

©2022 GSyC
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 tratar una página web con estructura de árbol

- Estándar de Internet, 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 pueda tener la página, p.e. desde JavaScript, se hacen directamente con el DOM, el HTML no se vuelve a utilizar

jQuery

jQuery es una librería JavaScript que permite recorrer un documento, seleccionar objetos del DOM, hacer animaciones, manejar eventos, usar Ajax y hacer plugins sobre JavaScript

- Creada por John Resig en 2006, es software libre
- En la década de 2010, aunque se puede manejar el DOM directamente desde JavaScript, es más habitual y conveniente emplear jQuery. Pero JavaScript mejora paulatinamente, a principios de la década de 2020 muchos proyectos abandonan jQuery para volver a usar JavaScript nativo (moderno)
- También cuenta con una librería similar a Bootstrap, jQuery-UI
 - En esta asignatura preferimos Bootstrap a jQuery-UI
 - Es frecuente preferir Bootstrap. jQuery-UI no es tan popular como Bootstrap

Modificar el HTML ¿Para qué?

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

- 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. (No lo trataremos en esta asignatura)
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>
- ...

Uso de jQuery

Es una librería compuesta por un único fichero

- Se puede descargar en el sistema de ficheros local y cargarla con

```
<script src="jquery.js"></script>
```

- Se puede usar un CDN. Por ejemplo el de gogle

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
```

jQuery es una única función, que a su vez incluye diversas funciones

- Esto es posible porque JavaScript usa funciones de orden superior, *higher-order functions*. Nos puede ayudar considerarlo un objeto con métodos (en JavaScript las funciones son objetos)
- Esta función normalmente recibe el nombre de \$
Típicamente recibe como primer argumento un selector CSS que indica sobre qué debe actuar

Añadir y quitar clases

Un patrón habitual en jQuery es

- Definir una clase CSS con el aspecto deseado. Por ejemplo ocultar un elemento
- Invocar los métodos `removeClass` y `addClass` para añadir y quitar esa clase cuando se produzca cierto evento

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

[...]

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

```
<script>
  function mostrar_marcofoto() {
    $("#marco_foto").removeClass("oculto")
  };

  function ocultar_marcofoto() {
    $("#marco_foto").addClass("oculto")
  };

  function main() {
    $("#boton01").click(mostrar_marcofoto);
    $("#boton02").click(ocultar_marcofoto);
  };

  $(document).ready(main());
</script>
```

La función `$(document).ready()`; acepta la función principal que se ejecutará cuando acabe de cargarse el documento HTML

http://ortuno.es/hola_jquery01.html

El ejemplo anterior es correcto pero no es idiomático. Para asignar un manejador a un evento, lo habitual en JavaScript es usar funciones anónimas

```
$(document).ready(function() {  
    $("#boton01").click(function() {  
        $("#marco_foto").removeClass("oculto")  
    });  
  
    $("#boton02").click(function() {  
        $("#marco_foto").addClass("oculto")  
    });  
});
```

http://ortuno.es/hola_jquery02.html

Esto asigna un manejador al evento *ready* que a su vez se encarga de asignar manejadores al evento *click* de los botones

Se puede añadir más de un manejador a un evento, basta llamar varias veces a la función correspondiente.

El siguiente ejemplo es equivalente al anterior

```
$(document).ready(function() {  
    $("#boton01").click(function() {  
        $("#marco_foto").removeClass("oculto")  
    });  
});  
  
$(document).ready(function() {  
    $("#boton02").click(function() {  
        $("#marco_foto").addClass("oculto")  
    });  
});
```

http://ortuno.es/hola_jquery03.html

Esto asigna un primer manejador al evento *ready* que asigna el manejador de un botón, y un segundo manejador para el mismo evento *ready*, que asigna el manejador del segundo botón

Quitar y poner una clase cuando se recibe un evento es muy común, así que hay una función para ello: `toggleClass()`

```
$(document).ready(function() {  
    $("#boton01").click(function() {  
        $("#marco_foto").toggleClass("oculto")  
    });  
});
```

<http://ortuno.es/toggle.html>

Aunque normalmente es preferible quitar y poner clases, también es posible modificar las propiedades css directamente

```
$(document).ready(function() {  
  $("span").mouseover(function() {  
    $(this).css("background-color", "grey");  
  });  
  
  $("span").mouseleave(function() {  
    $(this).css("background-color", "white");  
  });  
});
```

Eventos mouseover, mouseleave

- Cuando el ratón se coloca sobre cierto elemento, este recibe el evento *mouseover*
Para tratarlo, le pasamos a una función con el mismo nombre (`mouseover()`) el manejador, esto es, la función que se invocará cuando se reciba el evento
- De la misma forma, la retirada del ratón de un elemento es *mouseleave*

Ejemplo: Destaquemos el párrafo sobre el que se posiciona el ratón

```
.destacado {  
  background-color: LightGrey;  
}
```

```
$(document).ready(function() {  
  $("p").mouseover(function() {  
    $("p").addClass("destacado")  
  });  
  
  $("p").mouseleave(function() {  
    $("p").removeClass("destacado")  
  });  
});
```

http://ortuno.es/eventos_ratón01.html

Esto tiene un problema: no estamos destacando el párrafo seleccionado, sino todos los párrafos

Para añadir el manejador al elemento que nos ha devuelto la consulta anterior, usamos `this`

```
$(document).ready(function() {  
    $("p").mouseover(function() {  
        $(this).addClass("destacado");  
    });  
  
    $("p").mouseleave(function() {  
        $(this).removeClass("destacado");  
    });  
});
```

http://ortuno.es/eventos_ratón02.html

Modificar el texto de un elemento

Con el método `text()` podemos cambiar el texto de un elemento¹
Como ejemplo, cada vez que se pulse un botón escribiremos la hora actual dentro de un DIV

La hora estará en formato ISO 8651, para ello usaremos *moment*, una librería muy popular para procesar fechas

- Podemos descargarla desde su sitio web
`https://momentjs.com` e incluirla en nuestro directorio
- Podemos descargarla desde un CDN y añadirlo en el atributo `src` de un elemento `script`
`https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.19.1/moment-with-locales.min.js`

`moment().format()` devuelve la fecha actual en diversos formatos. Si no se indica ninguno, por omisión devuelve una cadena ISO 8651, p.e.

2017-11-11T22:21:17+01:00

¹En su argumento no puede haber html, sería eliminado

```
<button id="boton01">Registrar hora </button>
<p id="text01">--</p>
<script>
  $(document).ready(function() {
    $("#boton01").click(function() {
      $("#text01").text(moment().format());
    });
  });
</script>
```

<http://ortuno.es/text.html>

Leer el texto de un elemento

El método `text()` de jQuery devuelve el texto de un elemento

Esto es un patrón común en los método de jQuery:

- Invocados sin argumentos, devuelven un valor
- Invocados con argumentos, modifican un valor

Ejemplo: leer una velocidad en m/s desde un elemento

`<div id="v_in">`

y escribirla expresada en km/h en un `<div id="v_out">`

```
$("#boton02").click(function() {  
    let v_text=$("#v_in").text();  
    let v=Number(v_text)  
    let v_out=v*3.6+"Km/h";  
    $("#v_out").text(v_out);  
});
```

<http://ortuno.es/text2.html>

Este programa es un ejemplo de diseño muy deficiente: lógica de negocio repartida por los botones

Un diseño mucho más razonable es el que usamos en las prácticas:

```
$("#boton02").click(function() {  
    let v_text=$("#v_in").text();  
    let v_out=calcula_velocidad(v_text, "m/s", "km/h");  
    $("#v_out").text(v_out);  
});
```

Variables globales

Recuerda que las variables globales se comparten entre todos los scripts del mismo documento HTML

```
<script>
  let x = 0; // Variable global a todo el documento
</script>
[...]
<script>
  $(document).ready(function() {
    $("#boton01").click(function() {
      x = x + 1;
      $("#display01").text(String(x));
    });
    $("#boton02").click(function() {
      x = x + 5;
      $("#display01").text(String(x));
    });
  });
</script>
```

Por supuesto, este código también podría estar en ficheros .js aparte

http://ortuno.es/variables_globales.html

- En cualquier lenguaje de programación, las variables globales son peligrosas. Según la metodología que sigamos o bien minimizaremos su uso o bien las suprimiremos por completo
- En el caso de javascript en el navegador, es aún más peligroso, por ser variables que se comparten jentre varios ficheros! (todos los incluidos en el HTML)
- Pero suele ser aceptable usar un único objeto global, con distintos atributos, para propiedades que sean claramente comunes a toda la página

Por otro lado, observa que los manejadores no pueden recibir parámetros. El siguiente ejemplo es incorrecto

```
let x = 0;
function suma(a){
    console.log("invocando funcion suma. x:", x, "a:",a);
    x = x + a;
    $("#display01").text(String(x));
}
$(document).ready(function() {
    // ¡MAL! No funciona
    $("#boton01").click(suma(1));
    $("#boton02").click(suma(5));
});
```

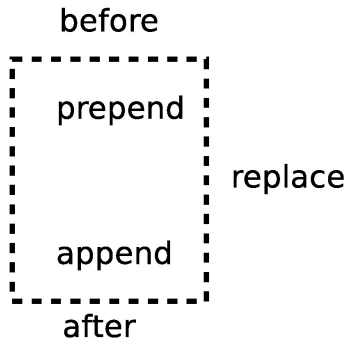
http://ortuno.es/no_parametros.html

Añadir contenido

Una vez seleccionado un elemento o elementos mediante un selector, podemos modificarlo con los siguientes métodos:

- `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
- `replacewith()`
Reemplaza 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
- *replacewith()* reemplaza el elemento

Ejemplo: añadir filas a una tabla

```
<body>
  <button id=boton01>Registrar hora</button>
  <table id=tabla_horas>
    <tr>
      <th>Hora ISO 8651</th>
    </tr>
  </table>

  <script>
    $(document).ready(function() {
      $("#boton01").click(function() {
        let texto;
        texto = '<tr class="hora"><td>' + moment().format() + '</td></tr>';
        $("#tabla_horas").append(texto);
      });
    });
  </script>
```

<http://ortuno.es/append.html>

Borrar contenido

El método `remove()` borra los elementos seleccionados

```
$(document).ready(function() {  
  $("#boton01").click(function() {  
    var texto  
    texto = '<tr class="hora"><td>' + moment().format() +  
    ↪ '</td></tr>';  
    $("#tabla_horas").append(texto);  
  });  
  
  $("#boton02").click(function() {  
    $(".hora").remove();  
  });  
});
```

<http://ortuno.es/remove.html>

Lectura de un radiobutton / checkbox

Para leer si un elemento de un formulario está marcado o no

- Pasamos a jQuery el selector `input[name=NOMBRE-DE-CAMPO]:checked, ID-FORMULARIO`
- Invocamos el método `val()`
- Esto nos devolverá el *value* asociado

```
$(document).ready(function() {  
  $('#formulario01 ').on('change', function() {  
    let respuesta = $('input[name=figura]:checked', '#formulario01').val();  
    console.log(respuesta);  
  });  
});  
  
<h3>Elige una figura</h3>  
<form id=formulario01>  
  <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>
```

<http://ortuno.es/radio.html>

tooltip

Un *tooltip* es un pequeño cuadro emergente con información contextual. Típicamente describe el elemento sobre el que se posiciona el ratón

Para añadir un *tooltip* a un elemento cualquiera, con Bootstrap y jQuery:

- ➊ Añadimos al elemento el atributo `data-toogle="tooltip"`
- ➋ Le añadimos el atributo `title` con el texto del *tooltip*
- ➌ Posicionamos el *tooltip* con el atributo `data-placement`, que puede tomar los valores `top`, `bottom`, `left` o `right`
- ➍ Una vez definidos los elementos *tooltip*, los activamos desde jQuery con

```
$('.[data-toogle="tooltip"]').tooltip();
```

```
<body>
  <div class="container">
    <br>
    <button data-toggle="tooltip" title="Esto es un tooltip"
      data-placement="bottom" >Al pasar el ratón aparecerá un
      <span class="italic">tooltip<span></button>
  </div>
  <script>
    $(document).ready(function(){
      $('[data-toggle="tooltip"]').tooltip();
    });
  </script>
</body>
</html>
```

<http://ortuno.es/tooltip.html>

Validación de un formulario

El método `change()` recibe una función que será el manejador con el evento que se genera cuando cambia un input de un formulario. Una vez seleccionado el input, el método `$(this).val()` contiene su valor.

```
<input type="password" name="contrasenia" id="contrasenia">
```

...

```
$("#contrasenia").change(function() {  
    if ( $(this).val().length > 5 ) {  
        $("#validacion").text("Contraseña aceptable");  
    } else {  
        $("#validacion").text("Contraseña muy corta");  
    };  
});
```

<http://ortuno.es/validar.html>

El ejemplo anterior tenía un problema: el evento `change` solo se dispara cuando el foco abandona el input (cuando el usuario ha acabado de editar ese campo y pasa al siguiente)

Si queremos vincular un manejador con cualquier cambio producido en un input, debemos inscribir el manejador a los eventos `change`, `keyup`, `paste` y `mouseup`

- Para ello empleamos el método `on()`, que recibe como primer argumento la lista de eventos, y como segundo, el manejador

```
$("#contrasenia").on('change keyup paste mouseup', function() {  
  if ( $(this).val().length > 5) {  
    $("#validacion").text("Contraseña aceptable");  
  } else {  
    $("#validacion").text("Contraseña muy corta");  
  };  
});
```

<http://ortuno.es/validar2.html>

Depuración

Si tu programa no se comporta como esperas

- Presta atención a los mensajes que pueda mostrar el motor de JavaScript en la consola
 - Atajo en Chrome: Ctrl Shift I
- Comprueba que los valores que deban compartirse por distintos manejadores sean variables globales (de esa página), esto es, declarados fuera de cualquier función
- Comprueba el valor de las variable relevantes mediante trazas en la consola (`console.log()`)