

# JAVASCRIPT.

* 1. **GENERALIDADES.**

Javascript es un lenguaje interpretado, es decir, un lenguaje que no necesita compilar los programas para poderlos ejecutar, sino que el propio navegador se encarga de interpretarlos y ejecutarlos sin necesidad de pasos intermedios, que permite añadir mayores efectos dinámicos a los documentos html como, por ejemplo, botones que, al ser pulsados, realizan cambios en los elementos de la página web.

Dentro del Diseño Web, los lenguajes de scripting como Javascript van tomando una relevancia muy importante al

utilizar una “programación del lado cliente”, es decir, que los elementos de esos lenguajes se tratan y manejan por los navegadores locales que, a su vez, consumen también recursos locales sin tener que depender de servidores remotos a no ser, por ejemplo, para cuestiones de seguridad como puede ser los procesos de identificación de usuarios.

Este auge de “la programación

del lado cliente” ha hecho que el desarrollo de Javascript haya sido enfocado hacia una mayor portabilidad e integración con los modernos navegadores que también han aportado su granito de arena al aumentar su rendimiento e incorporar interfaces de programación de aplicaciones (APIs) como Web Storage, Canvas, etc., que asisten al lenguaje en funciones elementales con la idea de proporcionar poderosas funciones a través de técnicas de programación sencillas y estándares que facilitan la creación de programas útiles para el Diseño Web.



*Podemos ver HTML5 como la mezcla de tres elementos básicos:* ***HTML****, lenguaje para el contenido,* ***CSS****, herramienta para el aspecto, y* ***Javascript****, lenguaje para el dinamismo de los elementos.*

# ELEMENTOS DE DEFINICIÓN DE SCRIPTS

#### <script>

Permite la incorporación de programas de scripting de “lado cliente” como Javascript

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Etiqueta | <script> | | | | | | |
| Descripción | Permite la incorporación programas de scripting | | | | | | |
| Elemento | Ni en bloque ni en línea | | | | | | |
| Navegadores que la  soportan |  |  |  |  |  |  |  |
| Chrome | Firefox | IExplorer | Ópera | Safari |  |
| Atributos | Globales | | | | | | |
| Atributos propios | async, charset, defer, src, type | | | | | | |
| Diferencia entre html  4.01 y html5 | async es un nuevo atributo HTML5, xml:space ha sido suprimido y type es obligatorio en HTML4 pero opcional en HTML5 | | | | | | |

|  |  |  |
| --- | --- | --- |
| Atributos propios | Valor | Descripción |
| async | Sin valor | Permite cargar el contenido de la página html sin esperar a que acabe la ejecución del script. Solo vale para scripts externos |
| charset | *Mapa de caracteres* | Mapa de caracteres empleado en un script externo. |
| defer | Sin valor | El script se ejecuta cuando se acabe de cargar la página |
| src | URL | Indica la dirección URL del script |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| type | *Tipo MIME* | Especifica el tipo MIME del script. Sus posibles valores son:   * text/javascript (defecto) * text/ecmascript * application/ecmascript * application/javascript |

  Nuevo en HTML5

Entre las etiquetas de apertura y cierre del elemento <script> van las instrucciones del programa y este elemento puede estar incluido dentro de <head> dentro de <body> dependiendo de cómo queremos la ejecución del mismo.

#### <noscript>

Define un contenido alternativo para cuando el navegador tenga deshabilitada la ejecución de scripts o cuando dicho navegador no los soporte.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Etiqueta | <noscript> | | | | | | |
| Descripción | Define un contenido alternativo cuando no se puede ejecutar un script | | | | | | |
| Elemento | Ni en bloque ni en línea | | | | | | |
| Navegadores que la  soportan |  |  |  |  |  |  |  |
| Chrome | Firefox | IExplorer | Ópera | Safari |  |
| Atributos | Globales | | | | | | |
| Atributos propios | No tiene | | | | | | |
| Diferencia entre html  4.01 y html5 | En HTML4 <noscript> solo puede usarse dentro de <body> y en HTML5 puede ir tanto en <head> como en <body> | | | | | | |

#### Ejemplo:

<!DOCTYPE html>

<html lang="es-es">

<head>

<title>Script</title>

<meta charset="utf-8">

</head>

<body>

<script type=”text/javascript”> document.write("Mensaje");

</script>

<noscript>

O bien tu navegador no soporta scripts o los tiene deshabilitados

</noscript>

</body>

</html>

En este ejemplo se ve que el elemento <script> del tipo “text/javascript”, o sea, javascript, está definido en <body> y se ejecutará al cargar la página siempre y cuando el navegador lo permita. Este pequeño programa contiene una única instrucción (document.write(“Mensaje”;) acabada en “;” que visualiza el mensaje “Mensaje” por pantalla cuando se ejecuta.

En el caso de no poder ejecutar el script visualizará el mensaje “O bien tu navegador no soporta scripts o los tiene deshabilitados”.

# INCORPORACIÓN DE SCRIPTS A UN DOCUMENTO HTML

Los scripts se pueden incorporar de la misma forma que las hojas de estilo CSS, es decir, en línea, incorporados al propio documento o en archivos externos.

#### En línea

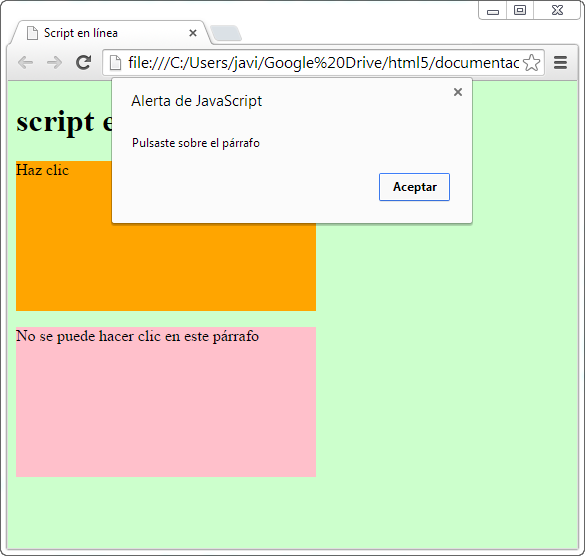
Se incorpora el código dentro de la etiqueta de apertura de un elemento a través de un manejador de eventos como puede ser onclick (pulsar con el ratón sobre él), onMouseOver (que pase el puntero del ratón sobre él), onMouseOut (cuando salga el puntero del ratón de él) u otros más que se producen cuando se pulsa una tecla o se carga una página.

#### Ejemplo:

|  |
| --- |
| <!DOCTYPE html>  <html lang="es-es">  <head>  <title>Script en línea</title>  <meta charset="utf-8">  <style type="text/css">  .fondoverde  {background-color:#ccffcc;}  #uno {background-color:orange;}  #dos  { background-color:pink;}  p  {width: 300px; height: 150px;}  </style>  </head>  <body class="fondoverde"> |

|  |
| --- |
| <h1> script en línea </h1>  <p id="uno" onclick="alert('Púlsaste sobre el párrafo')">Haz clic </p>  <p id="dos"> No se puede hacer clic en este párrafo</p`>  </body>  </html> |
|  |

|  |
| --- |
|  |

En este ejemplo en el elemento <p> con id=”uno” y texto inicial “”Haz clic” tiene definido que, cuando se haga clic sobre él con el ratón, saque una pequeña ventana con el texto “Pulsaste sobre el párrafo” (onclick=”alert(‘Pulsaste sobre el párrafo’)”)

#### Incrustado en <head> o <body>

Es cuando el código del script se inserta entre las etiquetas de apertura y cierre del elemento <script> que puede estar en cualquier parte del documento html aunque es recomendable ponerlo dentro <head> por aislar lo que es html de lo que es javascript.

#### Ejemplo:

|  |
| --- |
| <!DOCTYPE html>  <html lang="es-es">  <head>  <title>Script en línea</title>  <meta charset="utf-8">  <style type="text/css">  .fondoverde  {background-color:#ccffcc;}  </style>  <script type="text/javascript"> alert("Hola Mundo");  </script>  </head>  <body class="fondoverde">  <h1> Script incrustado</h1>  </body>  </html> |

|  |
| --- |
|  |
|  |

En este ejemplo se define un script dentro <head> que abre una ventana de alerta de javascript con el mensaje “Hola Mundo”. Lo primero que haría el navegador es ejecutar el script y después de pulsar el botón de aceptar de la ventana de alerta se ejecutaría lo que hay en <body>.



*Si pones código javascript dentro de <head> procura que sean funciones*

*de la página web*

*ejemplo anterior, se ejecutará antes el código javascript que el contenido*

*que se ejecuten cuando sucede algún evento ya que, si haces como en el*

#### En un archivo externo

Al igual que se hace con las hojas CSS externas, se puede guardar todo el código javascript en archivos de texto con extensión normalmente “.js” independientes de los documentos html lo que facilita su mantenimiento y su utilización en otras páginas web.



#### Ejemplo:

*Podemos insertar tantos archivos externos javascript como queramos en nuestras páginas.*

|  |
| --- |
| <!DOCTYPE html>  <html lang="es-es">  <head>  <title>Script en línea</title>  <meta charset="utf-8">  <style type="text/css">  .fondoverde  {background-color:#ccffcc;}  </style>  <script type="text/javascript" src=”archivo.js”>  </script>  </head>  <body class="fondoverde">  <h1> Script incrustado</h1>  </body> |

|  |
| --- |
| </html> |
| Contenido del archivo “archivo.js”  alert("Hola Mundo"); |

Este ejemplo realiza lo mismo que el anterior con la salvedad de que el código javascript está en el fichero “archivo.js” que se encuentra en la misma carpeta que la página html que lo utiliza.



*Si usas archivos externos puedes usar el parámetro* ***async*** *del elemento*

*<script> lo que permite que se pueda cargar el contenido de la página sin tener que esperar a que acabe de ejecutarse el script*

# PROGRAMACIÓN INICIAL

En este bloque vamos a ver los elementos y estructuras básicas de un programa javascript. Su sintaxis y lógica es muy parecida a otros lenguajes de programación como son Java o C pero existen sutiles diferencias que veremos a continuación.

#### Reglas sintácticas básicas

La sintaxis de un lenguaje de programación es el conjunto de normas que indican como se debe escribir un programa para que sea utilizado de forma correcta.

Las reglas básicas son las siguientes:

* + - * No se indica el tipo de variable cuando se crea por lo que puede albergar diferentes tipos de datos durante la ejecución del script.
      * Se distingue entre mayúsculas y minúsculas, es decir, si escribimos, por ejemplo, aLert(“Mensaje”) no se ejecutará nunca.
      * Aunque la sintaxis de un script javascript es muy parecida a la de programas escritos en C o Java, una de las

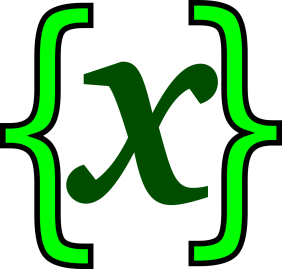
diferencias es que no es necesario acabar las instrucciones en “;” en javascript,

* + - * Se ignoran los espacios en blanco repetidos y las líneas en blanco entre instrucciones.
      * Se pueden incluir comentarios poniendo **//** delante de una línea de texto o poniendo **/\*** Texto del comentario **\*/**
      * Al igual que cualquier lenguaje de programación, javascript contiene un conjunto de “palabras reservadas” que tienen un significado específico y se emplean para formar estructuras sintácticas concretas.

#### Variables

Una variable es un elemento que permite guardar datos de forma temporal y utilizar su valor para la realización de ciertas operaciones.

Existen variables de diferentes tipos dependiendo del valor que se les asignen como numéricas, cadenas de texto, boolean, etc. También se pueden definir colecciones de variables del mismo tipo dentro de *arrays* de una o varias dimensiones.

La palabra reservada “var” sirve para declarar una variable la primera vez que aparece en el programa y luego ya se referencia la variable por el nombre de la misma.

No es necesario inicializar una variable a la hora de declararla.

**Ejemplos de variables simples:**

|  |
| --- |
| **var** variable = ”Esta es una cadena de texto”;  **var** num1 = 4; **var** num2 = 3; **var** num3;  **var** suma = num1 + num2; /\* num1 y num2 están declarados más arriba \*/  **var** logico = false; // logico es una variable boolean |

**Ejemplos de arrays:**

|  |
| --- |
| **var** impares = [1,3,5,7];  /\* El array “impares” tiene 4 valores comenzando por el elemento impares[0], que vale 1 y acabando en impares [3], que vale 7 \*/  **var** impares[4]=9**;**  **/\*** Al array “impares” se le agrega un quinto valor \*/  **var** suma=impares[0]+impares[2];  /\* A la variable “suma” se le asigna el valor 6 (1+5)\*/  **var** pares[ ];  /\* Se define el array “pares” para luego asignarle valores \*/  **var** tabla[ ][ ];  /\* Se define el array “tabla” para luego asignarle valores \*/  tabla[1][3]=”Elemento 1,3 de la tabla”;  /\*Apoyándonos en el ejemplo anterior asignamos al elemento 1,3 de la tabla la cadena de texto “Elemento 1,3 de la tabla” /\* |

#### Operadores

Permiten manejar los valores de las variables y realizar cálculos tanto lógicos como aritméticos con ellos.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipo** | **Símbolo** | **Descripción** | **Ejemplo** |
| Asignación | = | Carga un valor en una variable | num=3;  **var** num1=”hola”; |
| Incremento (\*) | ++ | Incrementa en una unidad una variable | **var** num=3;  ++num; //num = 4 |
| Decremento (\*) | -- | Decrementa en una unidad una variable | **var** num=3;  --num; //num = 2 |
| Negación | ! | Devuelve el valor opuesto de la variable | **var** variable=true; alert(¡variable);  //devuelve false |
| AND | && | Devuelve true si los dos valores son true o, en cualquier otro caso, falso | **var** varia1=true; **var** varia2=false; **var** a;  a=varia1 && varia2;  //a vale false |
| OR | || | Devuelve false si los dos valores son falsos o, en cualquier otro caso, verdadero | **var** varia1=true; **var** varia2=false; **var** a;  a=varia1 || varia2;  //a vale true |
| Matemáticos básicos | + - \* / | Suman, restan, multiplican o dividen variables | **var** vari1=3  **var** vari2=2  **var** a=vari1- vari2;  //a vale 1 |
| Módulo | % | Calcula el resto de | **var** vari1=7; |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | una división entera | **var** vari2=2;  **var** a=vari1% vari2;  //a vale 1 |
| Mayor que | > | Devuelve true si la variable de la izquierda es mayor que la de derecha | **var** vari1=3  **var** vari2=2  **var** a=vari1>vari2;  //a vale true |
| Menor que | < | Devuelve true si la variable de la derecha es mayor que la de izquierda | **var** vari1=3  **var** vari2=2  **var** a=vari1<vari2;  //a vale false |
| Mayor o igual que | >= | Devuelve true si la variable de la izquierda es mayor o igual que la de derecha | **var** vari1=3  **var** vari2=3  **var** a=vari1>=vari2;  //a vale true |
| Menor o igual que | <= | Devuelve true si la variable de la derecha es mayor o igual que la de izquerda | **var** vari1=3  **var** vari2=4  **var** a=vari1=>vari2;  //a vale false |
| Igual que | == | Devuelve true si ambas variables son iguales | **var** vari1=3  **var** vari2=4  **var** a=vari1==vari2;  //a vale false |
| Distinto de | != | Devuelve true si ambas variables son distintas | **var** vari1=3  **var** vari2=4  **var** a=vari1!=vari2;  //a vale true |

(\*) Estos dos operadores pueden ir delante o detrás de la variable. En el caso de ir delante el incremento/decremento se hace antes de operar y en el caso de ir detrás se opera primero y se incrementa después.

#### Estructuras de control

Permiten controlar e, incluso, alterar el flujo de ejecución de las instrucciones de cualquier programa.

#### if

Si se cumple la condición definida en su construcción ejecuta las instrucciones definidas en su bloque

|  |  |  |
| --- | --- | --- |
| **Formato** | **Ejemplo** | **Explicación** |
| **if** (condición){  …..  } | var num1=1; var num2=2;  If (num1<num2)  { var num3=num1+num2; alert(num3);  } | Se visualiza una ventana con el valor de num3 (suma de num1 y num2) ya que la condición se cumple |

#### if .. else

Si se cumple la condición definida en su construcción ejecuta el primer bloque de instrucciones y si no se cumple ejecuta el bloque de instrucciones definido detrás de la palabra reservada “else”.

|  |  |  |
| --- | --- | --- |
| **Formato** | **Ejemplo** | **Explicación** |
| **if** (condición){  …..  }  **else**  {  ….  } | var num1=1; var num2=2;  If (num1<num2)  { mayor=”no es”;} else  { mayor=”es”;}  alert (“num1 “ + mayor + “ mayor que num2”); | Si num2 es mayor que num1 asigna el valor “no es” a mayor y si no se cumple lo anterior asigna el valor “es” a mayor |
|  |  |  |

#### for

Estructura repetitiva que se ejecuta un número determinado de veces. Se compone de un valor inicial, una condición de final de bucle y un incremento/decremento.

|  |  |  |
| --- | --- | --- |
| **Formato** | **Ejemplo** | **Explicación** |
| **for** (ini;con;inc) | for (var n=0;n<5;n++)  {  alert (n);  } | Se define la variable n |
| { | con valor 0 y se  visualiza cinco veces |
| … | una ventana con los  números 0, 1, 2, 3 y 4 |
| } | cada vez ya que n |
|  | toma valores de 0 a 4 |
| /\* Ini:Inicial |  |
| Con; Condición |  |
| Inc; incremento |  |
| \*/ |  |

#### for .. in

Variante de la estructura **for** vista anteriormente que permite recorrer los componentes de un array u objeto sin tener que conocer su número de componentes.

|  |  |  |
| --- | --- | --- |
| **Formato** | **Ejemplo** | **Explicación** |
| **for** ( n **in** array)  {  …  }  /\* n: Índice \*/ | var vector = [1,2,3,4] for (indice in vector)  {  alert (vector[indice]);  } | Se define el array vector con cuatro componentes y luego se visualizan cuatro ventanas con los valores 1, 2, 3 y 4 que se corresponde con cada uno de los componentes del vector |



#### while

Estructura repetitiva que se puede cumplir 0 o más veces dependiendo de la condición definida en su construcción.

|  |  |  |
| --- | --- | --- |
| **Formato** | **Ejemplo** | **Explicación** |
| **while** (condición) | var n=1; while (n<7)  {  alert(n); n++;  } | Visualiza seis ventanas |
| { | (valores de 1 a 6 de n)  mientras n sea menor que |
| … | 7. Dentro del bucle se  incrementa n para que |
| } | llegue a 7 y la condición |
|  | sea falsa |

#### do .. while

Estructura repetitiva similar a **while** que se ejecuta una o varias veces dependiendo de la condición definida en su construcción

|  |  |  |
| --- | --- | --- |
| **Formato** | **Ejemplo** | **Explicación** |
| **do**  { | var n=1; do | Visualiza una única ventana con el valor 1 ya que la condición es falsa (2  no es mayor que 7) además |

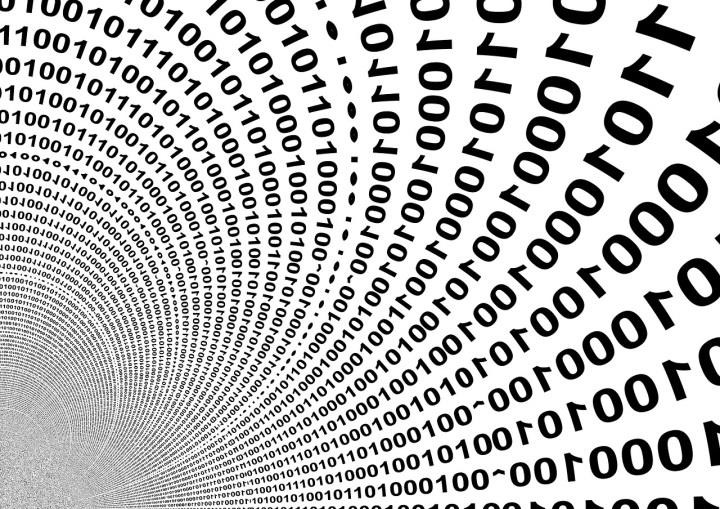
|  |  |  |
| --- | --- | --- |
| …  }  **while** (condición); | {  alert(n); n++;  } while (n>7); | de incrementar en uno n. |

#### switch

Estructura que sirve para comprobaciones múltiples que se puede usar en vez de una estructura if ..then .. else ..else ..else.. demasiado compleja. La palabra reservada *case* indica cada uno de los posibles valores a evaluar.

|  |  |  |
| --- | --- | --- |
| **Formato** | **Ejemplo** | **Explicación** |
| **switch** (variable) | var n=1;  var mensaje; switch (n)  {  case 1:  mensaje=”hola”; break;  case 2:  mensaje=”adiós”; break;  default: mensaje=”uno”; break;  }  alert(mensaje); | Si el valor de n es 1 se |
| { | asigna el valor “hola” a  mensaje, si el valor es 2 se |
| **case** valor1: | asigna “adiós” y si no es ni  1 ni 2 se asigna el valor |
| … | “uno” por lo que en este |
| **break**; | caso se visualizaría una  ventana con el mensaje |
|  | “hola” ya que n vale 1. La |
|  | instrucción *break* sirve |
| **case** valor2: | para romper la secuencia y |
| … | no ejecutar las  instrucciones que hubiera |
| **break**; | por debajo. |
| **default**: |  |
| … |  |
| **break**; |  |
| } |  |

#### Funciones

Una función en cualquier lenguaje de programación es un bloque de instrucciones agrupado que realiza una operación concreta y que se puede utilizar tantas veces como se quiera a través de una llamada. La

definición de la función lleva la palabra reservada **function** por delante seguida por paréntesis donde pueden ir parámetros pasados a dicha función.

Si a una variable se le asigna el valor que devuelve una función, esta función devolverá dicho valor con la palabra reservada **return**.

* **Formato del cuerpo de la función:**

**function** nombrefunción (parámetro1, parámetro2, …)

{

Instrucción1;

…..

}

/\* Una función puede no tener parámetros \*/

* **Llamada a la función**

**Nombrefunción** (parámetro1, parámetro2, ..);

var a = **Nombrefunción** (parámetro1, parámetro2, …)

/\* Si no lleva parámetros la función \*/

**Nombrefunción** ();

var a = **Nombrefunción** ();

* **Ejemplo:**

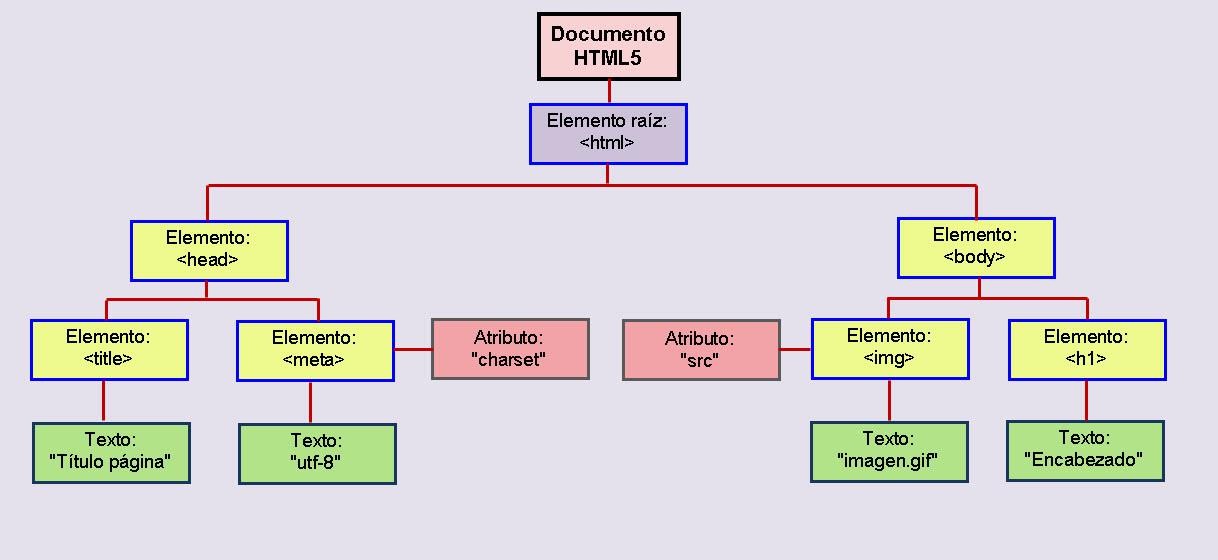
|  |  |
| --- | --- |
| **Function** sumar (num1, num2)  {  var suma=num1+num2;  **return** suma;  }  var n=1; var a=2;  var sum=sumar(n,a); alert(sum);  var j=1; var m=4;  sum=sumar(j,m)  alert(sum); | La función *sumar* suma dos números pasados como parámetros guardando dicha suma en la variable *suma* y devolviendo dicho valor con la palabra reservada **return.**  num1, num2 y suma son variables locales a la función y solo tienen validez en ella. En cambio n, a, j y m son variables globales que valen para todo el script.  En la primera ventana se visualiza 3 (num1 se corresponde con n y num2 con a) y en la segunda ventan se visualiza 5 (num1 es j y num2 m) |

# DOM

El ***Document Object Model*** o **DOM** (Modelo de Objetos del Documento) es una de las innovaciones en la construcción de páginas web dinámicas que permite el acceso y la manipulación de cualquier componente de un documento html (elementos, atributos, estilos, etc.) a través de un conjunto estándar de objetos que forman una estructura jerárquica de nodos.

El consorcio W3C se encarga de mantener las especificaciones del DOM y lo define de la siguiente manera:

*El Modelo de Objetos del W3C documento (DOM) es una plataforma y una interfaz que permite a los programas y scripts acceder y actualizar dinámicamente el contenido, la estructura y el estilo de un documento*



Prácticamente todos los lenguajes actuales tienen estructuras y elementos que les permiten realizar programación orientada a objetos. Javascript no es una excepción y, aunque no dispone de todas las características de los lenguajes orientados a objetos (ni la herencia ni el polimorfismo), permite la creación, definición de métodos y atributos de objetos a través de lo cual puede manipular sin problemas la estructura que crea el DOM.



*Un* ***objeto*** *es una entidad que dispone de propiedades o atributos y de operaciones específicas o métodos*

En el DOM, todos los elementos HTML están definidos como objetos, una propiedad es un valor que se puede obtener o usar como, por ejemplo, cambiar el contenido de un elemento HTML, y, por último, un método es una acción que se puede hacer sobre el objeto como, por ejemplo, añadir o eliminar un elemento HTML.

En el DOM HTML, el objeto *document* representa una página web y todos los demás objetos de dicha página descienden directa o indirectamente de él, es decir, es el nodo raíz.

#### Métodos DOM

A través de los métodos siguientes se puede buscar, acceder, crear, borrar o cambiar tanto elementos como sus valores asociados como pueden ser sus atributos o propiedades de estilo

|  |  |
| --- | --- |
| **Método** | **Descripción** |
| document.getElementById() | Búsqueda de elementos por **id** |
| document.getElementsByTagName() | Búsqueda de elementos por nombre de etiqueta |
| document.getElementsByClassName() | Búsqueda de elementos por nombre de clase CSS |
| Document.getElementsByName() | Búsqueda de elementos por nombre de elemento |
| *Elemento*.innerHTML=*”texto”* | Cambia el contenido de un elemento |
| *Elemento*.*atributo*=”*valor”* | Cambia el valor del atributo de un elemento |
| *Elemento.*setAttribute(*atributo, valor*) | Cambia el valor del atributo de un elemento |
| *Elemento.*style.*propiedad*=*”valor”* | Cambia el una propiedad de estilo de un elemento |
| document.createElement(“*elemento*”) | Crea un elemento HTML |
| document.removeChild(“*elemento*”) | Borra un elemento HTML |
| document.appendChild(“*elemento*”) | Añade un elemento HTML |
| document.replaceChild(“*elemento*”) | Reemplaza un elemento HTML |
| document.write(“*texto*”) | Escribe “*texto*” como contenido de la página web |

#### Propiedades DOM

Con las siguientes propiedades podemos acceder a los elementos en general de un documento html.

|  |  |
| --- | --- |
| **Propiedad** | **Descripción** |
| document.anchors | Devuelve todos los elementos <a> que tengan definido el atributo *name* |

|  |  |
| --- | --- |
| document.baseURI | Devuelve la URI absoluta base del documento |
| document.body | Devuelve el elemento <body> |
| document.cookie | Devuelve la cookie del documento |
| document.doctype | Devuelve del tipo de archivo del documento |
| document.documentElement | Devuelve el elemento <html> |
| document.documentMode | Devuelve el modo usado por el navegador |
| document.documentURI | Devuelve la URI absoluta del documento |
| document.domain | Devuelve el servidor de nombres |
| document.embeds | Devuelve todos los elementos <embed> |
| document.forms | Devuelve todos los elementos <form> |
| document.head | Devuelve el elemento <head> |
| document.images | Devuelve todos los elementos <img> |
| document.implementation | Devuelve la implementación DOM |
| document.inputEncoding | Devuelve al codificación del documento |
| document.lastModified | Devuelve el día y la hora de modificación del documento |
| document.links | Devuelve todos los elementos <area> y <a> que tengan definido el atributo *href* |
| document.readyState | Devuelve el estado de carga del documento |
| document.referrer | Devuelve la URI del documento referencia |
| document.scripts | Devuelve todos los elementos <script> |
| document.strictErrorChecking | Devuelve si el control de errores está activo |
| document.title | Devuelve el elemento <title> del documento |
| document.URL | Devuelve la URL completa del documento |

*Algunas de estas propiedades pueden devolver más un elemento, es decir, más de un valor por lo que estos valores deben recogerse en variables de tipo array de una dimensión donde cada componentes es uno de valores.*



#### Acceso a los nodos

Una vez creado el árbol de nodos DOM, el acceso a cualquiera de ellos se puede hacer o bien a través de sus nodos padre partiendo del nodo raíz o bien accediendo directamente a dicho nodo a través de los métodos y propiedades DOM.

* + Método **getElementsByTagName(“**etiqueta”**)**

Este método devuelve todos los elementos de un documento html que tengan como etiqueta la pasada como parámetro. Como puede haber varios elementos iguales, lo que implica que tienen etiquetas iguales, el resultado se guarda en una variable de tipo array.

#### Ejemplo breve:

var divisiones = document.getElementsByTagName(“div”);

En este ejemplo vamos a buscar todos los elementos <div> de nuestro documento y guardarlo en el array divisiones. Para ellos utilizamos el método getElementsByTagName() pasándole como parámetro la etiqueta “div” e indicando que busque desde la raíz del documento (document). En divisiones[0] estará el primer <div>, en divisiones[1], el el segundo <div> y así sucesivamente.

Ahora podríamos cambiar el lenguaje de cualquier <div> (atributo **lang**), por ejemplo, a través de los elementos de “divisiones” como podría ser

divisiones[0].lang=”en”;

#### Ejemplo completo:

<!DOCTYPE html>

<html lang=”es-es”>

<head>

<title>Título</title>

<meta charset=”utf-8”>

</head>

<body>

<div lang="en"></div>

<div lang="fr"></div>

<div lang="de"></div>

|  |  |
| --- | --- |
| <script type="text/javascript">  var divisiones=document.getElementsByTagName("div"); for (var i=0;i<divisiones.length;i++)  {  alert(divisiones[i].lang); //no cambia divisiones[i].lang="es"; alert(divisiones[i].lang);  }  </script>  </body>  </html> | |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |

En este ejemplo vemos que hay que tres elementos <div> cada uno de ellos con un lenguaje asociado (inglés, francés y alemán por este orden). A continuación vemos que está definido el elemento <script> dentro del cual vamos a cargar en una variable de tipo array de una dimensión (“divisiones”) cada uno de los elementos <div> y, con ayuda de un bucle, vamos a recorrer el vector “divisiones” y a cambiar el valor del atributo “lang” a español (es) en cada uno de los <div>. Para que sea más gráfico el script visualiza dos mensajes, el primero con el valor que tiene antes de entrar en el script (imágenes de la columna izquierda) y otro después de haberlo cambiado (imágenes de la columna derecha). Otro punto a tener en cuenta es la estructura del bucle; en él vemos que la variable de control “i” va de 0 a “divisiones.lenght”-1. La propiedad “lenght” de cualquier array devuelve el número de elementos por lo que en este caso “divisiones.lenght” devuelve 3 lo que implica que el bucle tiene que ir de 0 a 2 (“divisiones.lenght” -1).

* + Método **getElementsByName(**“nombre de elemento”**)**

Busca los elementos del documento cuya atributo “name” sea el parámetro pasado al método. Este atributo normalmente es único por lo que este método viene muy bien para acceder directamente a un elemento concreto



*Hay una excepción clara a lo de la unicidad del atributo* ***name*** *y son los*

*será tratada como una array.*

*atributo* ***name****. En este caso a la variable a la que se asigne este método*

*“botones de opción” o “radiobuttons” que comparten el mismo nombre o*

#### Ejemplo breve:

var nombre = document.getElementsByName(“nom”);

En este ejemplo buscamos por todo el doucumento html los elementos que tenga como atributo **name** el valor “nom” y lo asigna a la variable “nombre”.

Al igual que en el ejemplo anterior y suponiendo que solo hay un elemento con **name**=”nom”, a través de la variable “nombre” (más concretamente “nombre[0]” ya que se sigue interpretando como una variable array) podríamos cambiar cualquier atributo del elemento como, por ejemplo, el lenguaje asociado (atributo **lang**).

nombre[0].lang=”es”;

#### Ejemplo completo:

<!DOCTYPE html>

<html lang=”es-es”>

<head>

<title>Título</title>

<meta charset=”utf-8”>

</head>

<body>

<div lang="en" name="uno"></div>

<div lang="fr" name="dos"></div>

<div lang="de" name="tres"></div>

</body>

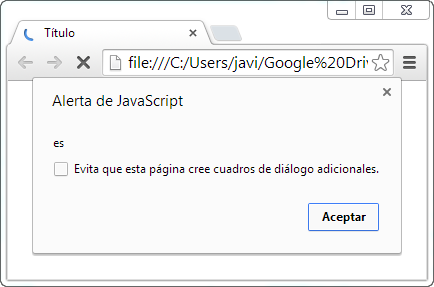
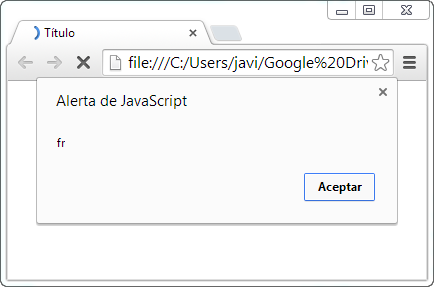
</html>

<script type="text/javascript">

var nombre=document.getElementsByName("dos"); alert(nombre[0].lang);

nombre[0].lang="es"; alert(nombre[0].lang);

</script>



En este ejemplo tenemos tres divisiones con tres nombres diferentes (uno, dos y tres) y con tres idiomas asociados diferentes (inglés, francés y alemán). Al ejecutar el script asignamos a la variable “nombre” el elemento html cuyo atributo **name** es “dos” (segunda división con idioma asociado francés o “fr”) y vemos que el primer mensaje indica que el lenguaje del elemento es “fr” (imagen columna izquierda) y que, cuando se cambia, su vloar es “es” (imagen columna derecha). Resaltar la idea de que nombre sigue siendo una variable array y que el elemento se guardará en la posición “0” del array.

* + Método **getElementById(**“id del elemento”**)**

Busca el único elemento del documento html cuyo atributo **id** coincida con el valor que se le ha pasado como parámetro.

#### Ejemplo breve:

var nombre = document.getElementById(“nom”);

En este ejemplo buscamos el único elemento que haya en el documento cuyo atributo **id** sea “nom” y lo guardamos en la variable “nombre”.

Al igual que en los ejemplos anteriores, a través de la variable “nombre” podríamos cambiar cualquier atributo del elemento como, por ejemplo, el lenguaje asociado (atributo **lang**).

nombre.lang=”es”;

<!DOCTYPE html>

<html lang=”es-es”>

<head>

<title>Título</title>

|  |  |
| --- | --- |
| <meta charset=”utf-8”>  </head>  <body>  <div lang="en" id="uno"></div>  <div lang="fr" id="dos"></div>  <div lang="de" id="tres"></div>  <script type="text/javascript">  var nombre=document.getElementById("dos"); alert(nombre.lang);  nombre.lang="es"; alert(nombre.lang);  </script>  </body>  </html> | |
|  |  |

Este ejemplo es muy parecido a los otros dos que hemos visto y la única diferencia es que la variable “nombre” no se toma como un array sino como una variable simple.

#### Creación de nodos

El proceso de creación de nodos va directamente relacionado con el de agregar nuevos elementos a una página web de forma dinámica, es decir, que dicho elemento no

estaba en el diseño original y se añade cuando sucede algún cambio o se lanza algún evento.

Los cuatro tipos más importantes de nodos de la especificación DOM son:

* + **Document**: Nodo raíz del que todos los demás descienden directa o indirectamente.
  + **Element**: Nodo que representa cada uno de los elementos HTML5. Puede contener atributos y otros nodos pueden descender de él.
  + **Attr**: Nodo que representa un atributo de un elemento HTML5
  + **Text**: Nodo que contiene el texto que va dentro de un elemento HTML5.

Los cuatro pasos que hay que seguir cuando se crea un nuevo elemento a un documento son:

1. Creación de un nodo de tipo **Element** que va a ser, en realidad, el elemento que queremos añadir.
2. Creación de un nodo de tipo **Text** que va a tener el contenido del elemento (normalmente es un texto)
3. Se añade el nodo **Text** como nodo hijo del nodo **Element**
4. Se añade el nodo **Element** al documento html como hijo o bien de otro elemento o bien de <body> o <head> dependiendo del tipo de elemento que sea.

#### Ejemplo básico de la creación de un artículo con el texto “Artículo”

/\* Crear un nodo de tipo **Element** \*/

var articulo=document.createElement(“article”);

/\*Crear un nodo de tipo **Text** \*/

var texto=document.createTextNode(“Artículo”);

/\*Añadir el nodo **Text** como hijo de **Element** \*/

articulo.appendChild(texto);

/\*Añadir el nodo **Element** como elemento de <body> \*/

document.body.appendChild(articulo);

**Ejemplo completo:**

<!DOCTYPE html>

<html lang=”es-es”>

<head>

<title>Título</title>

<meta charset=”utf-8”>

<style>

article {background-color:orange;}

</style>

</head>

<body>

</body>

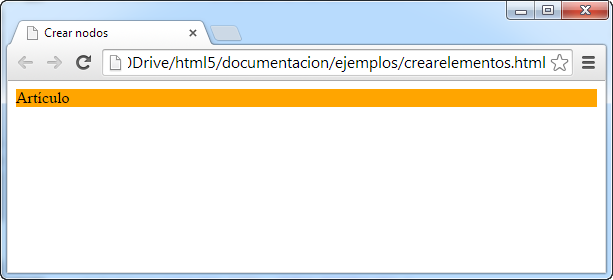
</html>

<script type="text/javascript">

var articulo = document.createElement("article"); var texto = document.createTextNode("Artículo"); articulo.appendChild(texto);

document.body.appendChild(articulo);

</script>



En este ejemplo se ve claramente que el elemento <article> es creado por el script de la página html y que se comporta como un elemento normal ya que, por ejemplo, se le pueden asignar estilos sin ningún problema.

#### Eliminación de nodos

La eliminación de nodos consta de dos pasos: búsqueda de elemento a través de los métodos que hemos visto anteriormente y borrado del elemento con la función **removeChild(**“nodo a borrar”**).**

#### Ejemplo básico de borrado del artículo con id=”articulo1”

var articulo = document.getElementById(“articulo1”); articulo.parentNode.removeChild(articulo);

En este caso buscamos el elemento cuyo **id** es “articulo1” y lo guardamos en la variable “articulo”. La siguiente instrucción

articulo.parentNode.removeChild(articulo); se interpreta de la siguiente manera:

*articulo.parentNode* es una propiedad que nos permite saber el padre de un elemento (en este caso el elemento es “articulo”). Averiguado el padre se le aplica la función **removeChild(**articulo**)** para borrar ese hijo del nodo padre.

*Si borras un nodo eliminas todos sus hijos a la vez por lo que no hay que ir uno por uno para quitarlos todos*



#### Ejemplo completo de borrado del artículo con id=”articulo1”

|  |
| --- |
| <!DOCTYPE html>  <html lang="es-es">  <head>  <title>Borrar nodos</title>  <meta charset="utf-8">  <style>  #articulo1  { background-color:orange;}  .fondoverde {background-color:#ccffcc;}  #articulo2 {background-color: grey;}  </style>  </head>  <body class="fondoverde">  <article id="articulo1">  <h1> Artículo 1</h1>  </article>  <article id="articulo2">  <h1> Artículo 2</h1>  </article>  <script type="text/javascript">  var articulo=document.getElementById("articulo1"); articulo.parentNode.removeChild(articulo);  </script> |

|  |
| --- |
| </body>  </html> |
|  |
|  |

En este ejemplo vemos como estaría la página html si no tuviera el script de borrado de un elemento (primera imagen) y luego como quedaría con dicho script (segunda imagen). En esta segunda imagen podemos observar que al borrar el elemento

<articulo> con id=“articulo1” hemos borrado también el elemento <h1> que dependía de él.

#### Reemplazar nodos

Con el método **replaceChild(**“nodo nuevo”, “nodo viejo”**)** podemos cambiar un elemento HTML5 por otro de forma dinámica.

#### Ejemplo básico de cambio de una artículo con id=”articulo1” por un párrafo con el texto “Párrafo uno”

var viejo = document.getElementById(“articulo1”); var nuevo = document.createElement(“`p”);

var texto = document.createTextNode(“Párrafo uno”); nuevo.appendChild(texto);

viejo.parentNode.replaceChild(nuevo,viejo);

Para buscar el nodo padre del elemento que queremos reemplazar utilizamos la misma forma que en el borrado de nodos y vemos que javascript permite reemplazar un elemento por otro diferente.

#### Ejemplo completo:

|  |
| --- |
| <!DOCTYPE html>  <html lang="es-es">  <head>  <title>Borrar nodos</title>  <meta charset="utf-8">  <style>  #articulo1  { background-color:orange;}  .fondoverde {background-color:#ccffcc;} |

|  |
| --- |
| #articulo2 {background-color: grey;}  p {background-color:#999ff;}  </style>  </head>  <body class="fondoverde">  <article id="articulo1">  <h1> Artículo 1</h1>  </article>  <article id="articulo2">  <h1> Artículo 2</h1>  </article>  <script type="text/javascript">  var viejo = document.getElementById("articulo1"); var nuevo = document.createElement("p");  var texto = document.createTextNode("Párrafo1"); nuevo.appendChild(texto); viejo.parentNode.replaceChild(nuevo,viejo);  </script>  </body>  </html> |

|  |
| --- |
|  |
|  |

En este ejemplo si no estuviera el script se vería la primera imagen y con el script cambiamos el elemento <article> con id=”articulo1” por un elemento <p> con el texto “Párrafo1” (segunda imagen).

#### Escribir un texto directamente en la página web

Con el método **write(**“texto”**)** aplicado al objeto **document** podemos insertar texto en nuestras páginas de forma dinámica.

#### Ejemplo:

|  |
| --- |
| <!DOCTYPE html>  <html lang="es-es">  <head>  <title>Borrar nodos</title>  <meta charset="utf-8">  <style>  .fondoverde {background-color:#ccffcc;}  </style>  </head>  <body class="fondoverde"> Inicio  <script type="text/javascript"> document.write(“En medio”);  </script>  Final  </body>  </html> |

|  |
| --- |
|  |

En este ejemplo se escribe primero “Inicio”, luego el script escribe “En medio” y por último “Fin”.

#### Cambiar el contenido de un elemento

A través de la propiedad **innerHTML=** podemos cambiar el contenido de un elemento.

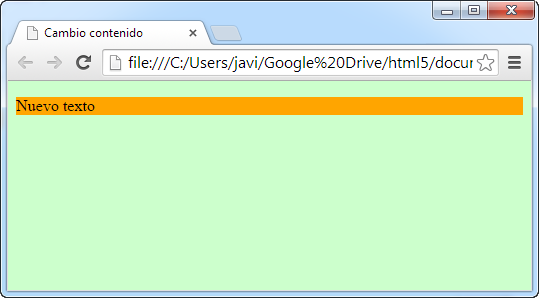
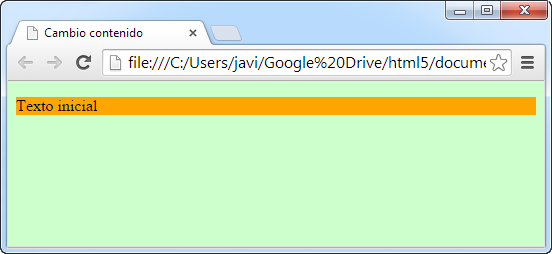
#### Ejemplo básico de cambio de texto de un elemento con id=“p1”

document.getElementById("p1").innerHTML = "Nuevo texto";

En este caso se cambiaría el texto que tuviera el elemento con id=”p1” y tendría el texto “Nuevo texto”

#### Ejemplo completo de cambio de texto de un elemento con id=”p1”:

|  |
| --- |
| <!DOCTYPE html>  <html lang="es-es">  <head>  <title>Cambio contenido</title>  <meta charset="utf-8">  <style>  .fondoverde {background-color:#ccffcc;} p  { background-color:orange;}  </style> |



</head>

<body class="fondoverde">

<p id="p1"> Texto inicial </p>

<script type="text/javascript"> document.getElementById("p1").innerHTML = "Nuevo texto";

</script>

</body>

</html>

En la primera imagen del ejemplo vemos cómo quedaría la página si tuviera el script y en la segunda es como se vería con el script de cambio de contenido del elemento cuyo id=”p1”

#### Cambio del valor de un atributo de un elemento html

Una posible forma para cambiar un atributo de un elemento html es la siguiente:

document.getElementById("*id*").*atributo*=”*valor”*;

#### Ejemplo breve del cambio de lenguaje a español del elemento con id=”p1”:

document.getElementById(“p1”).lang=”es”

* **Cambio del estilo de un elemento html**

El HTML DOM permite a Javascript cambiar los estilos de los elementos html. Una posible forma es la siguiente:

document.getElementById(“p1”).style.*propiedad*=”*nuevo estilo”*

**Ejemplo breve del cambio de color del fondo a naranja de un elemento con id=”p1”**

document.getElementById(“p1”).style.background-color=”orange”

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | *En todos estos ejemplos se ha usado una forma muy poco práctica de* |  |
| *insertar scripts Javascript ya que lo normal es usar un archivo externo que* |  |
| *se agrega a <head> compuesto de funciones que se llaman a través de* |  |
| *eventos* | | |

# EVENTOS

Un evento es una acción como puede ser un clic de ratón, una tecla pulsada, etc., que recoge un script o programa y, en base a esa acción, el programa realiza unas determinadas tareas.

Los eventos permiten al usuario interactuar con los elementos de cualquier programa e, incluso, pasar información por lo que es muy importante saberlos controlar para dotar de un mejor dinamismo a nuestras páginas web.



*Javascript es un* ***lenguaje basado en eventos****, es decir, que los programas escritos en este lenguaje pueden cambiar su comportamiento en base a los eventos que se produzcan.*

Ejemplos de eventos en una documento html son:

* Cuando el usuario hace clic con el ratón sobre un determinado elemento.
* Cuando una página web se carga.
* Cuando una imagen ha sido descargada
* Cuando se mueve el ratón sobre un elemento.
* Cuando un campo de formulario input se cambia.
* Cuando un formulario se envía.
* Cuando el usuario pulsa una tecla.

*Javascript permite asociar funciones a cada uno de los eventos que se producen.*



#### Tipos de eventos

Un mismo tipo de evento se puede aplicar sobre varios elementos html al igual que un elemento puede soportar varios tipos de eventos. Por ejemplo, hacer clic con el botón izquierdo del ratón se puede realizar sobre cualquier elemento HTML5, en cambio, cuando nos salimos de una página web el evento, denominado **onunload**, solo se puede asociar al elemento <body>.

La lista de eventos es muy extensa por lo aquí tienes un listado de los más importantes:

**Eventos de ratón:**

|  |  |
| --- | --- |
| **Evento** | **Descripción** |
| onclick | El evento se produce cuando el usuario hace clic sobre un elemento. |
| ondblclick | El evento se produce cuando el usuario hace doble-clic sobre un elemento |
| onmousemove | El evento se produce cuando el puntero se mueve mientras está encima del elemento |
| onmousenter | El evento se produce cuando el puntero entra en el elemento |
| onmouseleave | El evento se produce cuando el puntero sale del elemento |
| onmousedown | El evento se produce cuando se pulsa sin soltar un botón del ratón |

**Eventos de teclado:**

|  |  |
| --- | --- |
| **Evento** | **Descripción** |
| onkeydown | El evento se produce cuando el usuario presiona una tecla sin soltarla |
| onkeypress | El evento se produce cuando el usuario presiona una tecla |
| onkeyup | El evento se produce cuando el usuario suelta una tecla pulsada |

**Eventos de marcos/objetos:**

|  |  |
| --- | --- |
| **Evento** | **Descripción** |
| onload | El evento se produce cuando se carga un objeto |
| onresize | El evento se produce cuando se redimensiona la vista del documento |
| onunload | El evento se produce cuando se sale de la página (solo para  <body>) |

**Eventos de formulario:**

|  |  |
| --- | --- |
| **Evento** | **Descripción** |
| onblur | El evento se produce cuando se deselecciona un elemento |
| onchange | El evento se produce cuando se cambia el contenido |
| onfocus | El evento se produce cuando el elemento está seleccionado |
| onreset | El evento se produce cuando el formulario se resetea |
| onselect | El evento se produce cuando el usuario selecciona un texto |
| onsubmit | El evento se produce cuando el formulario es enviado |

**Ejemplos:**

* En este primer ejemplo se va a visualizar una ventana con el mensaje “Has pulsado el botón” cuando se hace un clic de ratón sobre un botón de un formulario

|  |
| --- |
| <!DOCTYPE html>  <html lang="es-es">  <head>  <title> Pulsar un botón </title>  <meta charset="UTF-8">  <style>  body { background-color:#ccffcc; }  </style>  <script>  function pulsar()  { alert("Has pulsado el botón"); }  </script>  </head> |

|  |
| --- |
| <body>  <form>  <input type="button" value="Pulsa el botón" onclick="pulsar()">  </form>  </body>  </html> |
|  |
|  |

Vemos en este ejemplo que se ha asociado un evento “onclick” al elemento “button” de un formulario, es decir, cuando se pulsa dicho elemento se ejecuta el código de la función “pulsa()” que abre una ventana, comando *alert,* con un mensaje.

* No es estrictamente necesario utilizar elementos de formulario para asociarles eventos. En el siguiente ejemplo vamos a aplicar el evento “onclick” a un párrafo o elemento <p>

|  |
| --- |
| <!DOCTYPE html>  <html lang="es-es">  <head>  <title> Pulsar sobre el párrafo </title>  <meta charset="UTF-8">  <style>  body { background-color:#ccffcc; }  p { background-color:yellow; width: 90px;  height: 60px; }  </style>  <script>  function pulsar()  { alert("Has pulsado sobre el párrafo");  }  </script>  </head>  <body>  <br><br>  <p onclick="pulsar()"> |

|  |
| --- |
| Pulsa sobre el párrafo  </p>  </body>  </html> |
|  |
|  |

En este ejemplo al pulsar el botón izquierdo del ratón sobre el párrafo de fondo amarillo, se ejecuta el código de la función “pulsar()” que abre una ventana con el mensaje “Has pulsado sobre el párrafo”.

* Ahora vamos a utilizar eventos para cambiar las propiedades de los elementos de nuestras páginas a través de lo visto en el tema anterior, DOM, En el siguiente ejemplo tenemos dos botones que, cuando se pulse sobre ellos, van a cambiar el color de fondo de un párrafo.

<!DOCTYPE html>

<html lang="es-es">

<head>

<title> Cambiando de color </title>

<meta charset="UTF-8">

<style>

body { background-color:#ccffcc; }

p { background-color:yellow; width: 200px;

height: 60px; }

</style>

<script>

function verde()

function azul()

{

document.getElementById("p1").style.background= "blue";

}

{

document.getElementById("p1").style.background= "green";

}

</script>

</head>

<body>

<form>

<br><br>

<p id="p1">

Cambiando el color al párrafo

</p>

</body>

</html>

<input type="button" value="verde" onclick="verde()">

<input type="button" value="azul" onclick="azul()">

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |

En la primera imagen se ven los dos botones y el párrafo con color amarillo de fondo. Al pulsar el botón con el texto “azul” , el color de fondo del párrafo pasa a ser azul debido a que se ejecuta la función “azul” (segunda imagen). También se puede pulsar el botón verde y cambiar el color de fondo del párrafo a verde a través del código de la función “verde”. Para hacer este cambio de color accedemos al párrafo a través de su “id” que en el ejemplo es “p1”

* En el siguiente ejemplo vamos a utilizar el evento “onmousemove” para cambiar el color de fondo de un párrafo cuando pasa el ratón por encima de él

|  |
| --- |
| <!DOCTYPE html>  <html lang="es-es">  <head>  <title> Ratón sobre elemento </title>  <meta charset="UTF-8">  <style>  body { background-color:#ccffcc; }  p { background-color:yellow;  width: 200px; height: 60px; } |

|  |
| --- |
| </style>  <script>  function mover()  { document.getElementById("párrafo").style.background= "red";  }  </script>  </head>  <body>  <br><br>  <p onmousemove="mover()" id="párrafo">  Ratón sobre elemento  </p>  </body>  </html> |

|  |
| --- |
|  |
|  |

En este ejemplo, cuando el ratón pasa por encima del párrafo con id “párrafo” cambia su color de fondo de amarillo a rojo.

* También podemos manipular elementos html con los métodos DOM vistos en el tema anterior y los eventos en tiempo real. En este ejemplo vamos a cambiar el texto de un párrafo cuando se pulsa un botón.

|  |
| --- |
| <!DOCTYPE html>  <html lang="es-es">  <head>  <title> Cambio contenido </title>  <meta charset="UTF-8">  <style>  body { background-color:#ccffcc; }  p { background-color:yellow; width: 200px;  height: 60px; }  </style>  <script>  function cambiar()  { document.getElementById("párrafo").innerHTML= "Cambiando el texto"; }  </script>  </head>  <body>  <br><br>  <form> |



</form>

<p id="párrafo">

Texto inicial

</p>

</body>

</html>

<input type="button" value="pulsa" onclick="cambiar()">

|  |
| --- |
|  |

En este ejemplo, cuando se pulsa sobre el botón (evento onclick), se cambia el texto que hay dentro del párrafo con id “párrafo”

* Ahora vamos a crear elementos a través de la aplicación de eventos y del DOM

|  |
| --- |
| <!DOCTYPE html>  <html lang="es-es">  <head>  <title> Creación párrafo </title>  <meta charset="UTF-8">  <style>  body { background-color:#ccffcc; }  p { background-color:yellow;  width: 200px; height: 60px; }  </style> |

|  |
| --- |
| <script>  function crear()  {  var párrafo = document.createElement("p");  var texto = document.createTextNode("Texto interior"); párrafo.appendChild(texto); document.body.appendChild(párrafo);  }  </script>  </head>  <body>  <br><br>  <form>  <input type="button" value="pulsa" onclick="crear()">  </form>  </body>  </html> |

|  |
| --- |
|  |
|  |

En este ejemplo, cuando se pulsa el botón, se crea un párrafo con el texto “Texto interior” de la misma forma que hemos visto en el tema sobre DOM. Al ser el elemento “p” un elemento de tipo bloque se crea debajo del formulario y se le aplican los estilos como a cualquier otro elemento de la página.

* A continuación vamos a ver un ejemplo de cómo crear un elemento con un “id” concreto y como borrarlo a través de dos botones. Tomaremos de base el ejemplo anterior.

|  |
| --- |
| <!DOCTYPE html>  <html lang="es-es"> |

|  |
| --- |
| <head>  <title> Creación y borrado de párrafo </title>  <meta charset="UTF-8">  <style>  body { background-color:#ccffcc; } p { background-color:yellow;  width: 200px; height: 60px; }  </style>  <script>  function crear()  {  var párrafo = document.createElement("p");  var texto = document.createTextNode("Texto interior"); párrafo.appendChild(texto);  párrafo.id=”par”; document.body.appendChild(párrafo);  }  function borrar()  {  var párrafo = document.getElementById("par"); párrafo.parentNode.removeChild(párrafo);  } |

</script>



</head>

<body>

<br><br>

<form>

<input type="button" value="crear" onclick="crear()">

<input type="button" value="borrar" onclick="borrar()">

</form>

</body>

</html>

|  |
| --- |
|  |
|  |

En este ejemplo disponemos de dos botones, uno para crear el elemento <p> con id=”par” y otro botón para borrar ese mismo elemento. La segunda imagen refleja la creación del elemento <p> y la tercera el borrado del mismo elemento.

* En el siguiente ejemplo vamos a ver cómo ejecutar funciones cuando se carga una página sin necesidad de botones.

|  |
| --- |
| <!DOCTYPE html>  <html lang="es-es">  <head> |

|  |
| --- |
| <title> Al cargar la página </title>  <meta charset="UTF-8">  <style>  body { background-color:#ccffcc; }  p { background-color:yellow; width: 200px;  height: 60px; }  </style>  <script>  function cargar()  {  var párrafo = document.createElement("p");  var texto = document.createTextNode("Texto interior"); párrafo.appendChild(texto);  párrafo.id="par"; document.body.appendChild(párrafo);  }  </script>  </head>  <body onload="cargar()">  </body>  </html> |

|  |
| --- |
|  |

En este ejemplo se ve que la parte <body> … </body> no contiene ningún texto ni ningún elemento HTML pero, al cargar la página, se produce el evento “onload” que, asociado a <body>, crea un párrafo o elemento <p> con id=”par” y “Texto interior” como texto de dicho párrafo.

***EXPRESIONES REGULARES***

En JavaScript, hay varios patrones y banderas que puedes usar en expresiones regulares para definir patrones de búsqueda más específicos y controlar cómo se realiza la búsqueda. Aquí tienes algunos de los patrones y banderas más comunes:

* *Patrones:*

**Caracteres Literales**: Los caracteres literales son simplemente caracteres que coinciden exactamente con ellos mismos. Por ejemplo, /hello/ coincidirá con la palabra "hello" en el texto.

**Clases de Caracteres**: Las clases de caracteres te permiten especificar conjuntos de caracteres que quieres buscar. Por ejemplo:

[aeiou]: Coincide con cualquier vocal.

[0-9]: Coincide con cualquier dígito numérico.

.: Coincide con cualquier carácter excepto los saltos de línea.

**Anclas:** Las anclas te permiten especificar ubicaciones específicas en el texto. Por ejemplo:

^: Coincide con el comienzo de una cadena.

$: Coincide con el final de una cadena.

Cuantificadores: Los cuantificadores te permiten especificar la cantidad de veces que un patrón debe repetirse. Por ejemplo:

\*: Coincide con cero o más repeticiones del patrón anterior.

+: Coincide con una o más repeticiones del patrón anterior.

?: Coincide con cero o una repetición del patrón anterior.

{n}: Coincide exactamente con n repeticiones del patrón anterior.

{n, m}: Coincide con al menos n y como máximo m repeticiones del patrón anterior.

* *Banderas:*

g (global): Realiza una búsqueda global, es decir, busca todas las coincidencias en lugar de detenerse después de encontrar la primera coincidencia.

i (ignore case): Realiza una búsqueda que ignora la distinción entre mayúsculas y minúsculas. Por ejemplo, /hello/i coincidirá con "hello", "Hello", "HELLO", etc.

m (multiline): Modifica el comportamiento de los anclajes ^ y $ para que coincidan con el comienzo y el final de cada línea en lugar del comienzo y el final del texto completo.

s (dotAll): Hace que el metacarácter . coincida con cualquier carácter, incluidos los saltos de línea.

Estos son solo algunos de los patrones y banderas más comunes en JavaScript. Hay otros patrones y banderas más avanzados disponibles, pero estos son los fundamentales que se utilizan con mayor frecuencia.

En JavaScript, cuando defines una expresión regular, se utiliza el formato ***/patrón/flags***. Aquí, el primer y último carácter / sirven como delimitadores para indicar que lo que está dentro de ellos es el patrón de la expresión regular.

Por ejemplo:

/[aeiou]/gi: Aquí, [aeiou] es el patrón de la expresión regular, que busca coincidencias con cualquier vocal en la cadena. Las letras g e i son flags (banderas) que modifican el comportamiento de la búsqueda.

/patrón/flags: El patrón es la parte de la expresión regular que describe qué estamos buscando, y las flags modifican cómo se realiza la búsqueda.

Los delimitadores / son necesarios para que JavaScript pueda distinguir entre el patrón de la expresión regular y las flags. Sin ellos, JavaScript no entenderá correctamente dónde termina el patrón y dónde comienzan las flags. Por lo tanto, siempre comenzamos una expresión regular con / y la terminamos con /.

En la variable vocales: ***/patrón/flags***

//  10. Escribe un programa que pida una frase y escriba las vocales que aparecen

    var frase = prompt("Introduce una frase:");

    var vocales = frase.match(/[aeiou]/gi);

    var conteo = vocales ? vocales.length : 0;

    document.write("La frase tiene " + conteo + " vocales");