JavaScript

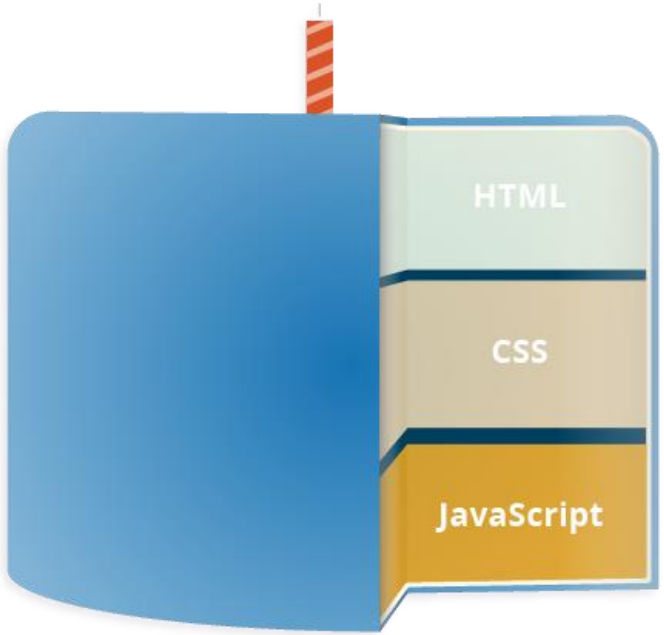
Historia

- El JavaScript es un lenguaje de programación que surgió por la necesidad de ampliar las posibilidades del HTML.
- En efecto, al poco tiempo de que las páginas web apareciesen, se hizo patente que se necesitaba algo más que las limitadas prestaciones del lenguaje básico, ya que el HTML solamente provee de elementos que actuan exclusivamente sobre el texto y su estilo, pero no permite, como ejemplo sencillo, ni siquiera abrir una nueva ventana o emitir un mensaje de aviso.
- La temprana aparición de este lenguaje, es posiblemente la causa de que se haya convertido en un estándar soportado por todos los navegadores actuales, a diferencia de otros, que solo funcionan en los navegadores de sus firmas creadoras.

Historia

- Como tantas otras aportaciones al mundo www, fue Netscape quien inició la implementación de JavaScript, y posteriormente, una alianza entre Netscape y Sun, creadora del lenguaje Java, permitió que JavaScript tomase la debida consistencia, definiéndose como una variante de Java, pero mucho más simple de usar.
- Esto no significa que JavaScript sea Java simplificado o reducido. Salvo el nombre y la sintaxis, JavaScript no tiene mucho en común con Java, pero cumple su propósito de lenguaje auxiliar del HTML en los navegadores, y sólo en ellos ya que no es posible utilizarlo fuera del entorno de las páginas.
- No hay que confundirlo con el JScript de Microsoft, que aunque bastante parecido, no tiene la compatibilidad del original JavaScript, ya que, como todo lo que hacen, está pensado exclusivamente para su propio navegador.

- JavaScript es un lenguaje de programación que te permite realizar actividades complejas en una página web
- HTML es un lenguaje de marcado que usa la estructura para dar un sentido al contenido web.
- CSS es un lenguaje de reglas en cascada que usamos para aplicar un estilo a nuestro contenido en HTML.
- JavaScript Es un lenguaje de programación que te permite crear contenido nuevo y dinámico, controlar archivos de multimedia, crear imágenes animadas y muchas otras cosas más.



```
<p>Player 1: Chris</p>
```

```
p {
  font-family: 'helvetica neue', helvetica, sans-serif;
  letter-spacing: 1px;
  text-transform: uppercase;
  text-align: center;
  border: 2px solid rgba(0,0,200,0.6);
  background: rgba(0,0,200,0.3);
  color: rgba(0,0,200,0.6);
  box-shadow: 1px 1px 2px rgba(0,0,200,0.4);
  border-radius: 10px;
  padding: 3px 10px;
  display: inline-block;
  cursor:pointer;
}
```

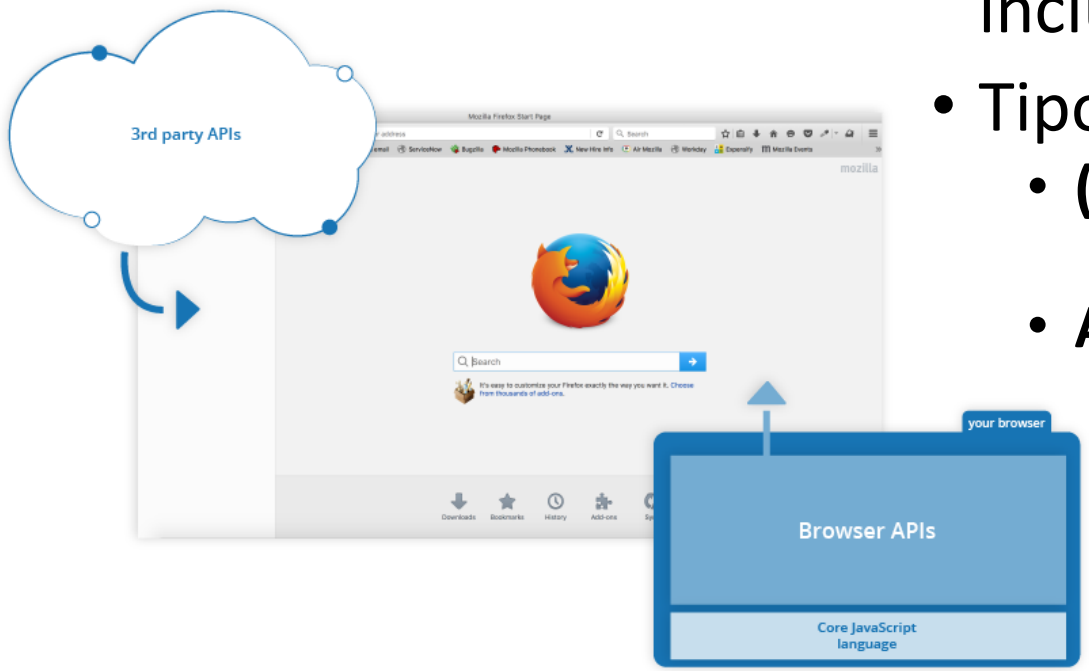
Resultado

```
var para =
document.querySelector('
p');

para.addEventListener('cli
ck', updateName);

function updateName() {
  var name =
prompt('Enter a new
name');
  para.textContent =
'Player 1: ' + name;
}
```

¿Qué podemos hacer?

- Almacenar valores útiles dentro de variables
- Las operaciones escritas en formato de texto (Conocidas como "**Strings**" en lenguaje de programación).
- Para hacer funcionar el código en respuesta a algunos eventos que están ocurriendo en la página web.
- ¡muchas más cosas!

APIS



- Los APIS son inserciones de líneas, incluso bloques gigantes de código listos para usar que permiten a un desarrollador implementarlo a programas que de cualquier otra forma podría ser difícil o incluso imposible de terminar
- Tipos
  - **(Browser APIs)**
    - El DOM (Modelo de Objeto de Documento) API
  - **APIs de Terceras personas**
    - Google Maps API

¿Cómo añadir JavaScript a tu página web?

- JavaScript por dentro
  - Etiquetas
    - <script> </script> dentro del body

- JavaScript Externo
  - crea un nuevo archivo en la misma dirección o sitio de tu archivo muestra de HTML, asegúrate que lleve la extensión .js pe:script.js
  - <script src="script.js"></script>

Comentarios

- Comentarios de una sola línea escritos después de dos barras inclinadas (//), Ejemplo:

```
// I am a comment
```

- Comentarios de varias líneas escritos entre las cadenas /* y */, ejemplo:

```
/*
  I am also
  a comment
*/
```

Ejemplo — Juego adivina el número

- Crear un juego sencillo del tipo "adivina el número". Debe elegir un número aleatorio entre 1 y 100, luego desafiar al jugador a adivinar el número en 6 intentos. Después de cada intento debería decirle al jugador si ha acertado o no - y si está equivocado, debería decirle si se ha quedado corto, o se ha pasado. También debería decir los números que ya se han probado anteriormente. El juego acabará una vez que el jugador acierte o cuando se acaben los intentos. Cuando el juego se acabe, se le debe dar al jugador la opción de volver a jugar.

- Ejemplo

Pasos para posible solución

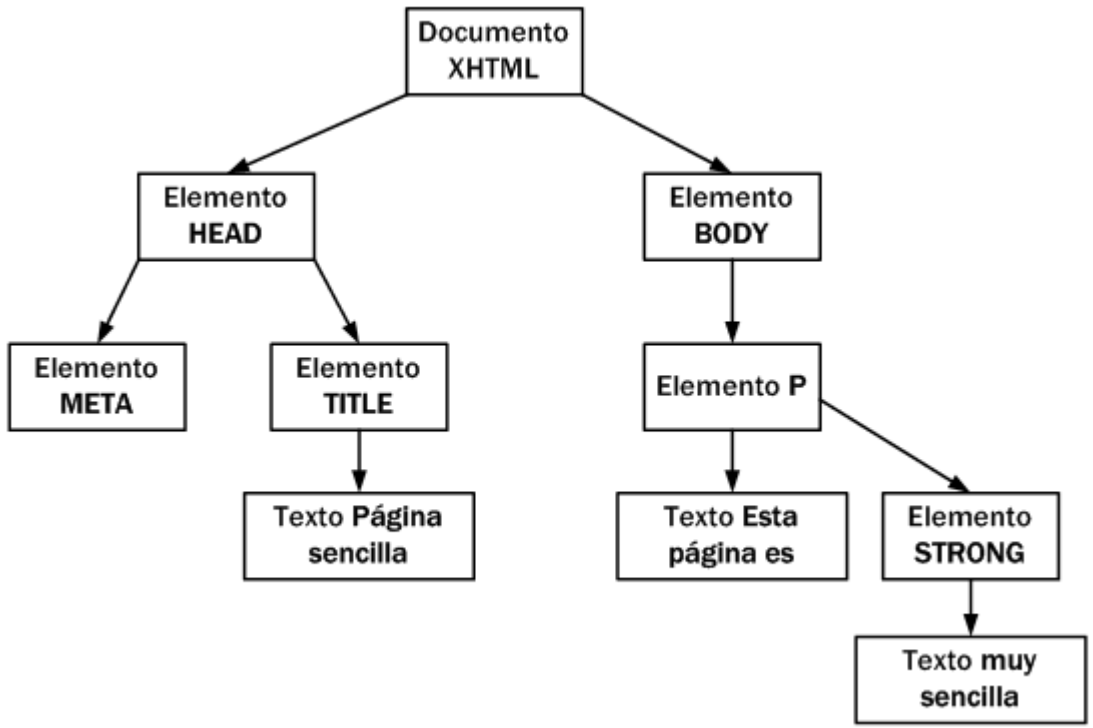
1. Generar un número aleatorio entre 1 y 100.
2. Registrar el número del intento en el que el jugador se encuentre. Empezando en 1.
3. Darle al jugador una forma de adivinar cuál es el número.
4. Una vez que se ha introducido un número, registrarlo en alguna parte para que el jugador pueda ver sus intentos previos.
5. Siguiente, comprobar si el número es correcto.
6. Si es correcto:
   a) Mostrar un mensaje de felicitaciones.
   b) Hacer que el jugador no pueda introducir más intentos.
   c) Mostrar un control que le permita al jugador reiniciar el juego.
7. Si es incorrecto y al jugador todavía le quedan intentos:
   a) Decirle al jugador que ha fallado.
   b) Dejar al jugador intentarlo de nuevo.
   c) Incrementar el número de intentos en 1.
8. Si el jugador falla y no le quedan turnos:
   a) Decirle al jugador que el juego se ha acabado.
   b) Hacer que el jugador no pueda introducir más intentos.
   c) Mostrar un control que permita al jugador volver a empezar el juego.
9. Una vez que el juego se reinicia, asegurarse de que la lógica del juego y la IU estén completamente restablecidos, volver al paso 1.

¿Qué es el DOM?

- El modelo de objeto de documento (DOM) da una representación del documento como un grupo de nodos y objetos estructurados que tienen propiedades y métodos.

- DOM transforma todos los documentos (X)HTML en un conjunto de elementos llamados nodos, que están interconectados y que representan los contenidos de las páginas web y las relaciones entre ellos. Por su aspecto, la unión de todos los nodos se llama "árbol de nodos".

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<title>Página sencilla</title>
</head>

<body>
<p>Esta página es <strong>muy sencilla</strong></p>
</body>
</html>
```



TIPOS DE NODOS

- La especificación completa de DOM define 12 tipos de nodos, aunque las páginas XHTML habituales se pueden manipular manejando solamente cuatro o cinco tipos de nodos:

- Document, nodo raíz del que derivan todos los demás nodos del árbol.
- Element, representa cada una de las etiquetas XHTML. Se trata del único nodo que puede contener atributos y el único del que pueden derivar otros nodos.
- Attr, se define un nodo de este tipo para representar cada uno de los atributos de las etiquetas XHTML, es decir, uno por cada par atributo=valor.
- Text, nodo que contiene el texto encerrado por una etiqueta XHTML.
- Comment, representa los comentarios incluidos en la página XHTML.
- Los otros tipos de nodos existentes que no se van a considerar son DocumentType, CDataSection, DocumentFragment, Entity, EntityReference, ProcessingInstruction y Notation.

ACCESO DIRECTO A LOS NODOS

- GETELEMENTSBYTAGNAME()
  - La función getElementsByTagName(nombreEtiqueta) obtiene todos los elementos de la página (X)HTML cuya etiqueta sea igual que el parámetro que se le pasa a la función.

```
var parrafos = document.getElementsByTagName("p");
```

  - El valor que devuelve la función es un array con todos los nodos que cumplen la condición de que su etiqueta coincide con el parámetro proporcionado. El valor devuelto es un array de nodos DOM, no un array de cadenas de texto o un array de objetos normales. Por lo tanto, se debe procesar cada valor del array de la forma que se muestra en las siguientes secciones.

```
var primerParrafo = parrafos[0];

for(var i=0; i<parrafos.length; i++) {
  var parrafo = parrafos[i];
}

var parrafos =
document.getElementsByTagName("p");
var primerParrafo = parrafos[0];
var enlaces =
primerParrafo.getElementsByTagName("a");
```

ACCESO DIRECTO A LOS NODOS

- GETELEMENTSBYNAME()
  - La función getElementsByName() es similar a la anterior, pero en este caso se buscan los elementos cuyo atributo **name** sea igual al parámetro proporcionado. En el siguiente ejemplo, se obtiene directamente el único párrafo con el nombre indicado:

```
var parrafoEspecial = document.getElementsByName("especial");
<p name="prueba">...</p>
<p name="especial">...</p>
<p>...</p>
```

  - Normalmente el atributo name es único para los elementos HTML que lo definen, por lo que es un método muy práctico para acceder directamente al nodo deseado.
  - En el caso de los elementos HTML radiobutton, el atributo name es común a todos los radiobutton que están relacionados, por lo que la función devuelve una colección de elementos.

ACCESO DIRECTO A LOS NODOS

- GETELEMENTBYID()
  - La función getElementById() es la más utilizada cuando se desarrollan aplicaciones web dinámicas. Se trata de la función preferida para acceder directamente a un nodo y poder leer o modificar sus propiedades.

  - La función getElementById() devuelve el elemento (X)HTML cuyo atributo **id** coincide con el parámetro indicado en la función.
  - Como el atributo id debe ser único para cada elemento de una misma página, la función devuelve únicamente el nodo deseado.

  ```
  var cabecera = document.getElementById("cabecera");
  <div id="cabecera">
  <a href="/" id="logo">...</a>
  </div>
  ```

  - La función getElementById() es tan importante y tan utilizada en todas las aplicaciones web, que casi todos los ejemplos y ejercicios que siguen la utilizan constantemente.

ACCESO DIRECTO A LOS NODOS

- QUERYSELECTOR()
  - La función querySelector() acepta como parámetro un **selector** (CSS) que identifica el elemento (o elementos) a seleccionar. En el caso de esta función, únicamente es devuelto el primer elemento que cumple la condición. Si no existe el elemento, el valor retornado es null.

```
var logo = document.querySelector(".enlace");
<div id="cabecera">
<a href="/" class="enlace">...</a>
</div>
<div id="cuerpo">
<p>Loren ipsum <a href="enlace">...</a></p>
</div>
```

  - En este caso, a pesar de existir varios elementos de la clase enlace, únicamente es seleccionado el primero de ellos.

ACCESO DIRECTO A LOS NODOS

- QUERYSELECTORALL()
  - La función querySelectorAll() acepta como parámetro un selector que identifica el elemento (o elementos) a seleccionar. Esta función devuelve un objeto de tipo NodeList con los elementos que coincidan con el selector.

```
var enlaces = document.querySelectorAll(".enlace");
<div id="cabecera">
  <a href="/" class="enlace">...</a>
</div>
<div id="cuerpo">
    <p>Loren ipsum <a href="enlace">...</a></p>
</div>
```

  - Para acceder a los elementos almacenados en NodeList, recorremos el objeto como si de un array se tratase:

```
for (var i=0; i<enlaces.length; i++) {
    var enlaces = enlaces[i];
}
```

ACCESO DIRECTO A LOS ATRIBUTOS (X)HTML

- Mediante DOM, es posible acceder de forma sencilla a todos los atributos (X)HTML y todas las propiedades CSS de cualquier elemento de la página.
- Los atributos XHTML de los elementos de la página se transforman automáticamente en propiedades de los nodos. Para acceder a su valor, simplemente se indica el nombre del atributo XHTML detrás del nombre del nodo.
- El siguiente ejemplo obtiene de forma directa la dirección a la que enlaza el enlace:

```
var enlace = document.getElementById("enlace");
console.log(enlace.href);                                   // muestra http://www...com
<a id="enlace" href="http://www...com">Enlace</a>
```

- En el ejemplo anterior, se obtiene el nodo DOM que representa el enlace mediante la función document.getElementById(). A continuación, se obtiene el atributo href del enlace mediante enlace.href. Para obtener por ejemplo el atributo id, se utilizaría enlace.id.
- Las propiedades CSS requieren un paso extra para acceder a ellas. Para obtener el valor de cualquier propiedad CSS del nodo, se debe utilizar el atributo style. El siguiente ejemplo obtiene el valor de la propiedad margin de la imagen:

```
var imagen = document.getElementById("imagen");
console.log(imagen.style.margin);
<img id="imagen" style="margin:0; border:0;" src="logo.png" />
```

Variables

- var nombreDeVariable1 [= valor1] [, nombreDeVariable2 [= valor2] ... [, nombreDeVariableN [=valorN]]];
- se recomienda siempre declarar las variables, sin importar si están en una función o un ámbito global.
- Asignar un valor a una variable no declarada implica crearla como variable global

La declaración de variables (y todas las declaraciones en general) se procesa antes de ejecutar cualquier código

---

```
bla = 2;
var bla;
//  Es entendido implicitamente como:
var bla;
bla = 2;
```

```
var a;
console.log(a);                // Imprime "undefined" o "" dependiendo del navegador.
```

Ejemplo

- var x = y, y = 'A';
- console.log(x + y);

¿Qué imprime?

- // Imprimirá
- "undefinedA"

Operadores lógicos

Operador	Descripción	Ejemplo
+	Suma	6 + 9
-	Resta	20 - 15
*	Multiplicación	3 * 7
/	División	10 / 5

Operadores de compración

Operador	Descripción	Ejemplo
===	Estricta igualdad (¿es exactamente lo mismo?)	5 === 2 + 4
!==	No igualdad (¿no es lo mismo?)	'Chris' !== 'Ch' + 'ris'
<	Menor que	10 < 6
>	Mayor que	10 > 20
>=	Mayor o igual	10>=10
<=	Menor o igual	6 <= 10

Sentencia if

```
if (condición) sentencia1 [else sentencia2]
```

```
if (x = y) {
   /* sentencia */
}
```

```
if (condición1)
   sentencia1
else if (condición2)
   sentencia2
else if (condición3)
   sentencia3
...
else
   sentenciaN
```

```
if (cipher_char == from_char) {
   result = result + to_char;
   x++;
} else
   result = result + clear_char;
```

ciclos

```
while (condicion)
  sentencia
```

```
n = 0;
x = 0;
while (n < 3) {
  n ++;
  x += n;
}
```

```
do sentencia while (condicion);
```

```
do {
   i += 1;
   document.write(i);
} while (i < 5);
```

Funciones

- La definición de una función (también llamada declaración de función o sentencia de función) consiste de la palabra clave (reservada) function, seguida por:

- El nombre de la función (opcional).
- Una lista de argumentos para la función, encerrados entre paréntesis y separados por comas (,).
- Las sentencias JavaScript que definen la función, encerradas por llaves, { }.

- Los parámetros primitivos (como puede ser un número) son pasados a las funciones **por valor**

- Si pasa un objecto (p. ej. un valor no primitivo, como un Array o un objeto definido por el usuario) como parámetro, y la función cambia las propiedades del objeto, este cambio es visible desde afuera de la función

Ejemplo

```
function myFunc(theObject) {
  theObject.make = "Toyota";
}

var mycar = {make: "Honda", model: "Accord", year: 1998},
    x,
    y;

x = mycar.make;     // x toma el valor "Honda"

myFunc(mycar);
y = mycar.make;     // y toma el valor "Toyota“
// (la propiedad make fue cambida por la funcion)
```

- Tenga en cuenta que asignar un nuevo objeto al parámetro no tendrá ningún efecto fuera de la función, porque esto está cambiando el valor del parámetro en lugar de una de las propiedades del objeto

```
function myFunc(theObject) {
  theObject = {make: "Ford", model: "Focus", year: 2006};
}
var mycar = {make: "Honda", model: "Accord", year: 1998},
    x,
    y;
x = mycar.make;     // x toma el valor "Honda"
myFunc(mycar);
y = mycar.make;     // y sigue con el valor "Honda"
```

Función anónima

- var square = function(number) {return number * number};
- var x = square(4) //x obtiene el valor 16

- var factorial = function fac(n) {return n<2 ? 1 : n*fac(n-1)};

- print(factorial(3));

Evento	**Descripción**	**Elementos para los que está definido**
onblur	Deseleccionar el elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onchange	Deseleccionar un elemento que se ha modificado	<input>, <select>, <textarea>
onclick	Pinchar y soltar el ratón	Todos los elementos
ondblclick	Pinchar dos veces seguidas con el ratón	Todos los elementos
onfocus	Seleccionar un elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onkeydown	Pulsar una tecla (sin soltar)	Elementos de formulario y <body>
onkeypress	Pulsar una tecla	Elementos de formulario y <body>
onkeyup	Soltar una tecla pulsada	Elementos de formulario y <body>
onload	La página se ha cargado completamente	<body>
onmousedown	Pulsar (sin soltar) un botón del ratón	Todos los elementos
onmousemove	Mover el ratón	Todos los elementos

Evento	**Descripción**	**Elementos para los que está definido**
onmouseout	El ratón "sale" del elemento (pasa por encima de otro elemento)	Todos los elementos
onmouseover	El ratón "entra" en el elemento (pasa por encima del elemento)	Todos los elementos
onmouseup	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
onreset	Inicializar el formulario (borrar todos sus datos)	<form>
onresize	Se ha modificado el tamaño de la ventana del navegador	<body>
onselect	Seleccionar un texto	<input>, <textarea>
onsubmit	Enviar el formulario	<form>
onunload	Se abandona la página (por ejemplo al cerrar el navegador)	<body>

Manejadores de eventos como atributos (X)HTML

- El código se incluye en un atributo del propio elemento (X)HTML

```
<input type="button"
value="Pinchame y verás"
onclick="alert('Gracias por
pinchar');" />
```

```
<div onclick="console.log('Has
pinchado con el ratón');"
onmouseover="console.log('Acab
as de pasar el ratón por
encima');">
    Puedes pinchar sobre este
elemento o simplemente pasar el
ratón por encima
</div>
```

MANEJADORES DE EVENTOS Y VARIABLE THIS

- En los eventos, se puede utilizar la variable this para referirse al elemento XHTML que ha provocado el evento

```
<div id="contenidos" style="width:150px;
height:60px; border:thin solid silver"
onmouseover="document.getElementById('
contenidos').style.borderColor='black';"
onmouseout="document.getElementById('c
ontenidos').style.borderColor='silver';">
    Sección de contenidos...
</div>
```

```
<div id="contenidos" style="width:150px;
height:60px; border:thin solid silver"
onmouseover="this.style.borderColor='blac
k';"
onmouseout="this.style.borderColor='silver
';">
    Sección de contenidos...
</div>
```

MANEJADORES DE EVENTOS COMO FUNCIONES EXTERNAS

- Esta técnica consiste en extraer todas las instrucciones de JavaScript y agruparlas en una función externa. Una vez definida la función, en el atributo del elemento (X)HTML se incluye el nombre de la función, para indicar que es la función que se ejecuta cuando se produce el evento.

- La llamada a la función se realiza de la forma habitual, indicando su nombre seguido de los paréntesis y de forma opcional, incluyendo todos los argumentos y parámetros que se necesiten.

- El principal inconveniente de este método es que en las funciones externas no se puede seguir utilizando la variable this y por tanto, es necesario pasar esta variable como parámetro a la función.

```
function resalta(elemento) {                    switch(elemento.style.borderColor) {
                                                 case 'silver':
                                                case 'silver silver silver silver':
                                                case '#c0c0c0':
                                                  elemento.style.borderColor = 'black';
                                                 break;
                                                 case 'black':
                                                  case 'black black black black':
                                                  case '#000000':
                                                elemento.style.borderColor = 'silver';
                                                 break;   } }
<div style="width:150px; height:60px; border:thin solid silver" onmouseover="resalta(this)"
onmouseout="resalta(this)">
  Sección de contenidos...
</div>
```

Manejadores de eventos semánticos

- Esta técnica es una evolución del método de las funciones externas, ya que se basa en utilizar las propiedades DOM de los elementos (X)HTML para asignar todas las funciones externas que actúan de manejadores de eventos

- La técnica de los manejadores semánticos consiste en:

- Asignar un identificador único al elemento (X)HTML mediante el atributo id.
- Crear una función de JavaScript encargada de manejar el evento.
- Asignar la función externa al evento correspondiente en el elemento deseado.

- <input id="pinchable" type="button" value="Pinchame y verás" onclick="alert('Gracias por pinchar');" />
- Se puede transformar en:

- // Función externa
- function muestraMensaje() {
- alert('Gracias por pinchar');
- }
-
- // Asignar la función externa al elemento
- document.getElementById("pinchable").onclick = muestraMensaje;
-
- // Elemento XHTML
- <input id="pinchable" type="button" value="Pinchame y verás" />