



Funciones

Concepto de función



Sintaxis

- Una función de JavaScript se define con la palabra clave `function`, seguido por un nombre, y, a continuación, entre paréntesis los parámetros de la función.

```
function nombre(parámetro1, parámetro2,...) {  
    código a ejecutar  
}
```

- Los nombres de funciones pueden contener letras, dígitos, guiones bajos y símbolos de dólar (las mismas reglas que para los nombres de variable).
- Si los paréntesis incluyen varios parámetros, éstos se separarán por comas (,).
- El código que se ejecuta, por la función, se coloca entre llaves: {}

Invocación de una función

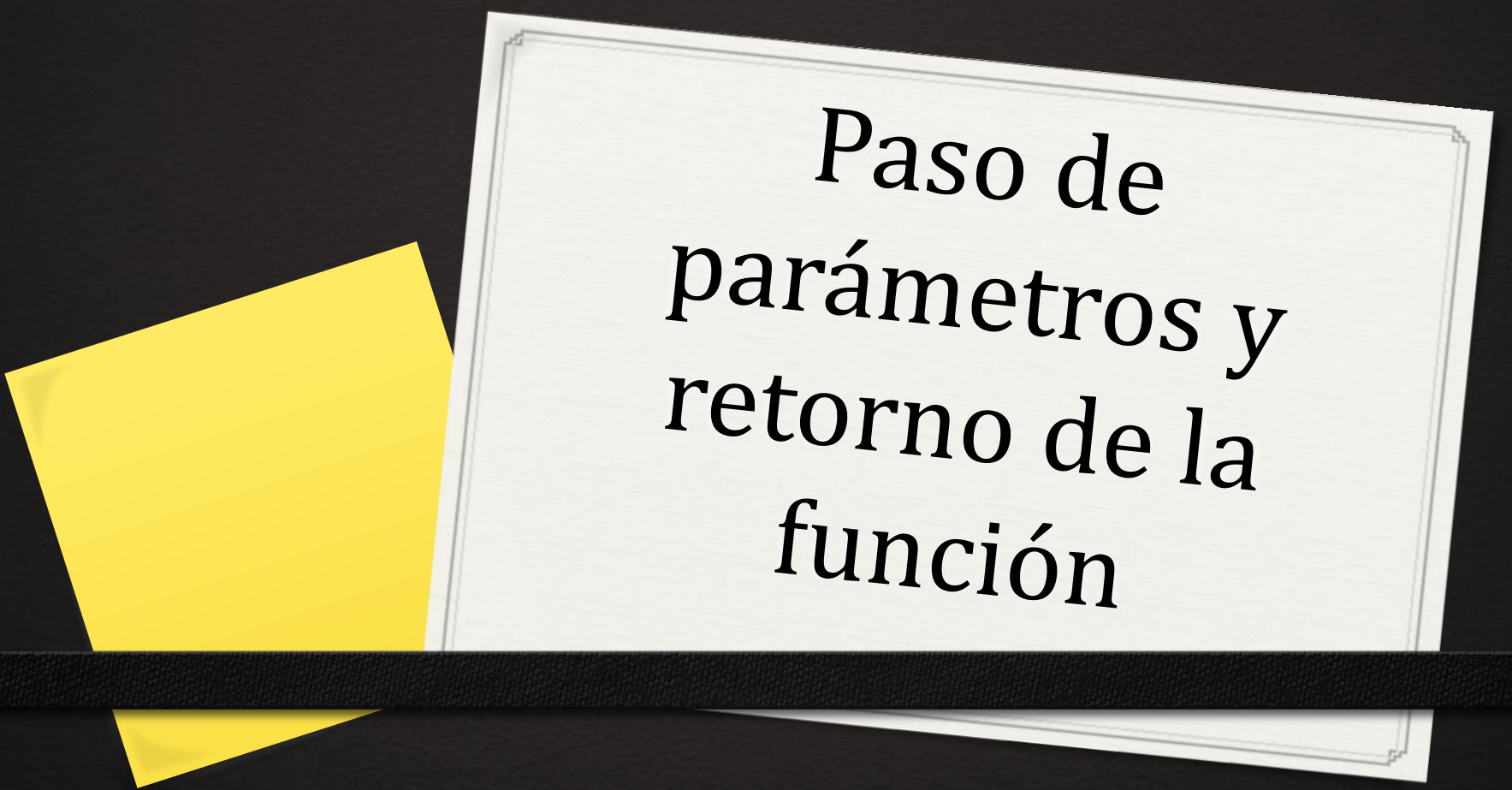
- El código dentro de la función se ejecutará cuando "algo" invoca (llama) a la función:
 - Cuando se produce un evento (por ejemplo, un usuario hace clic en un botón)
 - Cuando se invoca desde algún código JavaScript
 - Automáticamente (la función se autoinvoca)
- Cuando las funciones adquieren utilidad y potencia en JS es cuando se trabaja con las funciones junto a los eventos.



Manos a la obra

Crear un script que mediante una función devuelva la fecha del sistema y la muestre en el navegador

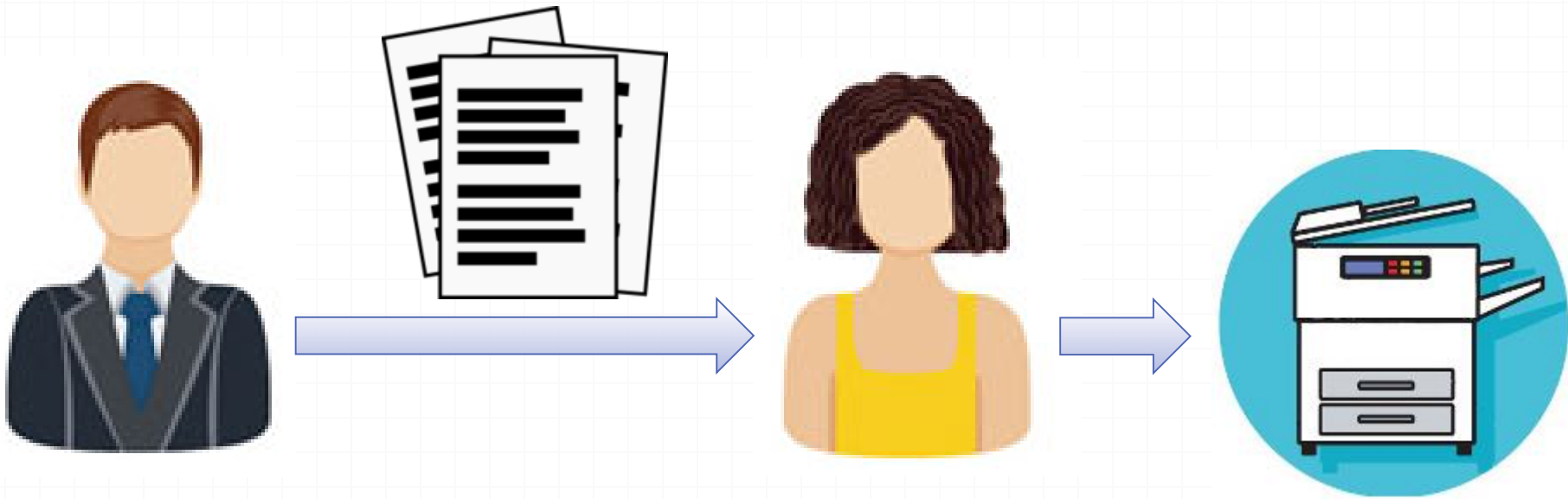
```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Una función sencillita</title>
  <script>
    function fechaHoy() {
      var hoy=new Date();
      document.write(hoy);
    }
  </script>
</head>
<body>
  <h1>¿Qué día es hoy?</h1>
  <p>Hoy es...</p>
  <script> fechaHoy(); </script>
</body>
</html>
```



Paso de
parámetros y
retorno de la
función

¿Qué es un parámetro?

```
function nombre(parámetro1, parámetro2,...) {  
    código a ejecutar  
}
```



Ejemplo

○ Función que calcula la suma de dos números:

```
function sumar(num1, num2) {  
    var resultado= num1 + num2  
}
```

○ Llamada a la función desde un punto cualquiera de la página:

```
sumar(4,6);
```



```
function sumar() {  
    var num1 = 4;  
    var num2 = 6;  
    var resultado= num1 + num2  
}
```




Manos a la obra

Crear un documento html en el que deberemos incluir una función que sume dos números. Luego deberemos llamar varias veces a esa función desde el documento html

```
<head>
  <meta charset="utf-8">

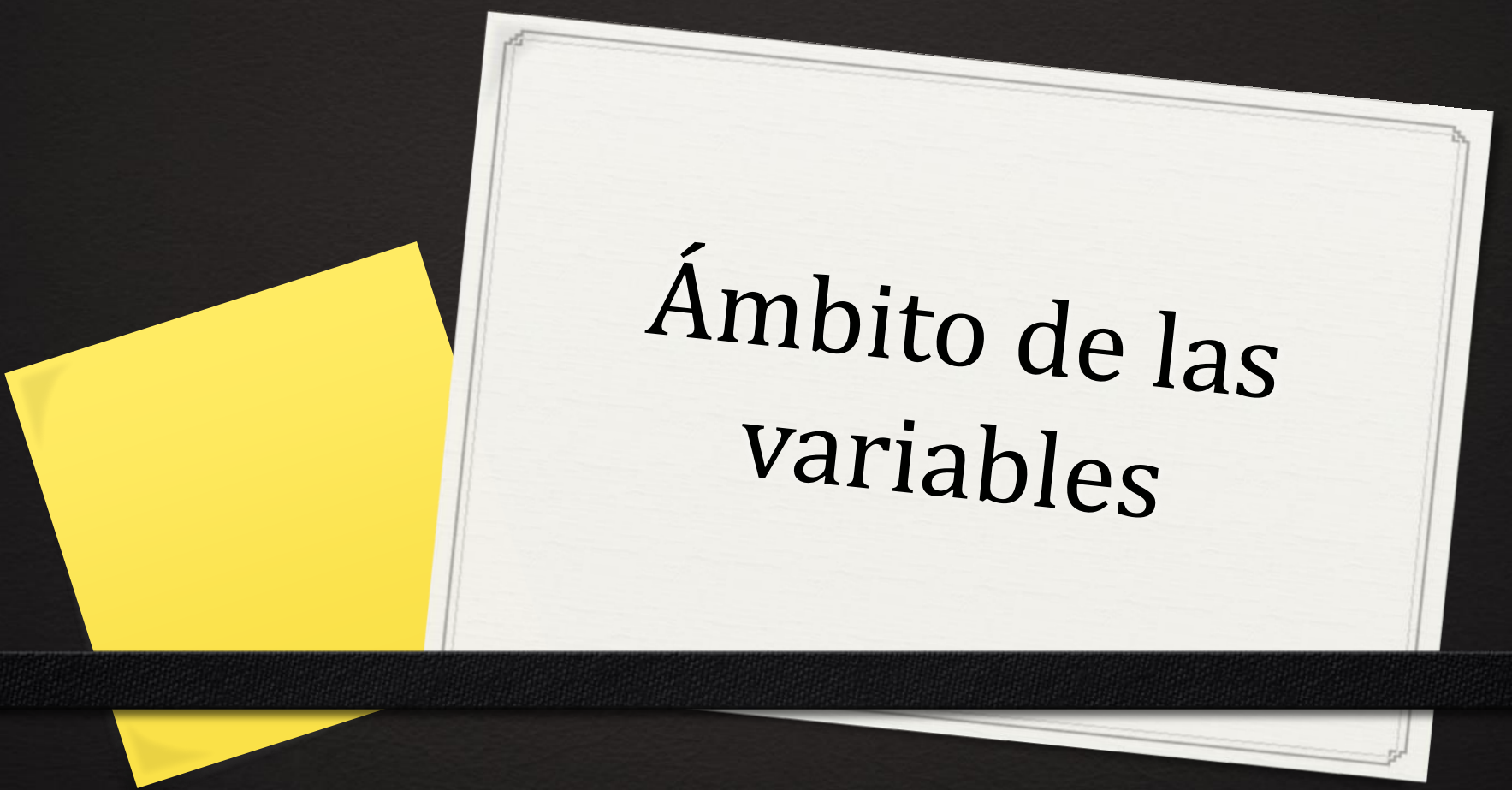
  <script>
    function myFunction(a, b) {
      var resultado= a + b;
      document.write(resultado);
    }
  </script>
</head>
```

```
<body>

  <h1>Suma de números</h1>

  <p>Primera suma: 4 + 3 =
    <script>
      myFunction(4, 3);
    </script>
  </p>

  <p>Segunda suma: 5 + 8 =
    <script>
      myFunction(5, 8);
    </script>
  </p>
</body>
```



Ámbito de las variables

Variables locales

- Las variables declaradas dentro de una función son locales a la función.

- Esto quiere decir que sólo son accesibles dentro de la función.

```
/*El código externo a la función no puede usar la variable marcaCoche*/
```

```
function myFunction() {  
    var marcaCoche = "Volvo";  
    /*El código dentro de la función puede utilizar la variable marcaCoche*/  
}
```

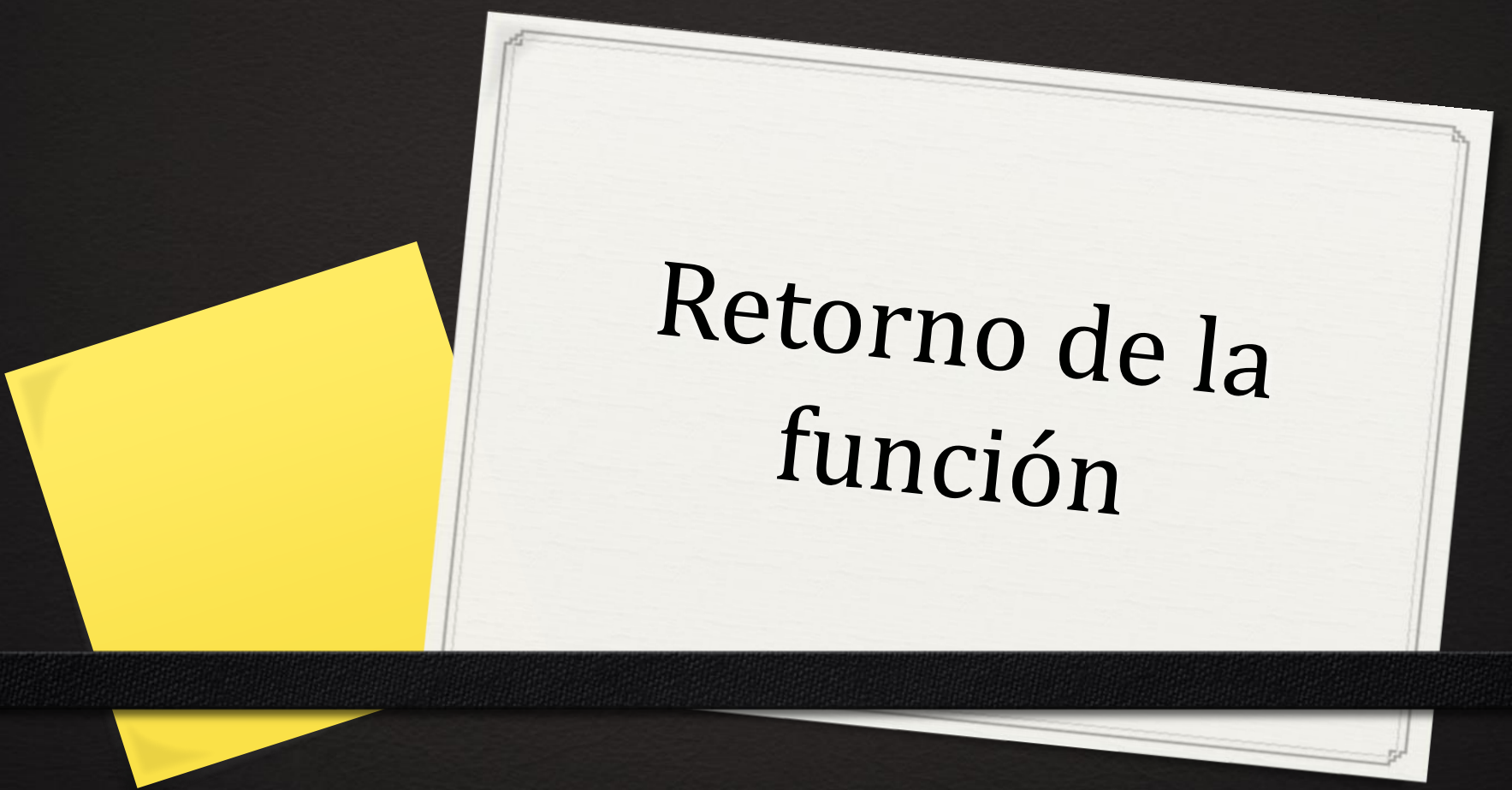
- Como las variables locales sólo se reconocen en el interior de sus funciones, se puede asignar el mismo nombre a las variables de diferentes funciones.
- Las variables locales se crean cuando se inicia una función, y se borran cuando se termina la función.

Variables globales

- Una variable declarada fuera de una función es una variable **global**.
- Una variable con alcance global puede ser utilizada por cualquier función de la página web

```
var marcaCoche = "Volvo";  
/*El código externo a las funciones pueden usar la  
variable marcaCoche*/  
function myFunction() {  
    /*El código dentro de la función puede utilizar  
    la variable marcaCoche*/  
}
```

- Las variables globales se destruyen cuando se cierra la página



Retorno de la
función

Retorno de la función

- Las funciones de JavaScript pueden devolver valores a través de la sentencia `return`.
- Cuando JavaScript encuentra una sentencia **return** dentro de una función, finalizará la ejecución de esa función.
 - Si la función fue invocada desde una sentencia, JavaScript "volverá" para ejecutar el código que esté después de la sentencia que invocaba a la función
- Múltiples return
 - En principio, en Javascript las funciones sólo pueden devolver un valor.
 - Aún así, en una misma función podemos colocar más de un return (ayudándonos por ejemplo de un condicional).

```
function multipleReturn(numero){  
  var resto = numero % 2  
  if (resto == 0)  
    return 0  
  else  
    return numero  
}
```



Manos a la obra

Crear un documento html en el que deberemos definir una función que calcule el total a pagar por un producto.

Para que pueda calcular este precio a pagar deberemos pasarle el precio original y la cantidad de unidades a comprar.



Eventos

Concepto de Eventos

- Las aplicaciones web creadas con el lenguaje JavaScript pueden utilizar el *modelo de programación basada en eventos*
- En este tipo de programación, los scripts se dedican a esperar a que el usuario "haga algo" (que pulse una tecla, que mueva o haga clic con el ratón, que cierre la ventana del navegador, etc.).
 - A continuación, el script responde a la acción del usuario normalmente procesando esa información y generando un resultado.
- JavaScript permite definir cierto código que se ejecute cuando se detecta un evento.

```
<elemento-HTML evento = 'código JavaScript'>  
<elemento-HTML evento = "código JavaScript">  
<button onclick="displayDate()">La fecha es...</button>
```
- Las funciones o código JavaScript que se definen para cada evento se denominan "*manejador de eventos*"

Eventos más comunes

Existen muchos eventos que se pueden producir, aunque los más comunes son los siguientes:

Evento	Descripción
onchange	Un elemento HTML ha cambiado
onclick	El usuario hace click sobre el elemento HTML
onmouseover	El usuario pasa el ratón por encima de un elemento HTML
onmouseout	El usuario mueve el ratón fuera de un elemento HTML
onkeydown	El usuario pulsa una tecla del teclado
onload	El navegador finaliza la carga de la página



Manos a la obra

- Crear un documento html con tres párrafos: Cuando pasemos el ratón por encima deberá mostrarse un mensaje que diga ¡Bien hecho!. Al hacer clic sobre el segundo párrafo deberá mostrarse un mensaje que diga ¡Buen trabajo!. En el tercer párrafo deberá parecer el alert cuando se hace clic sobre la palabra aquí
- A la página anterior añadir un botón. Cuando hagamos clic sobre el botón deberá mostrarse un mensaje que diga ¡Buen trabajo!
- Crear una página web que ponga simplemente un título y un párrafo (vale nuestro lorem ipsum). Al terminar de cargar debe de mostrar un mensaje que diga que la página se ha terminado de cargar.
- Modifica la página anterior y añade un botón de votación, de manera que cuando el usuario pique sobre él se abra una ventana donde nos pregunte si nos guste la página y nos de la opción de Aceptar o cancelar (método `confirm`)



Funciones y Eventos



Manos a la obra

Crear un documento HTML que presente un pequeño texto y un botón de manera que al picar sobre el botón, se abra una ventana emergente que muestre la fecha del sistema

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Eventos con función asociada</title>
</head>
<body>

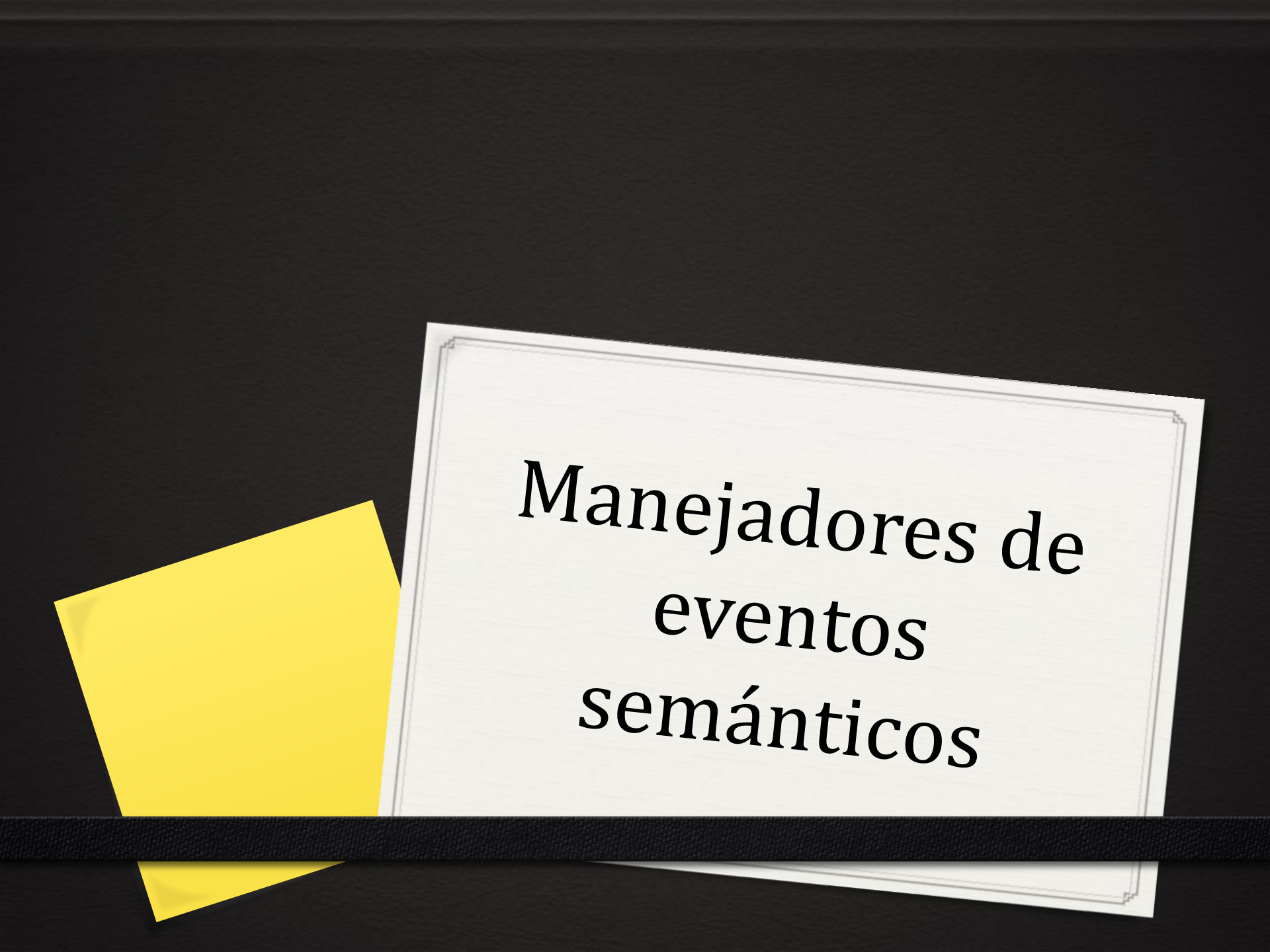
  <h1>Evento en botón</h1>

  <button onclick="miFuncion()">Pincha aquí</button>

  <script>
    function miFuncion(){
      alert("¡¡¡Buen trabajo!!!");
    }
  </script>

</body>
</html>
```





Manejadores de
eventos
semánticos

- Los métodos que se han visto para añadir manejadores de eventos (como atributos HTML y como funciones externas) tienen un grave inconveniente: "ensucian" el código HTML de la página.
- Como es conocido, una de las buenas prácticas básicas en el diseño de páginas y aplicaciones web es la separación de los contenidos (HTML) y su aspecto o presentación (CSS).
 - Siempre que sea posible, también se recomienda separar los contenidos (HTML) y su comportamiento o programación (JavaScript).
- Existe un método alternativo para definir los manejadores de eventos de JavaScript: *los manejadores de eventos semánticos*.
- La técnica de los manejadores semánticos consiste en:
 1. Asignar un identificador único al elemento HTML mediante el atributo id.
 2. Crear una función de JavaScript encargada de manejar el evento.
 3. Asignar la función externa al evento correspondiente en el elemento deseado.

Ejemplo

o El siguiente ejemplo:

```
<button onclick="alert('Gracias por pinchar');">  
  Pincha aquí  
</button>
```

o Lo podemos transformar en:

1. Asignar un identificador único al elemento HTML mediante el atributo id.

```
<button id="pinchable">Pincha aquí</button>
```

2. Crear una función de JavaScript encargada de manejar el evento.

```
function muestraMensaje() {  
  alert('Gracias por pinchar');  
}
```

3. Asignar la función externa al evento correspondiente en el elemento deseado.

```
document.getElementById("pinchable").onclick = muestraMensaje;
```


o Este último paso es la clave de la técnica:

1. Se obtiene el elemento al que se quiere asociar la función externa

```
document.getElementById("pinchable");
```

2. A continuación, se utiliza una propiedad del elemento con el mismo nombre que el evento que se quiere manejar.

```
document.getElementById("pinchable").onclick = ...
```

3. Por último, se asigna la función externa mediante su nombre

```
document.getElementById("pinchable").onclick = muestraMensaje;
```

Comentario

- o El script anterior¿ funcionaría si lo pusiéramos en el head?
 - o NOOOO
- o ¿Por qué?
 - o Porque el código HTML se lee de arriba abajo, si llamamos al método getElementById en el head ese elemento aún no existe.
- o ¿Cómo lo podemos solucionar?
 - o Agregando un evento que indique que cuando se haya terminado de cargar la página es cuando debería asociar el evento al elemento con el id="pulsar"

```
<head>
  <meta charset="utf-8">
  <title>Eventos sencillos</title>

  <script>
    /*Definimos la función*/
    function mostrarMensaje(){
      alert('Gracias por pulsar');
    }

    window.onload=function(){
      document.getElementById("pulsar").onclick=
        mostrarMensaje;
    }
  </script>
</head>
```



Manos a la obra

- Vamos a crear un reloj digital que se muestre en la página cuando ésta se cargue:
 - Primeramente, definiremos un div en el documento html al que podemos asignar el id="reloj" y que será donde vamos a mostrar el reloj. Podemos ponerle un tamaño de 20pt.
 - Deberemos crear una función (a la que podéis llamar obtenerHora):
 - deberá generar una variable de tipo fecha de la que deberemos extraer la hora, los minutos y los segundos).
 - Deberá mostrar la hora:minutos:segundos obtenidos en el div creado en el primer punto
 - Queremos que el reloj vaya mostrando en cada momento el segundo y se vaya actualizando automáticamente. Una manera de conseguirlo, es utilizando la función setTimeout que permite llamar cíclicamente a una función cada cierto intervalo de tiempo.
 - En nuestro ejemplo podemos usarla como setTimeout(obtenerHora(),500) que lo que hace es llamar a la función que hemos creado cada 500 milisegundos (medio segundo).

A white rectangular card with a thin double-line border is positioned diagonally. The word "this" is written in a black, lowercase, serif font on the right side of the card. To the left of the white card is a bright yellow rectangular card, also tilted. A thick, black, textured horizontal band crosses the bottom of the scene, partially obscuring the bottom edges of both cards. The background is a solid black.

this

Variable this

- o JavaScript define una variable especial llamada this que se crea automáticamente y que se emplea en algunas técnicas avanzadas de programación.
- o En los eventos, se puede utilizar la variable this para referirse al elemento HTML que ha provocado el evento. Esta variable es muy útil para ejemplos como el siguiente.



Manos a la obra

- 0 Crear un script para conseguir el efecto de imagen de sustitución.

```
<body>

  <div id="imagen">
    
  </div>
</body>
```

```
<script>
  function ratonEntra(MyImage) {
    MyImage.src = "../img/imagen2.png";
  }

  function ratonSale(MyImage) {
    MyImage.src = "../img/imagen1.png";
  }
</script>
```



JS HTML DOM

DOM

- DOM = **D**ocument **O**bject **M**odel
- Con el DOM de HTML, JavaScript puede acceder y cambiar todos los elementos de un documento HTML.
- Una página HTML normal no es más que una sucesión de caracteres, por lo que es un formato muy difícil de manipular. Por ello, los navegadores web transforman automáticamente todas las páginas web en una estructura más eficiente de manipular.
 - Esta transformación la realizan todos los navegadores de forma automática.
 - Cuando se carga una página web, el navegador crea el DOM de la página.
- Con el modelo de objetos, JavaScript puede realizar todas las tareas que necesita para crear HTML dinámico:
 - Puede cambiar todos los elementos HTML en la página
 - Puede cambiar todos los atributos de HTML en la página
 - Puede cambiar todos los estilos CSS en la página
 - Puede eliminar elementos y atributos HTML existente
 - Puede agregar nuevos elementos y atributos HTML
 - Puede reaccionar a todos los eventos de HTML existentes en la página
 - Puede crear nuevos eventos en la página HTML
- ***DOM HTML es un estándar de cómo obtener, cambiar, añadir o eliminar elementos HTML.***

El árbol de nodos

- DOM transforma todos los documentos 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>
```

```
<html>
```

```
<head>
```

```
  <title>Página sencilla</title>
```

```
</head>
```

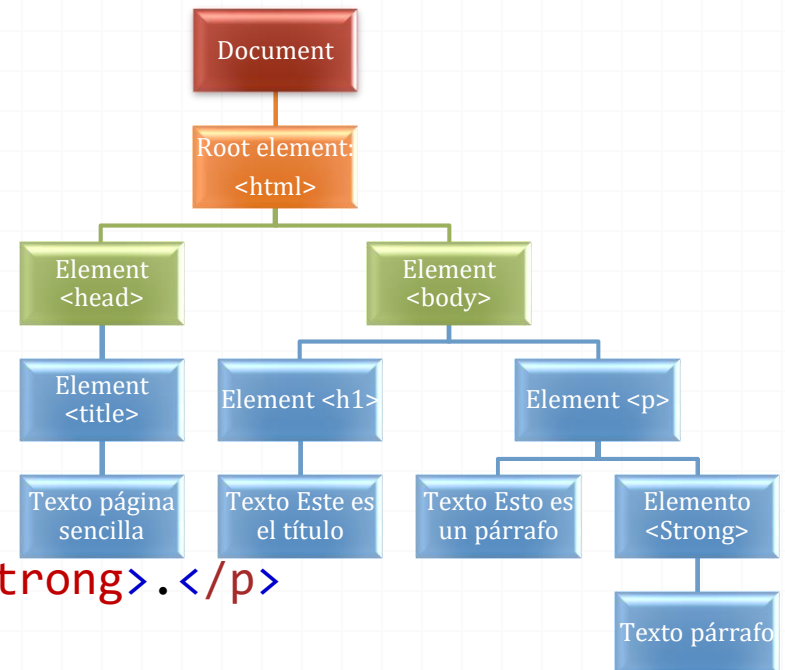
```
<body>
```

```
  <h1>Este es el título</h1>
```

```
  <p>Esto es un <strong>párrafo</strong>.</p>
```

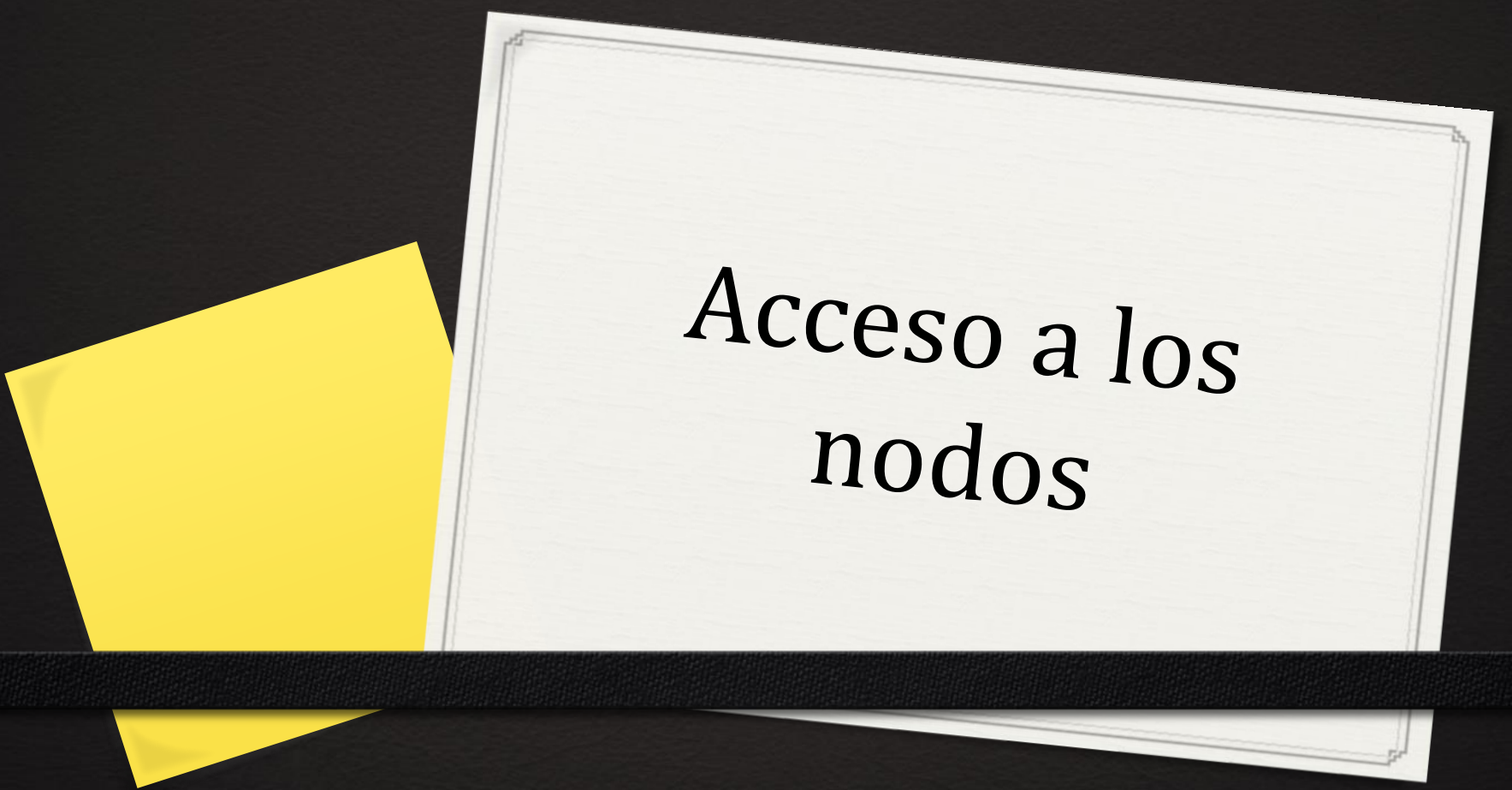
```
</body>
```

```
</html>
```



Tipos de nodos

- La especificación completa de DOM define varios tipos de nodos, aunque las páginas HTML 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 HTML. 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 HTML, es decir, uno por cada par atributo=valor.
 - *Text*, nodo que contiene el texto encerrado por una etiqueta HTML.
 - *Comment*, representa los comentarios incluidos en la página HTML.



Acceso a los
nodos

Acceso a los nodos

- Una vez construido automáticamente el árbol completo de nodos DOM, ya es posible utilizar las funciones DOM para acceder de forma directa a cualquier nodo del árbol.

```
<html>
```

```
<body>
```

```
<p id="demo"></p>
```

```
<script>
```

```
document.getElementById("demo").innerHTML = "¡Hola Mundo!";
```

```
</script>
```

```
</body>
```

```
</html>
```

- En el ejemplo anterior, `getElementById` es un **método**, mientras que `innerHTML` es una **propiedad**
- La propiedad `innerHTML` es útil para obtener o reemplazar el contenido de cualquier elemento HTML (incluidos `<html>` y `<body>`).

getElementsByTagName(nombreEtiqueta)

- o La función anterior obtiene todos los elementos de la página HTML cuya etiqueta sea igual que el parámetro que se le pasa a la función

```
var parrafos = document.getElementsByTagName("p");
```
- o El valor que se indica delante del nombre de la función es el nodo a partir del cual se realiza la búsqueda de los elementos.
 - o Cuando utilizamos la palabra document, nos estamos refiriendo a toda la página web.
- o El valor que va entre paréntesis se refiere al tipo de elemento que queremos obtener (en este caso, hemos puesto p lo que quiere decir que queremos obtener los elementos de tipo párrafo).
- o Es decir, en el ejemplo anterior, estamos indicando que queremos obtener todos los párrafos de la página.
- o 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.
 - o El valor devuelto es un array de nodos DOM, no un array de cadenas de texto o un array de objetos normales.

- o De este modo, se puede obtener el primer párrafo de la página de la siguiente manera:

```
var primerParrafo = parrafos[0];
```

- o De la misma forma, se podrían recorrer todos los párrafos de la página con el siguiente código:

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

- o La función `getElementsByTagName()` se puede aplicar de forma recursiva sobre cada uno de los nodos devueltos por la función. En el siguiente ejemplo, se obtienen todos los enlaces del primer párrafo de la página:

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


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.
- Internet Explorer 6.0 no implementa de forma correcta esta función.

getElementById()

- o La función `getElementById()` es la más utilizada cuando se desarrollan aplicaciones web dinámicas.
 - o Se trata de la función preferida para acceder directamente a un nodo y poder leer o modificar sus propiedades.
 - o La función `getElementById()` devuelve el elemento HTML cuyo atributo `id` coincide con el parámetro indicado en la función.
 - o 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>
```
- o Internet Explorer 6.0 interpreta incorrectamente esta función.



Ejemplos

- o Buscar un elemento por su id
 - o Ejemplo01
- o Buscar un elemento por el nombre de su etiqueta
 - o Ejemplo02
 - o Ejemplo03
- o Buscar un elemento por el nombre de su clase
 - o Ejemplo04
- o Otros elementos DOM
 - o document.anchors
 - o document.body
 - o document.documentElement
 - o document.embeds
 - o document.forms
 - o document.head
 - o document.images
 - o document.links
 - o document.scripts
 - o document.title
 - o Ejemplo05
 - o Ejemplo06
 - o Ejemplo07



Acceso directo a
los atributos
HTML

Acceder a los atributos HTML

- Una vez que se ha accedido a un nodo, el siguiente paso natural consiste en acceder y/o modificar sus atributos y propiedades.
- Mediante DOM, es posible acceder de forma sencilla a todos los atributos HTML y todas las propiedades CSS de cualquier elemento de la página:
 - Para acceder a su valor, simplemente se indica el nombre del atributo HTML detrás del nombre del nodo.
- El ejemplo siguiente obtiene la dirección a la que enlaza el enlace:

```
/* Obtenemos el nodo DOM que representa el enlace*/  
var enlace = document.getElementById("enlace");  
/*Obtenemos el atributo href del enlace con enlace.ref. Si  
quisiéramos obtener, por ejemplo el id, haríamos enlace.id*/  
alert(enlace.href); // muestra http://www.ejemplo.com
```

```
<a id="enlace" href="http://www.ejemplo.com">Enlace</a>
```

Obtener las propiedades CSS

- Las propiedades CSS no son tan fáciles de obtener como los atributos HTML.
- 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:

```
<body>
  
  ...
  <script>
    var imagen = document.getElementById("foto");
    alert(imagen.style.margin);
  </script>
  ...
</body>
```

- o Si el nombre de una propiedad CSS es compuesto, se accede a su valor modificando ligeramente su nombre:

```
var parrafo = document.getElementById("parrafo");  
alert(parrafo.style.fontWeight); // muestra "bold"  
<p id="parrafo" style="font-weight: bold;">...</p>
```

- o La transformación del nombre de las propiedades CSS compuestas consiste en eliminar todos los guiones medios (-) y escribir en mayúscula la letra siguiente a cada guión medio. A continuación se muestran algunos ejemplos:

- o font-weight se transforma en fontWeight
- o line-height se transforma en lineHeight
- o border-top-style se transforma en borderTopStyle
- o list-style-image se transforma en listStyleImage

- El único atributo HTML que no tiene el mismo nombre en HTML y en las propiedades DOM es el atributo `class`.
- Como la palabra `class` está reservada por JavaScript, no es posible utilizarla para acceder al atributo `class` del elemento HTML.
 - En su lugar, DOM utiliza el nombre `className` para acceder al atributo `class` de HTML:

```
var parrafo = document.getElementById("intro");  
alert(parrafo.class); // muestra "undefined"  
alert(parrafo.className); // muestra "normal"
```

```
<p id="intro" class="normal">...</p>
```



Manos a la obra

- A partir de la página web proporcionada y utilizando las funciones DOM, mostrar por pantalla la siguiente información:
 1. Número de enlaces de la página
 2. Dirección a la que enlaza el penúltimo enlace
 3. Numero de enlaces que enlazan a <http://prueba>
 4. Número de enlaces del tercer párrafo
- Crear la página del ejercicio 11
- Completar el código JavaScript proporcionado para que cuando se pinche sobre el enlace se muestre completo el contenido de texto.
 - Al pinchar sobre el enlace, se ejecuta la función llamada `muestra()`.

Ejercicio_10

Ejercicio_12