# DOM: Document Object Model

Miguel Ortuño
Escuela de Ingeniería de Fuenlabrada
Universidad Rey Juan Carlos

Abril de 2025

©2025 Miguel Ortuño
Algunos derechos reservados.
Este trabajo se distribuye bajo la licencia
Creative Commons Attribution Share-Alike 4.0

# DOM: Document Object Model

DOM, Document Object Model es un API que permite procesar una página HTML usando JavaScript. El documento HTML se represente en una estructura con forma de árbol

- Estándar de Internet, aparece en 1998, normalizado por el W3C
- Es el interfaz que emplean los navegadores web internamente
- Cuando un navegador carga una página HTML, la procesa para convertirla en la estructura del DOM. Desde ahí se representa en pantalla.
- Los cambios que se realicen en la página desde JavaScript se hacen directamente en el DOM. El HTML no se vuelve a utilizar

La forma de procesar el DOM ha ido cambiando con los años

- En la década de 2000 se usaba JavaScript *tal cual*, no había alternativa
- En la década de 2010 lo más habitual solía ser emplear la librería jQuery, resultaba más cómodo y conveniente
- JavaScript va mejorando, tomando las mejores ideas de jQuery. En 2020 hay una tendencia clara en el abandono de jQuery y en la vuelta al JavaScript *tal cual*. A veces se le llama *Vanilla JavaScript*
  - La palabra inglesa *vanilla*, literalmente significa *vainilla*. En sentido figurado significa *básico*, *sin adornos* o *convencional* [1]

[1] Este uso aparece en los años 1950, cuando los helados con sabor vainilla (artificial), se convierten en los más habituales por ser los más baratos

# Normalización del DOM y el lenguaje JavaScript

Tanto el lenguaje como el DOM tienen un estándar que los navegadores suelen seguir bastante bien

- En principio, las prácticas de esta asignatura, como cualquier otro programa similar, se podrá ejecutar sin cambios
  - En cualquier navegador web, a menos que sea muy antiguo
  - En cualquier sistema operativo: Linux, Windows, macOS, Android, iOS, iPadOS

Si el resultado es diferente, normalmente se debe a que

- Hay algún error en el código, que distintas plataformas pueden tratar de forma distinta. Por eso es tan importante validar el código
- Usamos alguna característica muy reciente, aún no incluida en el navegador

Hay muchas técnicas en JavaScript (funciones, métodos, atributos, etc) que han quedado obsoletas

- Si están anticuadas no es por moda o capricho: son menos eficientes, menos seguras o más complejas
- Siguen funcionando, para mantener la compatibilidad del código antiguo
- Aunque *funcionen* no debemos usarlas
  - Excepto tal vez en el mantenimiento de código antiguo
- Debemos saber reconocerlas, para evitarlas. Debemos prestar atención a los libros, tutoriales o recetas antiguos
  - Todo es nuevo cuando se escribe, hasta que deja de serlo. Cualquier documento técnico debería incluir su fecha (normalmente basta el año o el mes y el año)

# Modificar el HTML ¿Para qué?

Programando sobre el DOM se puede hacer prácticamente cualquier cosa con una página web

- Normalmente lo que deberíamos buscar es funcionalidad útil que mejore la experiencia de usuario
- Deberíamos evitar los efectos que llamen la atención gratuitamente, comportamiento no estándar o poco intuitivo, adornos que acaban molestando, etc

Funcionalidad que realmente mejora la experiencia de usuario:

- Es normal que una aplicación tenga muchos parámetros, difíciles de asimilar para el usuario
  Ocultar unos y mostrar otros facilita su trabajo
  - Se puede ocultar y/o marcar como deshabilitado lo que en cierto momento no se puede hacer
  - Jerarquizar el interfaz. Por ejemplo modo básico, modo normal, modo experto
- Ofrecer información y ayuda contextual
- Presentar la información en distintos formatos o unidades
- Validación de formularios

- Formularios mejorados
  Ejemplos
  - Una entrada donde el usuario indica un porcentaje desplazando una barra, no introduciendo un número
  - Una entrada que inmediatamente actualiza otra información. *Si gasta 20 entonces le quedan 80*
- Información sobre el progreso de lo que el usuario ha pedido. P.e *progress bar* en porcentaje, o en unidades de tiempo. O estimaciones del tiempo restante
- Información en tiempo real sobre sucesos diversos

- Generación de gráficos *bitmap*
  HTML Canvas.
- Generación de gráficos vectoriales.
  HTML SVG
  SVG: estándar para gráficos vectoriales. Muy extendido, soportado por ejemplo en aplicaciones como Adobe Illustrator o Inkscape
  Se pueden incrustar en el HTML y generar desde javascript.
  La librería más habitual es d3.js

  `https://github.com/d3/d3/wiki/Gallery`
- ...

# Ejecución de un programa Javascript

La ejecución de un programa JavaScript en el navegador tiene dos partes

1. Ejecución secuencial
   Se carga el documento HTML y se ejecutan todos sus scripts, normalmente en el orden en que aparecen en el html [2]. Suele durar décimas de segundo
2. Ejecución asíncrona, dirigida por eventos
   El código empieza a responder a los eventos de usuario (y algunos otros: carga de ficheros, red y errores). Dura todo el tiempo que esté la página web en el navegador

En la primera fase de ejecución secuencial, el código que vaya a ocuparse de algún evento se registra, para que en la segunda fase, cada vez que aparezca el evento, se lance

[2] Aunque esto se puede alterar con async y defer

# Correspondencia HTML - DOM

Todos los elementos de la página HTML que recibe el navegador se reproducen en una estructura análoga en el DOM

- Por cada elemento HTML hay un objeto *Element* en el DOM
- Por cada atributo del elemento HTML, hay una propiedad en el elemento JavaScript

# Objetos Globales

En el DOM hay dos objetos globales muy importantes. Están predefinidos, siempre están presentes en cualquier documento.

1. *Window*
2. *document*

Pero además

- Cualquier otra constante, variable, función o clase que defina el programador, se comparte por todos los scripts y módulos de la misma ventana o pestaña.
  Esta idea es fundamental: aunque tengas varios ficheros `.js`, si están incluidos en el mismo `.html`, es como si fueran un único `.js`
- En otras palabras, todos los scripts de una ventana o pestaña comparten el mismo espacio de nombres

Objeto *Window*

- El código JavaScript que se ejecuta en el navegador tiene un objeto global llamado *Window*
- Hay uno por cada ventana (o pestaña) del navegador, compartido por todos los scripts y todos los módulos (pero no los *WebWorkers*) [3]

Objeto *document*

- Cuando el navegador procesa el HTML, crea un objeto *document*
- Es el objeto principal del DOM, en él se insertan todos los objetos *Element* y todos los nodos de texto

[3] En node.js este objeto se llama global. En los WebWorkers este objeto se llama WorkerGlobalScope

# Eventos

Como hemos visto, una vez cargado y procesado el HTML, el programa JavaScript dentro del navegador entra en la fase dirigida por eventos

- En el contexto del desarrollo de software, un evento es una acción que sucede y que ha de ser tratada por el programa. Normalmente se genera de forma asíncrona y proviene de una fuente externa al propio programa (red, disco, ratón, teclado, reloj, etc)
- Uno de los tipos de eventos más importantes en cualquier programa suelen ser los eventos *de ratón*. Esta es la denominación habitual, pero informal. En rigor no deberíamos hablar de *ratón* sino de *dispositivo señalador* o *dispositivo apuntador*. Esto engloba ratón, *touchpad*, pantalla táctil, *trackball*, etc

Algunas definiciones sobre eventos en JavaScript en el DOM

- ¿Qué pasa?
  *event type*. También llamado *nombre*. Es una cadena de texto que especifica de qué evento se trata. P.e. *click*, *mouseover*, *keydown*, etc
- ¿Dónde pasa?
  *event target* (objetivo). Elemento que recibe el evento. Todo elemento de una página HTML puede recibirlos: botones, párrafos, enlaces, imágenes, formularios, tablas, etc
- ¿Cómo responder?
  *event handler* (manejador). Función que se ejecutará cuando se reciba el evento sobre el *target*
- Más detalles
  *event object*. Objeto con detalles sobre el evento: coordenadas del ratón, tecla pulsada, etc

# Cómo registrar manejadores de eventos

En JavaScript contemporáneo, para registrar un manejador de evento

- Seleccionamos el objeto que recibe el evento con el método `document.querySelector()`
  Su argumento será un selector CSS que especifique el elemento
- Invocamos al método `addEventListener` de este objeto, pasando como parámetros
  - El *event type* (nombre del evento)
  - El manejador
- Un mismo evento puede disparar varios manejadores, basta con llamar varias veces a `addEventListener`. Luego se ejecutarán en el orden en que fueron registrados los manejadores

- `querySelector()` devuelve el primer elemento que encaje en el selector (o null si no encaja ninguno)
- `querySelectorAll` devuelve todos los elementos que encajen en el selector

```
<button id="boton01">Dame un clic</button>
<script>
  'use strict'
  function manej_boton01() {
    console.log("Clic recibido.");
  }

  let b = document.querySelector("#boton01");
  // Atención a la almohadilla, es imprescindible
  b.addEventListener("click", manej_boton01);
  // Atajo para ver los log: F12
  // Atajo alternativo: Ctr Shift I
</script>
```

http://ortuno.es/hola_js_01.html

Importante: en el HTML, primero se define el elemento HTML y *después* el manejador. De lo contrario, el manejador se intentaría registrar en un objeto que aún no existe, esto provocaría un error.

Todo manejador recibe un objeto *event* con los detalles del evento

- Como en el ejemplo anterior, tal vez no lo necesitemos. En ese caso no hace falta declararlo
- En esta versión del ejemplo anterior, sí lo declaramos y lo usamos (para trazarlo)

```
<button id="boton01">Dame un clic</button>
<script>
  'use strict'
  function manej_boton01(event) {
    console.log("Clic recibido.");
    console.log(event);
  }

  let b = document.querySelector("#boton01");
  b.addEventListener("click", manej_boton01);
</script>
```

http://ortuno.es/hola_js_02.html

# Depuración

Para depurar un programa JavaScript en el navegador

- Tendremos siempre abierta la consola de logs del navegador, de lo contrario no veríamos ni siquiera los errores que se puedan producir
  - Atajo de teclado en Google Chrome: `Ctr Shift I`
- Escribiremos trazas con `console.log()`
  - Mientras seamos principiantes, es recomendable añadir trazas continuamente, sin esperar a que el programa falle. Cuando el programa funcione aparentemente bien, podremos quitarlas

Un evento que llega a un objeto *element*, se propaga a todos los antecesores de ese elemento. Ejemplo:

1. Un div tiene una tabla que tiene un botón que tiene una imagen
2. El usuario hace clic sobre la imagen
3. El evento llega a la imagen, al botón, a la tabla y al div

Aunque todos los antecesores reciben el evento, sabremos cuál ha sido el *primero* (en este ejemplo, la imagen) por el atributo *event.target*

# Técnicas obsoletas

En código, recetas y libros antiguos podremos encontrar selección de elementos con métodos como

```
document.getElementById
document.getElementsByName
document.getElementsByTagName(
document.getElementsByClassName
document.images
document.forms
document.links
... etc
```

Incluso ChatGPT, Deepseek, Gemini y Claude (en 2025) usan estas funciones anticuadas.
Debemos evitarlas, en favor de `document.querySelector()` y `document.querySelectorAll()`

Tratamiento obsoleto de eventos:

- Métodos *onclick*, *onmouseover*, *onload* ... muchos otros métodos con nombre `on + event type`
- Manejadores de eventos directamente en el HTML

```
<figure class="zoom" onmousemove="zoom(event)"
 style="background-image: url(//blah.com/image/a.jpg)">
```

Evitaremos estas formas, en favor de `addEventListener()`

# Texto de un elemento

Para acceder al contenido de un elemento en texto plano, cada objeto *element* tiene la propiedad *textContent*, que podemos leer o escribir

```
<button id="boton01">Registrar hora </button>
<p id="parrafo01">--</p>

<script>
  'use strict'
  function manej_boton01(event) {
    let parrafo01 = document.querySelector("#parrafo01");
    let fecha = new Date();
    parrafo01.textContent = fecha;
  }
  let boton01 = document.querySelector("#boton01");
  boton01.addEventListener("click", manej_boton01);
</script>
```

http://ortuno.es/texto.html

# Atributos inexistentes

Recuerda esta característica muy desafortunada de JavaScript

- Si intentamos acceder a una propiedad de un objeto que no exista, el motor no dará ningún error. Simplemente valdrá *undefined*
- P.e. si por error en vez de escribir `p.textContent` escribimos `p.textContext` el motor no nos avisará (al contrario de lo que pasaría en muchos otros lenguajes)

# Trazas

Para el principiante es recomendable poner muchas trazas en la escritura inicial, sin esperar a los errores

- Una traza al comienzo de cada manejador
- Una traza por cada elemento seleccionado
  - Olvidar la almohadilla al seleccionar elementos por su *id* es un error muy frecuente. En este error u otros similares, verás una traza vacía

Naturalmente, trazas tan detalladas acaban siendo molestas

- Bórralas cuando el programa parezca funcionar
- Según vayas adquiriendo experiencia, traza solo aquellos aspectos que te parezca que tengan especial relevancia o dificultad

Todo esto es aplicable a cualquier lenguaje de programación, no solo a JavaScript en el navegador

El ejemplo anterior podrías escribirlo inicialmente así:

```
function manej_boton01(event) {
  console.log("manej_boton01");
  let parrafo01 = document.querySelector("#parrafo01");
  console.log(parrafo01);
  let fecha = new Date();
  console.log("Valor de la fecha:"+fecha);
  parrafo01.textContent = fecha;
}
let boton01 = document.querySelector("#boton01");
console.log(boton01);
boton01.addEventListener("click", manej_boton01);
```

http://ortuno.es/texto.trazas.html

# Separación lógica de negocio / interface de usuario

- Es fundamental tener en cuenta que estamos tratando solo con el interface de usuario.
- La lógica de negocio de la aplicación debe estar bien diferenciada, en un código independiente que desarrollaremos y probaremos por separado
  - Excepto tal vez si se trata de funcionalidad trivial

Esto es un ejemplo de lo que **nunca deberíamos hacer**

```
function manej_boton02(event){
    console.log("manej_boton02");
    let div_v_in = document.querySelector("#v_in");
    let v_in = div_v_in.textContent;
    let v_out=v_in*3.6+"Km/h";    //  ¡MUY MAL!

    console.log(v_out);
    let v_out_div = document.querySelector("#v_out");
    v_out_div.textContent = v_out;
}
```

http://ortuno.es/calculo_mal.html

El ejemplo anterior en principio *funciona*, pero

- Mezcla continuamente la programación de botones y displays con los cálculos propios de la aplicación
- Por ser un ejemplo *de juguete* podría pensarse que no tiene importancia
- Pero en un programa real esto acabará dando muchos problemas. Cualquir retoque, corrección o ampliación será mucho más complejo porque afectará a todo el programa en su conjunto

Este tipo de manejadores deben limitarse a

- Leer los valores de entrada desde el HTML
- Llamar al código que realiza los cálculos
- Escribir los valores de salida en el HTML

Enfoque correcto:

```
let v_in = document.querySelector("#v_in").textContent;
let v_out = calcula_velocidad(v_in,"Km/h");

let display = document.querySelector("#v_out");
display.textContent = v_out;
```

# Variables globales

Observa el siguiente ejemplo

```
<body>
  <script>
    let x = 0; // Variable global a todo el documento
  </script>
  <div id="display01"></div>
  <br>
  <button id="boton01">Suma 1</button>
  <button id="boton02">Suma 5</button>
  <script>
    'use strict'
    function actualiza_valor() {
      let div = document.querySelector("#display01");
      div.textContent = x;
    }
  </script>
```

```
  <script>
    use 'strict'
    function manej_boton01() {
      x = x + 1;
      actualiza_valor();
    }

    function manej_boton02() {
      x = x + 5;
      actualiza_valor();
    }

    let boton01 = document.querySelector("#boton01");
    boton01.addEventListener("click", manej_boton01);
    let boton02 = document.querySelector("#boton02");
    boton02.addEventListener("click", manej_boton02);
  </script>
</body>
```

http://ortuno.es/globales.html

Aqui la *lógica de negocio* es tan sencilla que, excepcionalmente, la hemos dejado en el manejador

- La comunicación de un valor entre los manejadores la hacemos con una variable global[4].
  - Error frecuente de principante: declarar la variable dentro de una función. Entonces ya no es global y no funciona nada
- Las variables globales exigen mucha atención, son muy peligrosas. Algunas metodologías las prohíben, el resto, exige minimizarlas
- No deberíamos tener muchas variables globales por separado. Es preferible un único / unos pocos objetos globales, con los atributos necesarios
- Recuerda que estas variable son **única y exclusivamente** para comunicar valores entre manejadores

[4] Sería mejor hacerlo mediante cierres (closures), pero es algo más complejo y no lo trataremos aquí

# Creación dinámica de elementos

Con el método *textContent*, hemos visto cómo modificar el texto de un elemento preexistente
Para crear elementos dinámicamente y añadirles cualquier contenido

- `document.createElement()`
  - Recibe como argumento una cadena con el nombre de un tipo de elemento HTML (p, img, table, etc)
  - Devuelve un elemento de ese tipo
- Con el método *append* de un elemento, añadimos otro elemento en su interior

```
<body>
  <button id="boton01">Registrar hora </button>
  <div id="div01">--</div>

  <script>
    'use strict'
    function manej_boton01(event){
        let div01 = document.querySelector("#div01");
        let fecha = new Date();
        let parrafo = document.createElement("p");
        parrafo.textContent = fecha;
        div01.append(parrafo);
    }
    let boton01 = document.querySelector("#boton01");
    boton01.addEventListener("click", manej_boton01);

  </script>
</body>
```

http://ortuno.es/nuevo_p.html

# Diferencia textContent append

- textContent
  Es una propiedad

```
parrafo.textContent = fecha;
```

  *Reemplaza* (machaca el valor previo). Solo sirve para texto

- append
  Es un método

```
div01.append(parrafo);
```

  *Añade* cualquier cosa al elemento: texto, una fila, una imagen, un enlace...

En ocasiones puede resultar equivalente usar uno o usar otro. P.e. meter texto en un párrafo vacio o en un texto vacio

```
<button id="boton01">append</button>
<button id="boton02">textContent</button>
<br>
<div id="div01"></>
<script>
  'use strict'
  function usa_append(event){
      let div = document.querySelector("#div01");
      div.append("hola");
  }

  function usa_textContent(event){
      let div = document.querySelector("#div01");
      div.textContent = "hola";
  }

  let boton01 = document.querySelector("#boton01");
  boton01.addEventListener("click", usa_append);

  let boton02 = document.querySelector("#boton02");
  boton02.addEventListener("click", usa_textContent);
</script>
```

http://ortuno.es/append_textContent.html

# Insertar una fila en una tabla

Hemos visto cómo insertar un párrafo. De la misma forma, podemos inserta cualquier elemento en cualquier elemento. P.e una fila en una tabla

```
<table id=tabla_horas>
  <tr>
  <th>Hora del clic</th>
  </tr>
</table>
<button id="boton01">Registrar hora </button>
```

```
<script>
  'use strict'
  function manej_boton01(event){
      // Seleccionamos la tabla
      let tabla_horas = document.querySelector("#tabla_horas");

      let tr = document.createElement("tr");  // Creamos la fila

      let td = document.createElement("td");  // Creamos la celda
      let fecha = new Date();
      td.append(fecha); // Metemos la fecha en la celda

      tr.append(td);    // Metemos la celda en la fila
      tabla_horas.append(tr);  // Metemos fila en la tabla
  }
  let boton01 = document.querySelector("#boton01");
  boton01.addEventListener("click", manej_boton01);
</script>
```

http://ortuno.es/nueva_fila.html

# O dos celdas en una fila

```
let contador_clics = 0;
function manej_boton01(event){
    let tabla_horas = document.querySelector("#tabla_horas");

    contador_clics = contador_clics + 1;
    let tr = document.createElement("tr"); // Creamos fila

    let td = document.createElement("td"); // Creamos celda
    td.append(contador_clics);  // Metemos contador en celda
    tr.append(td);  // Añadimos celda a fila

    td = document.createElement("td"); // Creamos celda
    let fecha = new Date();
    td.append(fecha);  // Metemos fecha en celda
    tr.append(td);  // Metemos celda en fila

    tabla_horas.append(tr); // Metemos fila en tabla
}
let boton01 = document.querySelector("#boton01");
boton01.addEventListener("click", manej_boton01);
```

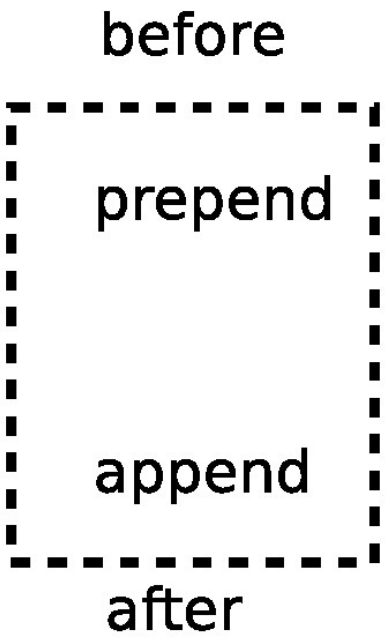
http://ortuno.es/nueva_fila_celdas.html

# Añadir contenido

Una vez seleccionado un elemento o elementos mediante un selector, insertarlo en diferentes posiciones

- append()
  Inserta contenido al final de la selección, dentro de la selección
- prepend()
  Inserta contenido al principio de la selección, dentro de la selección
- after()
  Inserta contenido inmediatamente después de la selección, fuera de la selección
- before()
  Inserta contenido inmediatamente antes de la selección, fuera de la selección

En esta figura representamos con una caja punteada el elemento señalado




- *before()* escribe inmediatamente antes
- *prepend()* escribe dentro, al principio
- *append()* escribe dentro, al final
- *after()* escribe inmediatamente después

También cambiarlo o eliminarlo

- replacewith()
  Reemplaza la selección
- remove()
  Borra la selección

# Creación de una imagen

Con `createElement` podemos crear cualquier elemento. Y si necesita atributos, también los añadiremos dinámicamente. Por ejemplo en una imagen, los atributos *src*, *alt*
Para construir dinámicamente algo como esto

```
<img src="images/gato.jpg" alt="Gato común europeo de color naranja">
```

por cada atributo en el elemento HTML hay una propiedad en el objeto element correspondiente en JavaScript, que casi siempre tiene el mismo nombre

```
<img src="images/gato.jpg" alt="Gato común europeo de color naranja">
        let img = document.createElement("img");  // Creamos la imagen
        img.src = "images/gato.jpg";
        img.alt = "Gato común europeo de color naranja";
```

Para especificar dónde está el fichero con la imagen, recuerda lo visto en el tema de HTML sobre el fichero especificado en

`<img src=...>`

(o cualquier otra referencia a un fichero)

- Un trayecto que no empieza por barra es relativo al directorio actual
  `images/gato.jpg`
- Un trayecto que empieza por barra es absoluto, cuelga del directorio raiz
  `/images/gato.jpg`
- Un trayecto absoluto que incluye el directorio *home* del usuario como una cadena literal, es un error serio
  `/home/alumnos/jperez/images/gato.jpg`

En algunos casos la propiedad en el objeto element no se llama igual que el atributo en el elemento HTML, por ser una palabra reservada en JavaScript

- El atributo *for* del elemento HTML `<label>` se corresponde con la propiedad `htmlFor` del objeto JavaScript
- El atributo *class* de cualquier elemento HTML se corresponde con la propiedad `className` del objeto JavaScript

En HTML los atributos siempre son de tipo cadena, en JavaScript se convierten a tipo booleano o cadena, si corresponde

```
<button id="boton01">Crear un gato</button>
<div id=div01>
</div>
<script>
  'use strict'
  function crear_gato(event){
      let div01 = document.querySelector("#div01");

      let img = document.createElement("img");  // Creamos la imagen
      img.src = "images/gato.jpg";
      img.alt = "Gato común europeo de color naranja";
      img.width="300";
      div01.append(img); // Metemos la imagen en el div
  }
  let boton01 = document.querySelector("#boton01");
  boton01.addEventListener("click", crear_gato);
</script>
```

http://ortuno.es/crea_imagen.html

# Identificación de un elemento seleccionado

La propiedad `event.target` también puede ser útil para saber qué elemento ha recibido un click

- Recuerda que cualquier elemento puede recibir el evento *click*, no hace falta que sea un botón
- Podemos identificar el elemento que recibe el clic con `event.target.id`
- Naturalmente, será necesario que el elemento tenga un atributo *id*, que se lo habremos añadido o bien de forma estática (en el HTML) o bien de forma dinámica (programándolo en JavaScript)

En el siguiente ejemplo, cuando creamos una imagen

- Le añadimos un *id*
- Le añadimos un manejador para que cada vez que la imagen reciba un click, dispare cierta función
- Esa función identifica la imagen a través de `event.target.id`

```
let contador_gatos = 0;
function crear_gato(){
    let div01 = document.querySelector("#div01");

    let img = document.createElement("img");  // Creamos la imagen
    img.src = "images/gato.jpg";
    img.alt = "Gato común europeo de color naranja";
    img.width="300";

    // Añadimos un id a la imagen
    contador_gatos = contador_gatos + 1;
    let id = "gato_" + String(contador_gatos) ;
    img.id = id;

    // Añadimos un manejador al evento click sobre la imagen
    img.addEventListener("click", identifica_foto)
    div01.append(img); // Metemos la imagen en el div
}
```

Observa que *contador_gatos* tenemos que declararla necesariamente fuera de la función *crear_gato()*

```
function identifica_foto(event){
    let id_foto = event.target.id;
    let span01 = document.querySelector("#span01");
    span01.textContent = id_foto;
}
```

http://ortuno.es/identifica_imagen.html

- En todos los ejemplos anteriores, cada objeto en particular tenía su propio manejador. Escribíamos tantos manejadores como objetos (estáticos).
- Esta técnica es más avanzada, exactamente el mismo manejador, *identifica_foto()* se dispara para muchos objetos distintos, creados dinámicamente.
- El manejador sabe *quién* ha recibido el evento gracias a la propiedad *event.target.id*.

# Modificación de un elemento

Podemos cambiar cualquier elemento dinámicamente, modificando sus atributos

```
<script>
  'use strict'
  pon_gato(); // Empezamos poniendo una imagen para que no quede
              // en la página una foto vacía, incorrecta

  function pon_gato(event){
      let img = document.querySelector("#foto");
      img.src = "images/gato.jpg";
      img.alt = "Gato común europeo de color naranja";
      img.width="300";
  }

  function pon_periquito(event){
      let img = document.querySelector("#foto");
      img.src = "images/periquito.jpg";
      img.alt = "Periquito azul";
      img.width="300";
  }
</script>
```

```
<script>
  let boton01 = document.querySelector("#boton01");
  boton01.addEventListener("click", pon_gato);

  let boton02 = document.querySelector("#boton02");
  boton02.addEventListener("click", pon_periquito);
</script>
```

http://ortuno.es/cambia_imagen.html

Es frecuente que queramos cambiar dinámicamente el aspecto de un documento HTML, esto es, sus atributos CSS. P.e. quitar o poner el valor *none* al atributo *display*
Aunque se puede modificar directamente un atributo CSS, normalmente es preferible

- Crear una regla CSS que modifique el atributo para cierta clase
- Quitar y poner la clase [5]

```
<style>
  .oculto {
    display: none;
  }
</style>
```

[5] De la misma forma que en un procesador de texto podemos modificar el formato de un párrafo directamente, pero en general es preferible asignar un estilo al párrafo, y modificar el formato del estilo

```
<button id="boton01">Ver foto</button>
<button id="boton02">Quitar foto</button>
<div id="marco_foto" class="oculto">
  <img src="images/plaza_espana.jpg" alt="Plaza de España, Madrid">
</div>
<script>
  'use strict'
  function quita_clase() {
      console.log("quita_clase");
      let marco = document.querySelector('#marco_foto');
      marco.classList.remove("oculto");
  };

  function pon_clase() {
      console.log("pon_clase");
      let marco = document.querySelector('#marco_foto');
      marco.classList.add("oculto");
  };

  let boton01 = document.querySelector("#boton01");
  boton01.addEventListener("click", quita_clase);
  let boton02 = document.querySelector("#boton02");
  boton02.addEventListener("click", pon_clase);
</script>
```

http://ortuno.es/quita_pon_01.html

En el ejemplo anterior, lo que buscamos es alternar entre esto

```
<div id="marco_foto" class="oculto">
  <img src="images/plaza_espana.jpg" alt="Plaza de España, Madrid">
</div>
```

y esto

```
<div id="marco_foto">
  <img src="images/plaza_espana.jpg" alt="Plaza de España, Madrid">
</div>
```

- Esto es, añadir y quitar la clase *oculto* al div cuyo identificador es *marco_foto*
- Esto es, añadir y quitar el atributo `class` con el valor `oculto`

Desde el punto de vista sintático, *class* es un atributo HTML como otro cualquiera, pero tiene algunas características que suelen despistar al principiante

- Posiblemente el nombre *class* es poco afortunado. Sería más coherente *classes*, porque es una lista de elementos
- Como hemos visto, el atributo `class` se corresponde con la propiedad JavaScript `className`, por ser *class* una palabra reservada

- Podemos quitar y poner clases manejando directamente la cadena almacenada en `className`
  Ejemplo. Para alternar entre estas dos versiones de un párrafo

```
<p class="noticia oculto deportes"> tal tal </p>
<p class="noticia deportes"> tal tal </p>
```

  Podríamos programar algo así

```
parrafo1.className = "noticia oculto deportes"
parrafo1.className = "noticia deportes"
```

- Aunque lo anterior es perfectamente válido, en el DOM podemos usar la propiedad `classList`, lo que resulta más cómodo

En el DOM, `classList` es una propiedad de los objetos elemento que devuelve una lista de las clases del elemento

- Esto es, de los valores del atributo *class*, troceados por espacios

En esta lista podemos usar los métodos `add()`, `remove()`, `contains()` y `toggle()`
De esta forma,

- Con el código

```
marco.classList.add("oculto");
```

conseguimos algo equivalente a

```
<div id="marco_foto" class="oculto">
```

- Con el código

```
marco.classList.remove("oculto");
```

conseguimos algo equivalente a

```
<div id="marco_foto">
```

Con el método `toggle` (alternar) de la lista de atributos

- Si el atributo está incluido en la lista, lo borramos
- Si no está incluido, lo añadimos

```
function alterna_clase() {
    let marco = document.querySelector('#marco_foto');
    marco.classList.toggle("oculto");
};

let boton01 = document.querySelector("#boton01");
boton01.addEventListener("click", alterna_clase);
```

http://ortuno.es/quita_pon_02.html

# querySelectorAll

En todos los ejemplos anteriores, seleccionábamos un elemento del DOM (y solo uno) por su identificador. Pero habrá ocasiones en las que necesitamos seleccionar una serie de elementos.
Para ello disponemos del método `document.querySelectorAll`

- No es exactamente un array, sino un objeto similar, *NodeList*
- Si esta lista está vacía, su atributo *length* valdrá 0

En este ejemplo, seleccionamos todos elementos que contengan la clase *gato*, para eliminarlos

```
function borra_gatos(event){
    let gatos = document.querySelectorAll(".gato");
    for (let gato of gatos){
        gato.remove();
    }
}
```

http://ortuno.es/elimina.html

Hay muchas formas de seleccionar una serie de elementos:

- Todos los que tengan cierto atributo, con cierto valor, p.e. el fichero de una imagen
- Todos los que tengan su nombre en una lista
- Todos los que tengan un identificador siguiendo cierto patrón
- ...

Pero en general lo recomendable será seleccionar un elemento por una clase

- Para ello, naturalmente, a la hora de crearlo, nos habremos encargado de añadirle la clase

# display / visibility

Para hacer un elemento invisible, puedo usar

- `display`
  Oculta el elemento y reposiciona el resto
- `visibility`
  Oculta el elemento, sin reposicionar el resto

```
<style>
  .sin_display {
    display: none;
  }
  .invisible {
    visibility: hidden;
  }
</style>
...
<script>
  for (let gato of gatos){
      if (usa_display)
          gato.classList.toggle("sin_display");
      else
          gato.classList.toggle("invisible");
      }
</script>
```

http://ortuno.es/selecciona.html

# Modificar el CSS directamente

Para modificar los atributos css, normalmente es preferible vincularlos a una clase y poner y quitar clases, como acabamos de ver

- Pero en ocasiones puede ser conveniente modificar los atributos directamente. Cada elemento del DOM tiene una propiedad *style* que a su vez contiene un objeto cuyos atributos son los atributos CSS.
- Con una modificación: los atributos CSS suelen incluir guiones, que no son válidos en JavaScript (se interpretarían como el operador de resta). Así que hay que convertir a notaciónDromedario[6]
  P.e. `border-width` sería `style.borderWidth`

[6] En español tenemos la NotaciónCamello y la notaciónDromedario, pero en inglés es diferente: PascalCase, camelCase

```
function manej_boton01(event) {
  let p01 = document.querySelector("#span01");
  p01.style.backgroundColor="LightSkyBlue";
}

function manej_boton02(event) {
  let p01 = document.querySelector("#span01");
  p01.style.backgroundColor="LightSalmon";
}
```

http://ortuno.es/css_js.html

# Eventos de ratón

- *mouseover*
  Se recibe cuando el ratón se coloca sobre el elemento
- *mouseout*
  Se recibe cuando el ratón abandona el elemento o cualquiera de sus descendentes
- *mouseleave*
  Se recibe cuando el ratón abandona el elemento seleccionado

# Eventos mouseout, mouseleave

Podemos observar la diferencia entre *mouseout* y *mouseleave* en esta demo. No usa JavaScript contemporáneo sino jQuery, pero podemos seguir el funcionamiento general

`https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_event_mouseleave_mouseout`

En este ejemplo tenemos:

- Un primer *div* de clase *over* que contiene un texto que contiene un *span*
  - A este div se le pone un manejador para el evento *mouseout*
  - Cada vez que el *ratón* abandona el div o que el ratón abandona uno de sus descendientes (el texto), se dispara el manejador y se incrementa el contador
- Un segundo *div* de clase *enter* que contiene un texto que contiene un *span*
  - A este div se le pone un manejador para el evento *mouseleave*
  - Cada vez que el ratón abandona el div, se dispara el manejador. Pero cuando abandona sus descendientes, no

# Eventos mouseout, mouseleave (ii)

Observaciones adicionales:
El autor del ejemplo de *w3schools* tiene un estilo de programación diferente al que empleamos en la asignatura. Debemos entenderlo aunque no lo usemos

- Al primer div le pone la clase *over*, al segundo, la clase *enter*
- Para capturar el primer div emplea el selector `div.over`, para el segundo, `div.enter`. Esto es: *el (los) div de clase over, el (los) div de clase enter*
- Para capturar los span emplea los selectores `.over span` y `.enter span`. Esto es: el (los) span que están dentro de un elemento de clase over y el (los) span que están dentro de un elemento de clase enter

En nuestos ejemplos, habríamos usado un id para cada div y para cada span

El ejemplo de *w3schools* también emplea otra técnica que aquí desaconsejamos: usar un valor almacenado en el interface de usuario

- Para incrementar los contadores, captura el valor que hay dentro del *span* y le suma 1. Lo hace en jQuery, en JavaScript contemporáneo usaríamos *textContent*
- El diseño que preferimos aquí emplearía una variable distinta para estos contadores. El span se limitaría a copiar esa variable

En este ejemplo añadimos manejadores de eventos de ratón a un elemento ya creado. Exactamente igual que en todos los ejemplos anteriores

```
function manej01(event) {
  event.target.classList.add("destacado");
}

function manej02(event) {
  event.target.classList.remove("destacado");
}

let p01 = document.querySelector("#p01");
p01.addEventListener("mouseover", manej01);
p01.addEventListener("mouseout", manej02);
```

http://ortuno.es/eventos_01.html

También podemos añadir los manejadores al crear elementos dinámicamente

```
function crea_parrafos(){
  for(let i=0; i<3; ++i){
    let div01 = document.querySelector("#div01");
    let p = document.createElement("p");
    p.textContent = "Lorem ipsum dolor sit amet";
    p.addEventListener("mouseover", manej01);
    p.addEventListener("mouseout", manej02);
    div01.append(p);
  }
}
```

http://ortuno.es/eventos_02.html

Pero si necesitamos un mismo manejador sobre una serie de elementos, no es necesario añadirlo uno a uno a todos ellos

- Si los elementos los creámos dinámicamente esto no debería ser problema, pero si son elementos preexistente, puede resultar un poco pesado

Recuerda que un evento que llega a un elemento, también llega a todos sus antecesores. Podemos:

- Poner en el manejador en un antecesor común (llamémosle *abuelo*)
- Identificar en el *abuelo* al *nieto* concreto donde se disparó el evento, con *event.target*. De esta forma, el manejador está en el *abuelo* pero la acción se realiza en el *nieto*

```
<div id=div01>
  <p id=p01>Lorem ipsum dolor sit amet.</p>
  <p id=p02>Lorem ipsum dolor sit amet.</p>
  <p id=p03>Lorem ipsum dolor sit amet.</p>
</div>
<script>
  function manej01(event) {
    event.target.classList.add("destacado");
  }

  function manej02(event) {
    event.target.classList.remove("destacado");
  }

  let div = document.querySelector("#div01");
  div.addEventListener("mouseover", manej01);
  div.addEventListener("mouseout", manej02);
</script>
```

http://ortuno.es/eventos_03.html

# Formularios (1)

Para acceder al valor de un formulario

- Añadimos un manejador para el evento *change*
- Leemos el atributo *value* del objeto *element* correspondiente

```
<form action="/action_page.html" id=formulario01>
  <label for="usuario">Usuario</label>
  <input type="text" name="usuario" id="usuario">
</form>
<script>
  function manej01(event) {
    let usuario = document.querySelector("#usuario");
    console.log(usuario.value);
  };
  let formulario01 = document.querySelector("#formulario01");
  formulario01.addEventListener("change", manej01);
</script>
```

http://ortuno.es/formulario_01.html

# Formularios (2)

Para consultar el estado de un *radiobutton* o un *checkbox*, usamos un selector como

```
input[name=figura]:checked
```

Captura todos los *input* que contengan el atributo *name* con el valor *figura*
El selector `:checked` captura todos los `<input>` activados

```
<form id=formulario01 action="/action_page.html">
  <input type="radio" name="figura" value="sota">Sota
  <input type="radio" name="figura" value="caballo">Caballo
  <input type="radio" name="figura" value="rey">Rey
</form>
<script>
  'use strict'
  function manej01(event) {
    let input = document.querySelector("input[name=figura]:checked");
    console.log(input.value);
  };
  let formulario01 = document.querySelector("#formulario01");
  formulario01.addEventListener("change", manej01);
</script>
```

http://ortuno.es/formulario_02.html

# Validación de un formulario

Validar un formulario consiste en comprobar que el usuario ha rellenado los campos con valores adecuados. Esto podemos hacerlo

- Con métodos específicos de HTML
- Con código JavaScript, como el siguiente ejemplo:
  Cuando el usuario acabe de escribir el valor para la contraseña (cuando cambie el foco a otro *input* o pulse *enviar*), se disparará el evento *change* que disparará el manejador para comprobar que la contraseña tenga la longitud mínima exigida

```
let contrasenia_minima = 8;

function valida_contrasenia(contrasenia){
  let texto;
  if (contrasenia.length >= contrasenia_minima) {
    texto = "Contraseña aceptable";
  } else {
    texto = "Contraseña muy corta";
  }
  return texto;
}

function manej01(event) {
  let contrasenia = document.querySelector("#contrasenia").value;
  let texto_validacion = valida_contrasenia(contrasenia);

  let display = document.querySelector("#validacion");
  display.textContent = texto_validacion;
};

let campo_contra = document.querySelector("#contrasenia");
campo_contra.addEventListener("change", manej01);
```

http://ortuno.es/validacion_01.html

Podemos mejorar el ejemplo anterior, de forma que no sea necesario que el usuario acabe de editar un campo para obtener realimentación. Para ello capturamos los eventos

- `Change`
  Fin de edición del *input*
- `keyup`
  Pulsación de teclado (liberación de la tecla pulsada)
- `paste`
  Pegado desde el portapapeles
- `mouseup`
  Pulsación del botón del ratón (fin de la pulsación)

```
function manej01(event) {
  let contrasenia = document.querySelector("#contrasenia").value;
  let texto_validacion = valida_contrasenia(contrasenia);

  let display = document.querySelector("#validacion");
  display.textContent = texto_validacion;
};
let campo_contra = document.querySelector("#contrasenia");
campo_contra.addEventListener("change", manej01);
campo_contra.addEventListener("keyup", manej01);
campo_contra.addEventListener("paste", manej01);
campo_contra.addEventListener("mouseup", manej01);
```

http://ortuno.es/validacion_02.html