el ámbito más habitual de una variable y le llamaremos a este tipo de variables globales a la página. Veremos también se pueden hacer variables con ámbitos distintos del global, es decir, variables que declararemos y tendrán validez en lugares más acotados.

Variables globales

Como hemos dicho, las variables globales son las que están declaradas en el ámbito más amplio posible, que en Javascript es una página web.

Las variables globales son accesibles desde cualquier lugar de la página, es decir, desde el script donde se han declarado y todos los demás scripts de la página, incluidos los manejadores de eventos, como el onclick, que ya vimos que se podía incluir dentro de determinadas etiquetas HTML.

Variables locales

También podremos declarar variables en lugares más acotados, como por ejemplo una función. A estas variables les llamaremos locales. Cuando se declaren variables locales sólo podremos acceder a ellas dentro del lugar donde se ha declarado, es decir, si la habíamos declarado en una función solo podremos acceder a ella cuando estemos en esa función.

Las variables pueden ser locales a una función, pero también pueden ser locales a otros ámbitos, como por ejemplo un bucle. En general, son ámbitos locales cualquier lugar acotado por llaves.

```
<SCRIPT>

function miFuncion (){

    var variableLocal

}

</SCRIPT>
```

En el script anterior hemos declarado una variable dentro de una función, por lo que esa variable sólo tendrá validez dentro de la función. Se pueden ver cómo se utilizan las llaves para acotar el lugar donde está definida esa función o su ámbito.

No hay problema en declarar una variable local con el mismo nombre que una global, en este caso la variable global será visible desde toda la página, excepto en el ámbito donde está declarada la variable local ya que en este sitio ese nombre de variable está

ocupado por la local y es ella quien tiene validez. En resumen, la variable que tendrá validez en cualquier sitio de la página es la global. Menos en el ámbito donde está declarada la variable local, que será ella quien tenga validez.

```
<SCRIPT>

var numero = 2

function miFuncion () {

    var numero = 19

    document.write(numero) //imprime 19

}

document.write(numero) //imprime 2

</SCRIPT>
```

Un consejo para los principiantes podría ser no declarar variables con los mismos nombres, para que nunca haya lugar a confusión sobre qué variable es la que tiene validez en cada momento.

¿Qué podemos guardar en variables?

Vemos el concepto de tipos de datos para el lenguaje Javascript y por qué es importante manejarlos bien.

En una variable podemos introducir varios tipos de información. Por ejemplo podríamos introducir simple texto, números enteros o reales, etc. A estas distintas clases de información se les conoce como tipos de datos. Cada uno tiene características y usos distintos.

Veamos cuáles son los tipos de datos más habituales de Javascript.

Números

Para empezar tenemos el tipo numérico, para guardar números como 9 o 23.6.

Cadenas

El tipo cadena de carácter guarda un texto. Siempre que escribamos una cadena de caracteres debemos utilizar las comillas (").

Boleanos

También contamos con el tipo boleano, que guarda una información que puede valer si (true) o no (false).

Por último sería relevante señalar aquí que nuestras variables pueden contener cosas más complicadas, como podría ser un objeto, una función, o vacío (null).

En realidad nuestras variables no están forzadas a guardar un tipo de datos en concreto y por lo tanto no especificamos ningún tipo de datos para una variable cuando la estamos declarando. Podemos introducir cualquier información en una variable de cualquier tipo, incluso podemos ir cambiando el contenido de una variable de un tipo a otro sin ningún problema.

Esta ligereza a la hora de asignar tipos a las variables puede ser una ventaja en un principio, sobretodo para personas inexpertas, pero a la larga puede ser fuente de errores ya que dependiendo del tipo que son las variables se comportarán de un modo u otro y si no controlamos con exactitud el tipo de las variables podemos encontrarnos sumando un texto a un número. Javascript operará perfectamente, y devolverá un dato, pero en algunos casos puede que no sea lo que estábamos esperando. Así pues, aunque tenemos libertad con los tipos, esta misma libertad nos hace estar más atentos a posibles desajustes difíciles de detectar a lo largo de los programas. Veamos lo que ocurriría en caso de sumar letras y números.

```
var sumando1 = 23;
var sumando2 = "33";
var suma = sumando1 + sumando2;

¿Qué nos mostrará este código?
```

Operadores en JavaScript

Al desarrollar programas en cualquier lenguaje se utilizan los operadores, que sirven para hacer los cálculos y operaciones necesarios para llevar a cabo tus objetivos. Hasta el menor de los programas imaginables necesita de los operadores para realizar cosas, ya que un programa que no realizase operaciones, sólo se limitaría a hacer siempre lo mismo.

Es el resultado de las operaciones lo que hace que un programa varíe su comportamiento según los datos que tenga para trabajar y nos ofrezca resultados que sean relevantes para el usuario que lo utilice. Existen operaciones más sencillas o complejas, que se pueden realizar con operandos de distintos tipos, como números o textos, veremos en este capítulo, y los siguientes, de manera detallada todos estos operadores disponibles en Javascript.

Operadores aritméticos

Son los utilizados para la realización de operaciones matemáticas simples como la suma, resta o multiplicación. En javascript son los siguientes:

- Suma de dos valores.
- Resta de dos valores, también puede utilizarse para cambiar el signo de un número si lo utilizamos con un solo operando -23.
- Multiplicación de dos valores (*).
- División de dos valores (/).
- El resto de la división de dos números (%).
- Incremento en una unidad (++).
- Decremento en una unidad (--).

Operadores de asignación

Sirven para asignar valores a las variables, ya hemos utilizado en ejemplos anteriores el operador de asignación =, pero hay otros operadores de este tipo, que provienen del lenguaje C y que muchos de los lectores ya conocerán.

- Asignación (=).
- Asignación con suma (+=).
- Asignación con resta (-=).
- Asignación con multiplicación (*=).
- Asignación con división (/=).
- Asignación con resto (%=).

Operadores con cadenas

Las cadenas de caracteres tienen sus propios operadores para realizar acciones típicas sobre cadenas

- Concatenar dos cadenas (+).
- Asignación con concatenación (+=).

Como hemos podido ver, también en el caso del operador +=, si estamos tratando con cadenas de texto y números entremezclados, tratará a los dos operadores como si fuesen cadenas.

Operadores lógicos

Estos operadores sirven para realizar operaciones lógicas, que son aquellas que dan como resultado un verdadero o un falso, y se utilizan para tomar decisiones en nuestros scripts. En vez de trabajar con números, para realizar este tipo de operaciones se utilizan operandos boleanos, que conocimos anteriormente, que son el verdadero (true) y el falso (false). Los operadores lógicos relacionan los operandos boleanos para dar como resultado otro operando boleano.

- Operador NO (!).
- Operador Y (&&).
- Operador O (||).

Operadores condicionales

Los operadores condicionales se utilizan en las expresiones condicionales para tomas de decisiones. Como estas expresiones condicionales serán objeto de estudio más adelante será mejor describir los operadores condicionales más adelante.

- Comprobar si dos valores son iguales (==).
- Comprobar si dos valores son distintos (!=).
- Mayor (>).
- Menor (<).
- Mayor o igual (>=).
- Menor o igual (<=).

Veremos ejemplos de operadores condicionales cuando expliquemos estructuras de control, como la condicional **if**.

Estructuras de control en JavaScript

Las estructuras de control nos permitirán controlar el flujo de nuestros programas. Por supuesto, también forman parte de los asuntos más básicos de Javascript y de cualquier lenguaje de programación, por lo que las veremos con detenimiento.

Toma de decisiones

Nos sirven para realizar unas acciones u otras en función del estado de las variables. Es decir, tomar decisiones para ejecutar unas instrucciones u otras dependiendo de lo que esté ocurriendo en ese instante en nuestros programas.

Por ejemplo, dependiendo si el usuario que entra en nuestra página es mayor de edad o no lo es, podemos permitirle o no ver los contenidos de nuestra página.

En Javascript podemos tomar decisiones utilizando dos enunciados distintos:

- If / else
- Switch

Bucles

Los bucles se utilizan para realizar ciertas acciones repetidamente. Son muy utilizados a todos los niveles en la programación. Con un bucle podemos por ejemplo imprimir en una página los números del 1 al 100 sin necesidad de escribir cien veces el la instrucción imprimir. Las estructuras de control son las siguientes:

- For
- While
- Do While

Estructura If / Else en JavaScript

IF es una estructura de control utilizada para **tomar decisiones**. Es un condicional que sirve para realizar unas u otras operaciones en función de una expresión. Funciona de la siguiente manera, primero se evalúa una expresión, si da resultado positivo se realizan las acciones relacionadas con el caso positivo y en caso negativo se ejecutan las acciones contenidas en el bloque else.

La sintaxis es la siguiente:

```
if (expresión) {
//acciones a realizar en caso positivo //...

} else {

//acciones a realizar en caso negativo //...

}
```

Fijémonos en varias cosas. Para empezar vemos como con unas llaves engloban las acciones que queremos realizar en caso de que se cumplan o no las expresiones. Estas llaves han de colocarse siempre, excepto en el caso de que sólo haya una instrucción como acciones a realizar, que son opcionales.

Expresiones condicionales

La expresión a evaluar se coloca siempre entre paréntesis y está compuesta por variables que se combinan entre si mediante operadores condicionales. Recordamos que los operadores condicionales relacionaban dos variables y devolvían siempre un resultado boleano. Por ejemplo un operador condicional es el operador "es igual" (==), que devuelve true en caso de que los dos operandos sean iguales o false en caso de que sean distintos.

```
if (edad > 18) {
  //Hacemos lo que sea

}
```

Las expresiones condicionales se pueden mezclar con expresiones lógicas para crear expresiones más complejas. Recordamos que las expresiones lógicas tienen como operandos a los boleanos y devuelven otros boleanos.

```
if (edad > 18 && edad < 30) {
  //Hacemos lo que sea

}
```

Estructura Switch en JavaScript

La estructura de control switch de Javascript es utilizada para tomar decisiones en función de distintos estados o valores de una variable.

La sintaxis de la estructura switch es la siguiente:

```
switch (variable o expresion) {

  case valor1:

    //Acciones a realizar cuando el resultado es valor 1

    break;

  case valor2:

    //Acciones a realizar cuando el resultado es valor 2

    break;

  default:

     //Acciones a realizar cuando el resultado es valor 1

     break;

}
```

La expresión se evalúa, si vale valor1 se ejecutan las sentencias relacionadas con ese caso. Si la expresión vale valor2 se ejecutan las instrucciones relacionadas con ese valor y así sucesivamente, por tantas opciones como deseemos. Finalmente, para todos los casos no contemplados anteriormente se ejecuta el caso por defecto.

La palabra break es opcional, pero si no la ponemos a partir de que se encuentre coincidencia con un valor se ejecutarán todas las sentencias relacionadas con este y todas las siguientes. Es decir, si en nuestro esquema anterior no hubiese ningún break y la expresión valiese valor1, se ejecutarían las sentencias relacionadas con valor1 y también las relacionadas con valor2, valor3 y default.

También es opcional la opción default u opción por defecto.

Por útlimo se pueden ejecutar las mismas acciones para varios “case” distintos. Por ejemplo, si la variable que estamos evaluando es el valor numérico de un día de la semana (1,2,3,4,5,6,7), si para el fin de semana debemos ejecutar las mismas acciones se haría de la siguiente manera:

```
switch (dia_de_la_semana) {

case 1:

   document.write("Es Lunes")

   break

case 2:

   document.write("Es Martes")

   break

case 3:

   document.write("Es Miércoles")

   break

case 4:

   document.write("Es Jueves")

   break

case 5:

   document.write("Es viernes")

   break

case 6:

case 7:

   document.write("Es fin de semana")

   break
```

```
default:

   document.write("Ese día no existe")

}
```

Bucle for en JavaScript

El bucle FOR se utiliza para repetir una o más instrucciones un determinado número de veces. De entre todos los bucles, el FOR se suele utilizar cuando sabemos seguro el número de veces que queremos que se ejecute. La sintaxis del bucle for se muestra a continuación:

```
for (inicialización; condición; actualización) {

    //sentencias a ejecutar en cada iteración

}
```

El bucle FOR tiene tres partes incluidas entre los paréntesis, que nos sirven para definir cómo deseamos que se realicen las repeticiones. La primera parte es la inicialización, que se ejecuta solamente al comenzar la primera iteración del bucle. En esta parte se suele colocar la variable que utilizaremos para llevar la cuenta de las veces que se ejecuta el bucle.

La segunda parte es la condición, que se evaluará cada vez que comience una iteración del bucle. Contiene una expresión para decidir cuándo se ha de detener el bucle, o mejor dicho, la condición que se debe cumplir para que continúe la ejecución del bucle.

Por último tenemos la actualización, que sirve para indicar los cambios que queramos ejecutar en las variables cada vez que termina la iteración del bucle, antes de comprobar si se debe seguir ejecutando.

Después del for se colocan las sentencias que queremos que se ejecuten en cada iteración, acotadas entre llaves.

Un ejemplo de utilización de este bucle lo podemos ver a continuación, donde se imprimirán los números del 0 al 10.

```
var i;
for (i=0;i<=10;i++) {

   console.log(i);

}
```

En este caso se inicializa la variable i a 0. Como condición para realizar una iteración, se tiene que cumplir que la variable i sea menor o igual que 10. Como actualización se incrementará en 1 la variable i.

Como se puede comprobar, este bucle es muy potente, ya que en una sola línea podemos indicar muchas cosas distintas y muy variadas, lo que permite una rápida configuración del bucle y una versatilidad enorme.

Bucle while en JavaScript

Estos bucles se utilizan cuando queremos repetir la ejecución de unas sentencias un número indefinido de veces, siempre que se cumpla una condición. Es más sencillo de comprender que el bucle FOR, pues no incorpora en la misma línea la inicialización de las variables su condición para seguir ejecutándose y su actualización. Sólo se indica, como veremos a continuación, la condición que se tiene que cumplir para que se realice una iteración.

```
while (condición){

   //sentencias a ejecutar

}
```

Bucle do…while en JavaScript

El bucle do...while es la última de las estructuras para implementar repeticiones de las que dispone en Javascript y es una variación del bucle while visto anteriormente. Se utiliza generalmente cuando no sabemos cuantas veces se habrá de ejecutar el bucle, igual que el bucle WHILE, con la diferencia de que sabemos seguro que el bucle por lo menos se ejecutará una vez.

La sintaxis es la siguiente:

```
do{

  //sentencias a ejecutar

} while (condicion)
```

El bucle se ejecuta siempre una vez y al final se evalúa la condición para decir si se ejecuta otra vez el bucle o se termina su ejecución.

Así pues, existen dos instrucciones que se pueden usar en de las distintas estructuras de control y principalmente en los bucles, que te servirán para controlar dos tipos de situaciones. Son las instrucciones **break** y **continue**:

- **break:** Significa detener la ejecución de un bucle y salir de él.
- **continue:** Sirve para detener la iteración actual del bucle y volver al principio del bucle para comenzar una nueva iteración.

Break y continue

Son dos instrucciones que aumentan el control sobre los bucles en Javascript. Sirven para parar y continuar con la siguiente iteración del bucle respectivamente.

Imagina por ejemplo que estas haciendo un bucle muy largo para encontrar algo en cientos o miles de sitios. Pero ponte en el caso que durante las primeras iteraciones encuentres ese valor que buscabas. Entonces no tendría sentido continuar con el resto del bucle para buscar ese elemento, pues ya lo habías encontrado. En estas situaciones nos conviene saber para el bucle cancelar el resto de iteraciones. Obviamente, ésto es solo un ejemplo de cómo podríamos vernos en la necesidad de controlar un poco más el bucle. En la vida real como programador encontrarás muchas otras ocasiones en las que te interesará hacer esto u otras cosas con ellos.

Funciones en JavaScript

Ahora vamos a ver un tema muy importante, sobretodo para los que no han programado nunca y con Javascript están dando sus primeros pasos en el mundo de la programación ya que veremos un concepto nuevo, el de función, y los usos que tiene. Para los que ya conozcan el concepto de función también será un capítulo útil, pues también veremos la sintaxis y funcionamiento de las funciones en Javascript.

Qué es una función

A la hora de hacer un programa ligeramente grande existen determinados procesos que se pueden concebir de forma independiente, y que son más sencillos de resolver que el problema entero. Además, estos suelen ser realizados repetidas veces a lo largo de la ejecución del programa. Estos procesos se pueden agrupar en una función, definida para que no tengamos que repetir una y otra vez ese código en nuestros scripts, sino que simplemente llamamos a la función y ella se encarga de hacer todo lo que debe.

Así que podemos ver una función como una serie de instrucciones que englobamos dentro de un mismo proceso. Este proceso se podrá luego ejecutar desde cualquier otro sitio con solo llamarlo. Por ejemplo, en una página web puede haber una función para cambiar el color del fondo y desde cualquier punto de la página podríamos llamarla para que nos cambie el color cuando lo deseemos.

Las funciones se utilizan constantemente, no sólo las que escribes tú, sino también las que ya están definidas en el sistema, pues todos los lenguajes de programación suelen tener un montón de funciones para realizar procesos habituales, como por ejemplo obtener la hora, imprimir un mensaje en la pantalla o convertir variables de un tipo a otro. Ya hemos visto alguna función en nuestros sencillos ejemplos anteriores. Por ejemplo, cuando hacíamos un document.write() en realidad estábamos llamando a la función write().

Cómo se escribe una función

Una función se debe definir con una sintaxis especial que vamos a conocer a continuación.

```
function nombre_funcion(){

  //instrucciones de la función

}
```

Primero se escribe la palabra **function**, reservada para este uso. Seguidamente se escribe el nombre de la función, que como los nombres de variables puede tener números, letras y algún carácter adicional como en guión bajo. A continuación se colocan entre llaves las distintas instrucciones de la función. Las llaves en el caso de las funciones no son opcionales, además es útil colocarlas siempre como se ve en el ejemplo, para que se reconozca fácilmente la estructura de instrucciones que engloba la función.

Dónde colocaremos las funciones en JavaScript

En principio, podemos colocar las funciones en cualquier parte de la página, siempre entre etiquetas <script>, claro está. No obstante existe una limitación a la hora de colocarla con relación a los lugares desde donde se la llame. Te adelantamos que lo más fácil es colocar la función antes de cualquier llamada a la misma y así seguro que nunca nos equivocaremos.

Existen dos opciones posibles para colocar el código de una función:

a) Colocar la función en el mismo bloque de script: En concreto, la función se puede definir en el bloque <script> donde esté la llamada a la función, aunque es indiferente si la llamada se encuentra antes o después del código de la función, dentro del mismo bloque <script>.

```
<SCRIPT>
miFuncion()
function miFuncion(){

//hago algo... document.write("Esto va bien")

} </SCRIPT>
```

Este ejemplo funciona correctamente porque la función está declarada en el mismo bloque que su llamada.

b) Colocar la función en otro bloque de script: También es válido que la función se encuentre en un bloque <script> anterior al bloque donde está la llamada.

```
<html>

<head>

<title>MI PÁGINA</title>

<script>

function miFuncion(){

   //hago algo...

}
```

```
</script>

</head>

<body>

<script>

   miFuncion()

</script>

</body>

</html>
```

Vemos un código completo sobre cómo podría ser una página web donde tenemos funciones Javascript. Como se puede comprobar, las funciones están en la cabecera de la página (dentro del head). Éste es un lugar excelente donde colocarlas, porque se supone que en la cabecera no se van a utilizar todavía y siempre podremos disfrutar de ellas en el cuerpo porque sabemos seguro que ya han sido declaradas.

Para que quede claro este asunto de la colocación de funciones veamos el siguiente ejemplo, que daría un error. Examina atentamente el código siguiente, que lanzará un error, debido a que hacemos una llamada a una función que se encuentra declarada en un bloque <script> posterior.

Parámetros de las funciones

Las funciones también tienen una entrada y una salida de datos. En este artículo veremos cómo podemos enviar datos a las funciones Javascript.

Parámetros

Los parámetros se usan para mandar valores a las funciones. Una función trabajará con los parámetros para realizar las acciones. Por decirlo de otra manera, los parámetros son los valores de entrada que recibe una función.

Por poner un ejemplo sencillo de entender, una función que realizase una suma de dos números tendría como parámetros a esos dos números. Los dos números son la entrada, así como la salida sería el resultado de la suma, pero eso lo veremos más tarde.

Los parámetros pueden recibir cualquier tipo de datos, numérico, textual, boleano o un objeto. Realmente no especificamos el tipo del parámetro, por eso debemos tener un cuidado especial al definir las acciones que realizamos dentro de la función y al pasarle valores, para asegurarnos que todo es consecuente con los tipos de datos que esperamos tengan nuestras variables o parámetros.

Múltiples parámetros

Una función puede recibir tantos parámetros como queramos y para expresarlo se colocan los nombres de los parámetros separados por comas, dentro de los paréntesis. Veamos rápidamente la sintaxis para que la función de antes, pero hecha para que reciba dos parámetros, el primero el nombre al que saludar y el color del texto:

```
function escribirBienvenida (nombre, colorTexto){

  document.write("<font color='" + colorTexto + "'>");

  document.write("<h1>Hola " + nombre + "</h1>");

  document.write("</font>");

}
```

Llamaríamos a la función con esta sintaxis. Entre los paréntesis colocaremos los valores de los parámetros:

```
var miNombre = "Pepe";

var miColor = "red";
```

Valores de retorno

Las funciones pueden devolver valores, a través de la sentencia return. También vemos un apunte sobre el ámbito de variables en funciones en Javascript.

Devolución de valores de retorno

Las funciones en Javascript también pueden retornar valores. De hecho, ésta es una de las utilidades más esenciales de las funciones, que debemos conocer, no sólo en Javascript sino en general en cualquier lenguaje de programación. De modo que, al invocar una función, se podrá realizar acciones y ofrecer un valor como salida.

Veamos un ejemplo de función que calcula la media de dos números. La función recibirá los dos números y retornará el valor de la media.

```
function media (numero1, numero2){

   var resultado = (numero1 + numero2) / 2;

   return resultado;

}
```

Para especificar el valor que retornará la función se utiliza la palabra **return** seguida de el valor que se desea devolver. En este caso se devuelve el contenido de la variable resultado, que contiene la media calculada de los dos números.

Quizás nos preguntemos ahora cómo recibir un dato que devuelve una función. Realmente en el código fuente de nuestros programas podemos invocar a las funciones en el lugar que deseemos. Cuando una función devuelve un valor simplemente se sustituye la llamada a la función por ese valor que devuelve. Así pues, para almacenar el valor que nos devuelve la función, tenemos que asignar la llamada a esa función como contenido de una variable:

```
var miMedia;

miMedia = media(3,5);

console.log(miMedia); //Devolverá el valor almacenado en miMedia = 4;
```

Como en todos los lenguajes de programación, existen funciones predefinidas (librería de funciones de JavaScript) que podemos utilizar.

Arrays JavaScript

Vemos que son los arrays en Javascript, para qué sirven y cómo utilizarlos. Veremos diversas formas de crearlos, así como definir y acceder a sus valores.

En los lenguajes de programación existen estructuras de datos especiales que nos sirven para guardar información más compleja que simples variables. Una estructura típica en todos los lenguajes es el Array, que es como una variable donde podemos introducir varios valores, en lugar de solamente uno como ocurre con la variables normales.

Los arrays nos permiten guardar varias variables y acceder a ellas de manera independiente, es como tener una variable con distintos compartimentos donde podemos introducir datos distintos. Para ello utilizamos un índice que nos permite especificar el compartimiento o posición a la que nos estamos refiriendo.

Creación de arrays JavaScript

El primer paso para utilizar un array es crearlo. Para ello utilizamos un objeto Javascript ya implementado en el navegador. Veremos en adelante un tema para explicar lo que es la orientación a objetos, aunque no será necesario para poder entender el uso de los arrays. Esta es la sentencia para crear un objeto array:

```
var miArray = new Array();
```

También podemos crear un array especificando el número de compartimentos que tendrá:

```
var miArray = new Array(10);
```

Es importante que nos fijemos que la palabra Array en código Javascript se escribe con la primera letra en mayúscula. Como en Javascript las mayúsculas y minúsculas si que importan, si lo escribimos en minúscula no funcionará.

Tanto se indique o no el número de casillas del **array javascript**, podemos introducir en el array cualquier dato. Si la casilla está creada se introduce simplemente y si la casilla no estaba creada se crea y luego se introduce el dato, con lo que el resultado final es el mismo. Esta creación de casillas es dinámica y se produce al mismo tiempo que los scripts se ejecutan. Veamos a continuación cómo introducir valores en nuestros arrays.

```
miArray[0] = 10;

miArray[1] = 20;

miArray[2] = 30;
```

Se introducen indicando entre corchetes el índice de la posición donde queríamos guardar el dato. En este caso introducimos 290 en la posición 0, 97 en la posición 1 y 127 en la 2.

Los arrays en Javascript empiezan siempre en la posición 0, así que un array que tenga por ejemplo 10 posiciones, tendrá casillas de la 0 a la 9. Para recoger datos de un array lo hacemos igual: poniendo entre corchetes el índice de la posición a la que queremos acceder. Veamos cómo se imprimiría en la pantalla el contenido de un array.

Declaración e iniciación de arrays en JavaScript

En Javascript tenemos a nuestra disposición una manera resumida de declarar un array y cargar valores en un mismo paso. Fijémonos en el código siguiente:

```
Var arrayRapido = [12, 30, “array creado e inicializado”];
```

Como se puede ver, se está definiendo una variable llamada arrayRapido y estamos indicando en los corchetes varios valores separados por comas. Esto es lo mismo que

haber declarado el array con la función Array() y luego haberle cargado los valores uno a uno.

Longitud de los arrays

Todos los arrays en JavaScript, a parte de almacenar los valores en cada una de sus casillas, también almacenan el número de posiciones que tienen. Para ello utilizan una propiedad del objeto array, la propiedad **length**. Ya veremos en objetos qué es una propiedad, pero para nuestro caso podemos imaginarnos que es como una variable, adicional a las posiciones, que almacena un número igual al número de casillas que tiene el array.

Para acceder a una propiedad de un objeto se ha de utilizar el operador punto. Se escribe el nombre del array que queremos acceder al número de posiciones que tiene, sin corchetes ni paréntesis, seguido de un punto y la palabra **length**.

```
var miArray = new Array();

miArray[0] = 10;

miArray[1] = 20;

miArray[2] = 30;

console.log(miArray.length); //Mostrará en consola 3 (el numero de casillas que tiene)
```

Es muy habitual que se utilice la propiedad **length** para poder recorrer un array por todas sus posiciones. Para ilustrarlo vamos a ver un ejemplo de recorrido por este array para mostrar sus valores.

```
var miArray = [1,2,3];

for(var i=0; i < miArray.length; i++){

   console.log(miArray[i]);

}
```

JavaScript HTML DOM

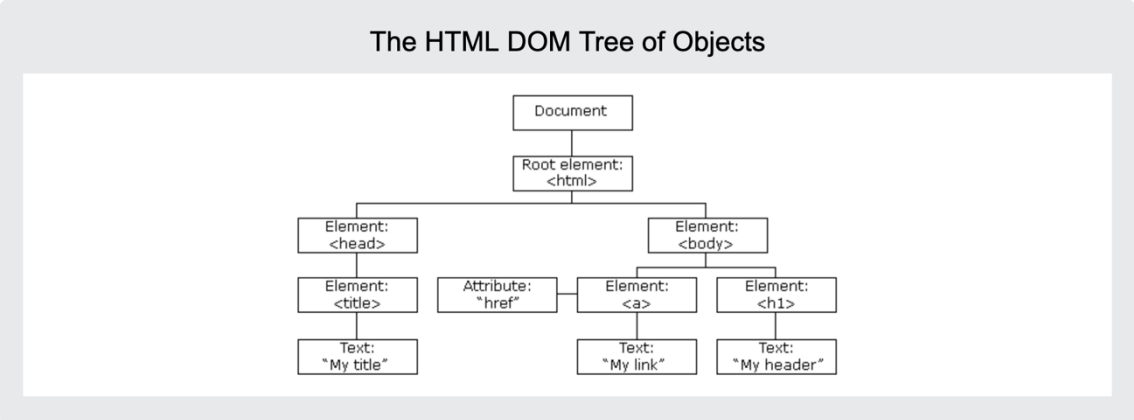
¿Qué es el DOM de HTML?

El HTML DOM es un modelo de **objeto** estándar y una **interfaz de programación** para HTML. Se define:

- Los elementos HTML como **objetos**
- Las **propiedades** de todos los elementos HTML
- Los **métodos** para acceder a todos los elementos HTML
- Los **eventos** para todos los elementos HTML

En otras palabras: **el HTML DOM es un estándar sobre cómo obtener, cambiar, agregar o eliminar elementos HTML.**

El modelo **HTML DOM** se construye como un árbol de **objetos**:



Con el modelo de objetos, JavaScript obtiene todo el poder que necesita para crear HTML dinámico:

- JavaScript puede cambiar todos los elementos HTML en la página
- JavaScript puede cambiar todos los atributos HTML en la página
- JavaScript puede cambiar todos los estilos CSS en la página

- JavaScript puede eliminar elementos y atributos HTML existentes
- JavaScript puede agregar nuevos elementos y atributos HTML
- JavaScript puede reaccionar a todos los eventos HTML existentes en la página
- JavaScript puede crear nuevos eventos HTML en la página

Métodos del DOM HTML

Los métodos HTML DOM son **acciones** que puede realizar (en elementos HTML).

Las propiedades HTML DOM son **valores** (de elementos HTML) que puede establecer o cambiar.

El método getElementById

Es la forma más común de acceder a un elemento HTML del DOM. Como su propio nombre lo indica, la única restricción para obtenerlo es que tenga asignada una **id.** Si tenemos el siguiente elemento HTML en nuestro código ** ,** con JavaScript podremos acceder a el de la siguiente manera:

var elemento = document.getElementById("contenendor");

La propiedad innerHTML

La forma más sencilla de obtener el contenido de un elemento es mediante la **propiedad innerHTML**.

La **propiedad innerHTML** es útil para obtener o reemplazar el contenido de los elementos HTML.

El objeto documento DOM HTML

El objeto del documento representa su página web.

Si desea acceder a cualquier elemento en una página HTML, siempre comienza accediendo al objeto del documento.

A continuación, se muestran algunos ejemplos de cómo puede utilizar el objeto de documento para acceder y manipular HTML.

Encontrar elementos en el DOM

Método	Descripción
document.getElementById(id)	Encuentra un elemento por su id
document.getElementByTagName(name)	Encuentra uno o varios elementos por el nombre de etiqueta
document.getElementByClassName(name)	Encuentra uno o varios elementos por el nombre de clase

Cambio de elementos HTML

Propiedad	Descripción
element.innerHTML = “contenido HTML”	Cambia el HTML interior de un elemento
element.attribute = “nuevo valor de atributo”	Cambia el valor de un atributo de un elemento
element.style.*property* = “nuevo valor de estilo”	Cambia el estilo de un elemento
Método	**Descripción**
element.setAttribute(“attribute”, “valor”)	Cambia el valor de un atributo de un elemento

Adición y eliminación de elementos

Método	Descripción
document.createElement(element)	Crea un elemento HTML
document.removeChild(element)	Elimina un elemento HTML
document.appendChild(element)	Añade un elemento HTML
document.replaceChild(old,new)	Reemplaza un elemento HTML
document.write(text)	Escribe en el documento HTML

Elementos en el DOM

En HTML DOM, el **objeto Element** representa un elemento HTML, como P, DIV, A, TABLE o cualquier otro elemento HTML.

En este curso solo veremos los 3 métodos básicos para obtener elementos del DOM:

- **getElementById():** Devuelve un elemento del DOM a partir de su id.

- **getElementByClassName():** Devuelve uno o varios elementos del DOM a partir de su nombre de clase. Hay que tener en cuenta que, normalmente, tendremos varios elementos con la misma clase, y por tanto, si queremos realizar alguna acción sobre ellos habrá que iterarlos. Hay varios métodos de hacer esto pero en este curso únicamente lo iteraremos con el bucle **for** que hemos estudiado.

- **getElementByTagName():** Devuelve uno o varios elementos del DOM a partir de su nombre de etiqueta (tag). Igual que con el método anterior, si tenemos varios elementos bajo una etiqueta lo que obtendremos será un array de elementos y por tanto también habrá que iterarlos para realizar acciones sobre cada uno de ellos.

Podéis consultar todos las propiedades y métodos que se pueden realizar sobre elementos en el DOM en el siguiente enlace:

https://www.w3schools.com/jsref/dom_obj_all.asp

Style en el DOM

El objeto **style** del DOM lo utilizamos para realizar cambios de estilo CSS sobre elementos del DOM utilizando JavaScript. El proceso es simple. Debemos obtener el elemento al que queremos aplicar o modificar el estilo CSS y a continuación asignarle el nuevo estilo de la siguiente manera:

elemento.style.propiedad = “nuevo valor”;

Ejemplo:

var elemento = document.getElementById(“id_del_elemento”);

elemento.style.color = “red”; //Aplicamos al elemento el color de fuente rojo

elemento.style.backgroundColor = “blue”; //Aplicamos al elemento el color de fondo azul

element.style.display = “none”; //Ocultamos el elemento con display none

Podéis consultar todos los estilos actualizables desde JavaScript en el siguiente enlace:

https://www.w3schools.com/jsref/dom_obj_style.asp

Eventos HTML DOM

Los eventos DOM permiten que JavaScript agregue detectores de eventos o controladores de eventos a elementos HTML.

La manera más habitual de utilizar los eventos en JavaScript es aplicárselo a un botón, de manera que cuando el usuario lo pulse se realicen ciertas instrucciones HTML.

En el caso de un botón tenemos varias maneras de aplicarles un *listener* (escuchador) a un botón:

<button onclick=”myFunction()”>Púlsame</button>

O bien, si temenos un botón en el HTML cuya id sea “boton”:

var boton = document.getElementById(“boton”);

botón.addEventListener(“click”, myFunction);

En el primer ejemplo, cuando el botón sea pulsado por el usuario, se ejecutará la función *myFunction().*

En el segundo ejemplo, primero obtenemos el elemento botón del DOM y una vez tenemos el botón le aplicamos el *listener* click (le estamos diciendo que estamos capturando el click del botón) y le asignamos la función *myFunction* (es decir, cuando haga click en el botón ejecutará las instrucciones contenidas en dentro de la función).

Podéis consultar la lista completa de eventos en la siguiente url:

https://www.w3schools.com/jsref/dom_obj_event.asp