MÓDULO 4:

Eventos, Objetos, Arrays, JSON, Storage, Boolean y MVC

Juan Quemada, DIT - UPM

Índice

MODULO 4 - Eventos, Objetos, Arrays, JSON, Storage, Boolean y MVC

1. Eventos, DOM e interacción 3
2. Mas sobre Objetos: Literal de objetos, notación corchetes y propiedades dinámicas 12
3. MVC de cliente: Eventos, modelo con objetos y arrays, vistas y controladores 17
4. JSON: JavaScript Object Notation 36
5. Boolean y operadores relacionados: !, && y ||, ===, !==, <, >, >=, <=, -?-:- 41
6. Almacenamiento en cliente: localStorage, sessionStorage y "Same Origin Policy" 44

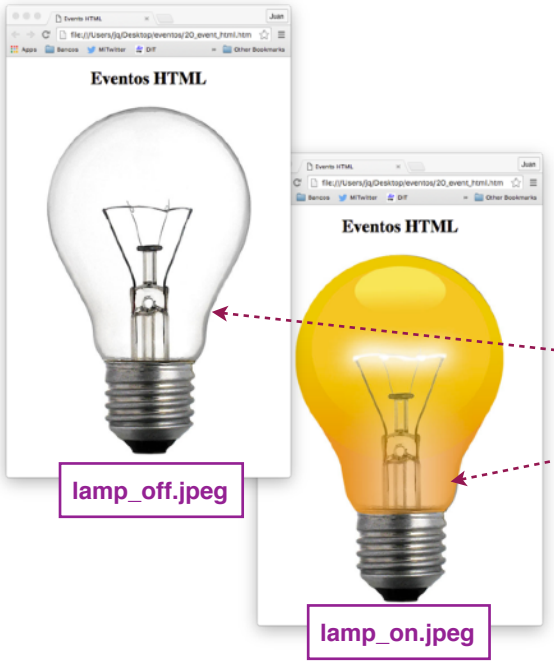
Eventos, DOM e interacción

Juan Quemada, DIT - UPM

Eventos y Manejadores

- **Evento**
  - Indica la ocurrencia de un hecho que es atendido por un **manejador**
- **Manejador** de un evento
  - Bloque de código o función asociada al evento (se ejecuta al ocurrir este)
- La norma de JavaScript incluye eventos asociados a acciones exteriores
  - Temporizador vence, click en botón, tocar en pantalla, pulsar tecla, **...**
    - Ver: https://developer.mozilla.org/en-US/docs/Web/Events
- Un programa JavaScript se guía por eventos (event driven)
  - El script inicial debe configurar los primeros manejadores
    - El programa solo ejecuta después los manejadores de los eventos que ocurren

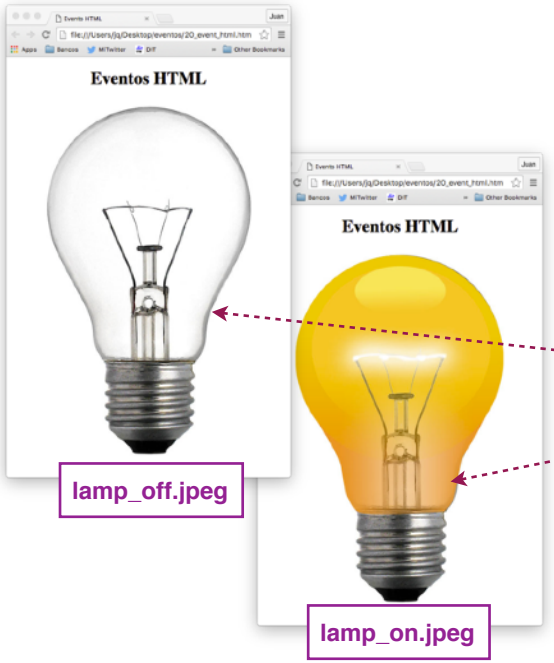
Eventos en HTML




```
<!DOCTYPE html>
<html><head><title>Evento HTML</title>
<style> body{text-align:center;} </style>
</head><body>
  <h1>Eventos</h1>

  <img src="lamp_off.jpg"
       ondblclick="this.src='lamp_off.jpg';"
       onclick="this.src='lamp_on.jpg';"
       />
</body>
</html>
```

- Los elementos HTML tienen **atributos** que asocian eventos, por ejemplo
  - **onclick** (clic), **ondbclick** (doble clic), …
    - https://uniwebsidad.com/libros/javascript/capitulo-6/modelo-basico-de-eventos-2
    - https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event
- El **código** asignado al atributo (el manejador) se **ejecuta** al ocurrir el **evento**
  - **this** referencia el **objeto DOM** del elemento asociado al evento (en este caso <img>)
    - Las **propiedades** de DOM asociadas a **atributos** de HTML suelen tener el **mismo nombre**, p.e. **src**
- onclick="this.src ='**lamp_on.jpeg**'" o ondbclick="this.src ='**lamp_off.jpeg**'"
  - asigna el URL de la imagen (**lamp_on.jpeg** o **lamp_off.jpeg**) al atributo **src** de **<img>**
    - **Enciende** o **apaga** la bombilla con **clic** o **doble-click** en ella (muestra **lamp_on.jpeg** o **lamp_off.jpeg**)

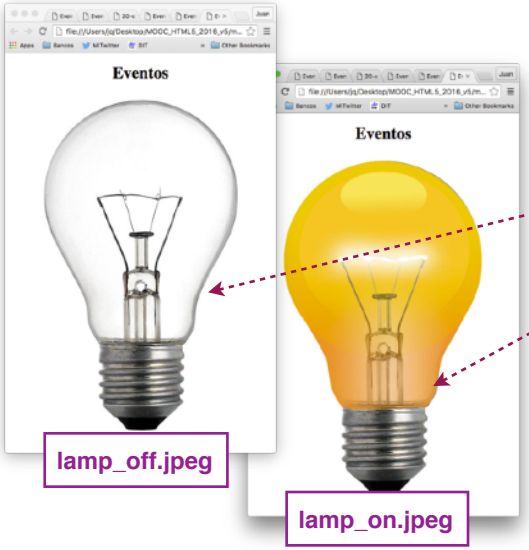
Eventos en HTML




```
<!DOCTYPE html>
<html><head><title>Evento HTML</title>
<style> body{text-align:center;} </style>
</head><body>
  <h1>Eventos</h1>

  <img src="lamp_off.jpg"
       ondblclick="this.src='lamp_off.jpg';"
       onclick="this.src='lamp_on.jpg';"
       />
</body>
</html>
```

- Las apps Web son muy grandes y hoy se recomienda
  - separar HTML, CSS y JavaScript en partes o ficheros diferentes
    - Así cada parte puede generarse por un equipo diferente
- Normalmente
  - HTML se incluye en el cuerpo de la página HTML
  - CSS se incluye en la cabecera o en un fichero separado
  - JavaScript se incluye en la cabecera o en un fichero separado

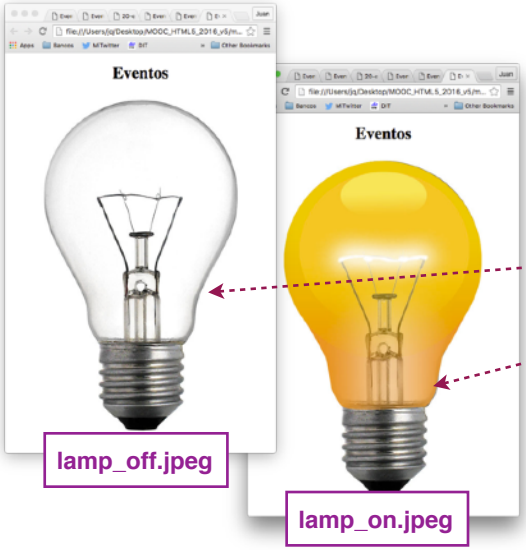
Eventos con addEventListener




```
<!DOCTYPE html>
<html><head><title>Evento</title><meta charset="UTF-8">
<style> body{text-align:center;} </style>

  <script type="text/javascript">

    function inicializar() {
      let img = document.getElementById('i1');

      img.addEventListener("dblclick", function () {img.src='lamp_off.jpg'});
      img.addEventListener("click",    function () {img.src='lamp_on.jpg'});
    }
  </script>
</head>   <!-- El arbol DOM debe estar construido al ocurrir onload -->
<body onload="inicializar()">
  <h1>Eventos</h1>
  <img src="lamp_off.jpg" id="i1"/>
</body></html>
```

- **objeto.addEventListener(tipo_ev, manejador)** asocia manejadores a **eventos**
  - Además existe el método **removeEventListener(..)** para **desinstalar** el **manejador**
    - https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener
    - https://developer.mozilla.org/en-US/docs/Web/Events
  - Un elemento DOM puede tener varios manejadores asociados, que se ejecutan en orden de instalación
- El **manejador** es un literal de función: **(event) => { .. código.. }** o **function (event) { .. código.. }**
  - **event** es un objeto con información del evento ocurrido: https://developer.mozilla.org/en-US/docs/Web/API/Event
    - El parámetro **event** no es necesario en este ejemplo, pero se ha incluido para hacerlo explícito
- La función **inicializar()** define los eventos y se invoca al ocurrir el **evento onload**
  - El **evento onload** (<body>) ocurre cuando lo página esta completamente cargada (sino no funcionaría)

Evento
DOMContentLoaded

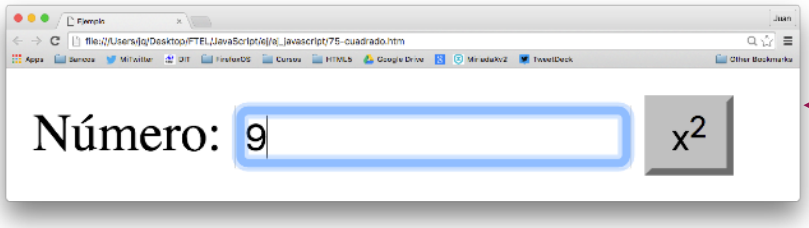



```
<!DOCTYPE html>
<html><head><title>Evento</title><meta charset="UTF-8">
<style> body{text-align:center;} </style>

  <script type="text/javascript">

    document.addEventListener(
      'DOMContentLoaded',
      () => {
        let img = document.getElementById('i1');

        img.addEventListener("dblclick", ()=> img.src='lamp_off.jpg');
        img.addEventListener("click", ()=> img.src='lamp_on.jpg');
      })
  </script>
</head>
<body>
  <h1>Eventos</h1>
  <img src="lamp_off.jpg" id="i1"/>
</body></html>
```

- El evento **DOMContentLoaded** indica cuando finaliza la carga de la página
  - Aunque los demás recursos (imágenes, ficheros JavaScript o CSS, ..) no hayan cargado todavía
    - https://developer.mozilla.org/en-US/docs/Web/API/Window/DOMContentLoaded_event
- El evento load ocurre en cambio cuando se ha cargado la página y todos sus recursos
  - **DOMContentLoaded** ocurre antes que **load** y es preferible utilizarlo
    - https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event

Ejemplo Mini-calculadora

- En un tema anterior vimos esta calculadora
  - Tiene manejadores asociados a elementos HTML directamente
- ¡Es una practica **no** recomendada!
  - Mezcla código JavaScript y HTML
- **JavaScript y HTML**
  - Deben separarse
    - Idealmente en ficheros diferentes




```
<!DOCTYPE html><html><head>
<title>Ejemplo</title><meta charset="utf-8">
<script type="text/javascript">

function vaciar () {
  document.getElementById("n1").value = "";
}
function cuadrado() {
  var num = document.getElementById("n1");
  num.value = num.value * num.value;
}
</script>
</head><body>
  Número:
  <input type="text" id="n1" onclick="vaciar()">

  <button onclick="cuadrado()">
    x<sup>2</sup>
  </button>
</body></html>
```

Calculadora con eventos JavaScript

- La programación de eventos en **Vainilla JavaScript** actual es sencilla
  - (Vainilla significa usar solo ES6+, sin librerías)
- Modificaciones
  - Los elementos HTML se identifican con id=".."
  - Objetos DOM se acceden con getElementById
  - La nuevas notaciones de ES6 hacen el código mas compacto
  - Los manejadores de eventos se instalan al ocurrir el evento "DOMContentLoaded" utilizando el método addEventListener(..)


```
<!DOCTYPE html><html><head><meta charset="utf-8">
  <script type="text/javascript">

    document.addEventListener(
      'DOMContentLoaded',
      () => {
        let n1 = document.getElementById('n1');
        n1.addEventListener(
          'click',
          () => n1.value = ""
        );

        document.getElementById('b1')
        .addEventListener(
          'click',
          () => n1.value = n1.value * n1.value
        );
      }
    )

  </script>
</head>
<body>
  Número:
  <input type="text" id="n1">

  <button id="b1"> x<sup>2</sup> </button>
</body>
</html>
```

Obtener objeto DOM deL cajetín

Obtener objeto DOM del botón

Manejador del evento click en cajetín

Vaciar el cajetín

Muestra en cajetín el cuadrado del número tecleado (accede al cajetín con **n1.value**).

Manejador del evento click en botón x^2

Bubbling (eventos delegados)

Los clicks sobre cualquier elemento generarán un evento que burbujeará hacia arriba., pero solo los elementos con atributos **id="b1"** o **id="n1"** invocarán un manejador.

Los dos **manejadores** de eventos se asocian a **document**, pero la activación del manejador se **filtra** con de atributos **id**

- Este ejemplo muestra como capturar eventos delegados ("bubbling") en document en Vainilla JavaScript
- **evento** del manejador: ev => {.....}
  - contiene el **elemento HTML** sobre el que se **clicó** en la propiedad **target**
- Atributos **id** o **class** se utilizan para **filtrar eventos** con el método **matches(..)** antes de invocar **controladores**
  - https://developer.mozilla.org/en-US/docs/Web/API/Element/matches

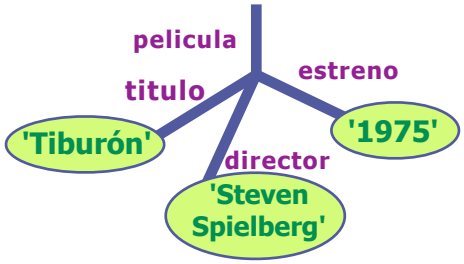
```
<!DOCTYPE html>
<html><head><title>Calculadora</title><meta charset="utf-8">
  <script type="text/javascript">

    document.addEventListener('click', ev => {
      if      (ev.target.matches('#n1')) vaciar();
      else if (ev.target.matches('#b1')) cuadrado();
    })

    function vaciar (){
      document.getElementById('n1').value = "";
    }

    function cuadrado (){
      let num = document.getElementById('n1');
      num.value = num.value * num.value;
    }
  </script>
</head><body>
  Número:
  <input type="text" id="n1">

  <button id="b1"> x<sup>2</sup> </button>
</body>
</html>
```

matches: filtra los eventos de elementos HTML que no casan con el selector CSS.

Mas sobre Objetos:

Literal de objetos, notación corchetes y propiedades dinámicas

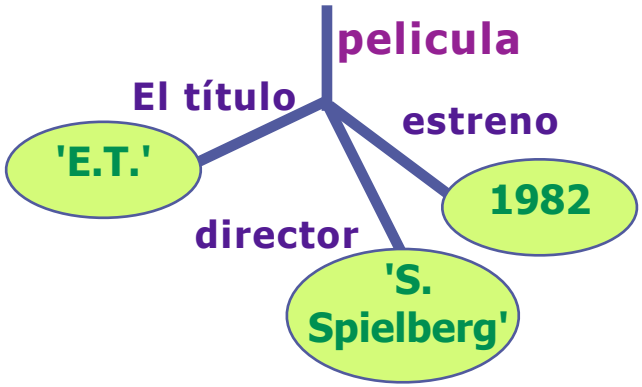
Literal de Objetos




```
// Ejemplo de objeto con datos de una película
let pelicula = {
    titulo: "Tiburón",
    director: "Steven Spielberg",
    estreno: 1975
};
pelicula.titulo      =>  "Tiburón"
pelicula.director    =>  "Steven Spielberg"
pelicula.estreno     =>   1975
```

- **Literal de objeto**
  - Agrupa propiedades (variables especiales asociadas a un objeto)
    - Por ejemplo **{ x: 5, y: 7, z: 2 }**
- **Propiedades** de un objeto
  - Se recomienda asignar nombres **diferentes** (no es obligatorio)
  - Sus nombres deben seguir la misma **sintaxis** que los de las **variables**
    - Por ejemplo **a, _method_, $1, ….**
- **Operador punto**
  - **objeto.propiedad**: Accede al contenido de propiedades por nombre

Notación array de objetos



- Operador **punto**
  - **objeto.propiedad**
    - **obj.a**, **obj._method_**, **obj.$1**, ….
- Operador **corchetes*** (o array)
  - **objeto[<expr>]**
    - Donde **<expr>** se interpreta como un **string** con el **nombre de la propiedad**
    - Accede a nombres con strings **arbitrarios**
      - Por ejemplo: **"", "El Título", "%xxx%", ...**

*Los arrays son objetos y el operador corchetes de ambos es el mismo. Los elementos de un array son propiedades especiales indexables con números o con el número entre comillas.

- **Literal** de objetos (con strings)
  - Los nombres de propiedad pueden ser también strings arbitrarios
    - Por ejemplo: "", "El Título", "%xxx%", ...

```
let pelicula = {
    'El título': 'E.T.',
    director: 'S. Spielberg',
    estreno: 1982
};

// Acceso a propiedades
movie['El título']  //  =>  'E.T.'
movie['director']  //  =>  'S. Spielberg'

// Acceso con variables usando [..]
let t = 'estreno';
pelicula[t];                  //  => 1982
pelicula['estre' + 'no']   //  => 1982

pelicula.t                    //  => undefined
```

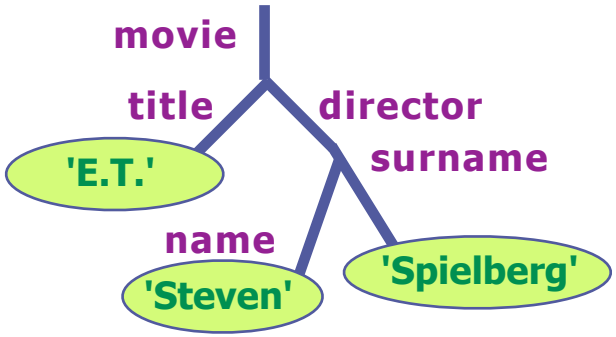
https://javascript.info/object

Objetos anidados: árboles

```
let movie = {
    title: 'E.T.',
    director:{
        name:'Steven',
        surname: 'Spielberg'
    }
};
```

- Los objetos pueden **anidarse** entre sí
  - Los objetos anidados representan **árboles**
    - **DOM** es un árbol enorme de objetos anidados

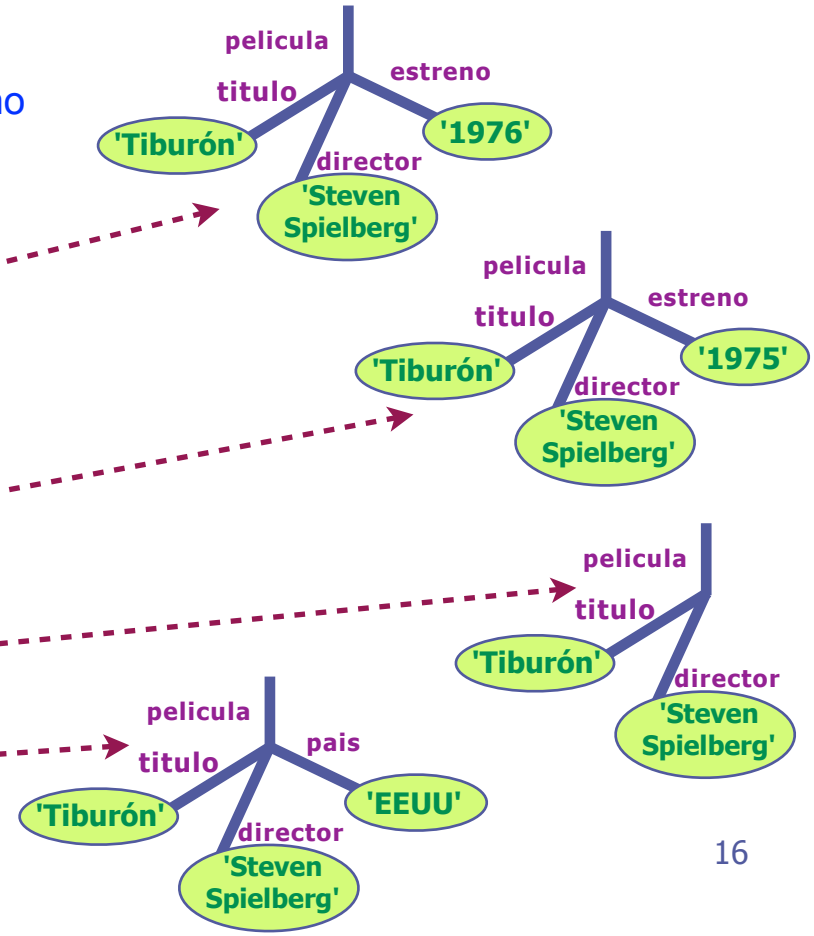
- La notación punto o array puede **encadenarse** y **mezclarse**
  - Representando un **camino en el árbol**
    - Las siguientes expresiones se evaluan así:
      - **movie.title => 'E.T.'**
      - **movie.director.name => 'Steven'**
      - **movie['director']['name'] => 'Steven'**
      - **movie['director'].surname => 'Spielberg'**
      - **movie.director => {name:'Steven', surname: 'Spielberg'}**
      - **movie.premiere => undefined**
      - **movie.premiere.year => Error_de_ejecución**



Propiedades dinámicas

- Las **propiedades** de un objeto se pueden **crear** y **destruir** con estás sentencias
  - **DOM** hace uso de la creación y borrado dinámico de propiedades para actualizar el árbol
- **Asignar** o **crear** propiedades: **obj.prop = valor;**
  - **pelicula.estreno = 1975;** **asigna** el valor 1975 a la propiedad **estreno** que **ya existe**
  - **pelicula.pais = "EEUU";** **crea** la propiedad **pais** y le **asigna** "EEUU", porque **no existe**
- Borrar propiedades: **delete obj.prop**
  - **delete pelicula.estreno** elimina la propiedad estreno
- Testear la propiedades: **"prop" in obj**
  - devuelve **true** o **false** según que exista la propiedad o no

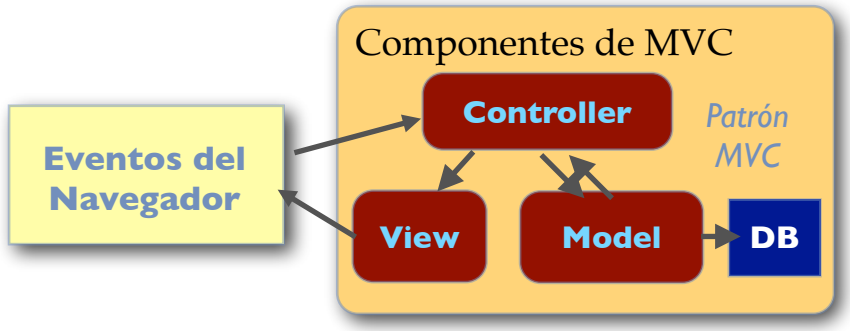
```
let pelicula = {  titulo: "Tiburón",
                  director: "Steven Spielberg",
                  estreno: 1976
};

pelicula.estreno = 1975;

delete pelicula.estreno;

pelicula.pais = "EEUU";
```



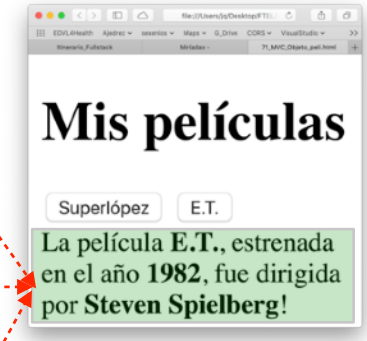
MVC de cliente: Eventos, modelo con objetos y arrays, vistas y controladores

MVC (Modelo-Vista-Controlador) de Cliente

El patrón MVC fue propuesto por Trygve Reenskaug en 1979 y es muy utilizado.



- **Evento**
  - Mueve la aplicación ejecutando **controladores** (**manejadores** de eventos)
- **Modelo**
  - Conjunto de **datos** de la aplicación, por ejemplo
    - var et = { titulo: **"E.T."**, director: **"Steven Spielberg"**, estreno: **1982**};
- **Vista**
  - Generador de la **representación visual**
    - Función que genera el código HTML que muestra los datos del modelo
- **Controlador**
  - Orquesta la respuesta a una petición del usuario
    - Normalmente, busca los datos, los formatea con la Vista y los muestra



Plantilla (o Interpolación) de strings

- **Plantilla de string** de ES6
  - Delimita el texto con comillas invertidas **`...`**
    - **`hola, que tal`** es equivalente a **"hola, que tal"** o a **'hola, que tal'**
- Permite definir strings multi-línea, por ejemplo
  - **`hola,**
    que tal`
- Además permite incluir parámetros con **${<expr>}**
  - **<expr>** se evalúa y se inserta transformada en string, por ejemplo

```
`Un día tiene ${24*60} minutos`
        // genera el string "Un día tiene 1440 minutos" y es equivalente a:
"Un día tiene " + 24*60 + " minutos"
```

- Simplifica la definición de vistas de MVC

Ejemplo I: MVC (Model-Vista-Controller)

```
<!doctype html><html><head>
    <meta charset="utf-8">
    <script type="text/javascript">

        // MODELO
        var superl = {titulo: "Superlópez", director: "Javier Ruiz Caldera", estreno: "2018"};
        var et = { titulo: "E.T.", director: "Steven Spielberg", estreno: 1982};

        // VISTA
        function showView (peli) {
            return `
            La película <b> ${peli.titulo}</b>, estrenada
            en el año <b> ${peli.estreno}</b>, fue
            dirigida por <b> ${peli.director}</b>!`
        }

        // CONTROLADORES
        function superlContr(){ document.getElementById("p").innerHTML = showView(superl);};
        function etContr(){ document.getElementById("p").innerHTML = showView(et);};
    </script>
</head>
<body>
    <h1> Mis películas </h1>
    <button onclick="superlContr()">Superlópez</button>
    <button onclick="etContr()">E.T.</button>
    <div id="p"></div>
</body>
</html>
```

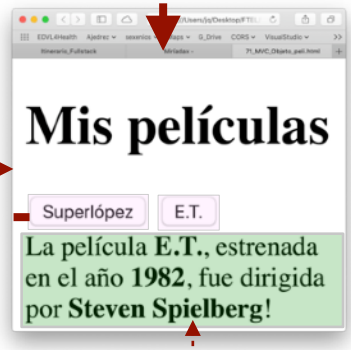
Modelo: objetos **superl** y **et** con datos de las películas.

Vista: función **showView (peli)** que genera el HTML visualizado con los datos de la película a mostrar.

Controladores: funciones **superlContr()** y **etContr()** que preparan la respuesta con los datos solicitados y la vista showView, y la muestran en el bloque .

Botones con **eventos onclick** que activan cada controlador al clicar su botón.

Bloque **<div** id="**p**"**>** donde se inserta la vista.




```
<div id="p">
  " La película "
  <b> E.T.</b>
  ", estrenada en el año "
  <b> 1982</b>
  ", fue dirigida por "
  <b> Steven Spielberg</b>
  "!"
</div>
```

Código HTML generado con indexView(pelis), e insertado por el controlador en el bloque **<div** id="**p**"**>** cuando se pulsa el botón **E.T.**

Ejemplo II: Generar HTML con lista de películas

```
<!doctype html><html><head>
    <meta charset="utf-8">
    <script type="text/javascript">

        // MODELO
        var peliculas = ["Superlópez", "E.T.", "Interstellar"];

        // VISTA
        function indexView (pelis) {
            var    i=0,   html = "<ul>";

            while(i < pelis.length) {
              html = html + `<li> ${pelis[i]}</li>`;
              i = i + 1;
            };

            return html + "</ul>";
        };

        // CONTROLADOR
        function indexContr() {
            document.getElementById("main").innerHTML = indexView(peliculas);
        };

        // EVENTOS
        document.addEventListener('DOMContentLoaded', ev => indexContr());
    </script>
</head>

<body>

    <h1> Mis películas </h1>
    <div id="main"></div>
</body>
</html>
```

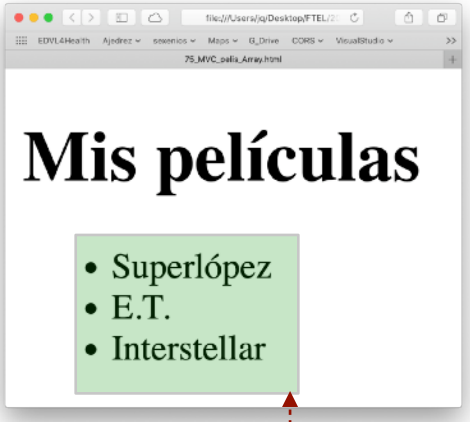
Modelo: array con una lista de títulos de películas.

Vista: función **indexView (pelis)** que genera el HTML visualizado con la lista de títulos de las películas del modelo.

Controlador: función **indexContr()** que prepara la respuesta con los datos del modelo y la vista indexView, para mostrarla en el bloque .

Instala **indexContr()** como manejador del evento **DOMContentLoaded**, de forma que este se ejecute cuando la página Web está ya cargada.

Bloque **<div** id="**main**"**>** donde se inserta la vista.




```
<div id="main">
  <ul>
    <li> Superlópez</li>
    <li> E.T.</li>
    <li> Interstellar</li>
  </ul>
</div>
```

Código HTML generado con **indexView(pelis)**, e insertado por el controlador en el bloque **<div** id="**p**"**>** al ocurrir el evento **onload**.

Ejemplo III: Añadir y borrar películas

Ejemplo III: gestionar una lista con un array

- El **Array** se puede utilizar para gestionar **listas**
  - Una **lista** es una secuencia ordenada de elementos
- La gestión de la **lista** necesita
  - **Actualizar**, **eliminar** y **añadir** elementos de la lista
- **Actualizar** un **elemento** (asignando un valor)
  - **a[i] = e** asigna el valor **e** al elemento **i** del array **a** (lista)
- **Eliminar** un elemento del **array**
  - **a.splice(i,1)** elimina el elemento de índice **i** del array **a**
- **Añadir** un elemento al array
  - **a.push(e)** añade el elemento **e** al final del array **a**
    - El array tendrá un elemento más
  - **a.splice(i, 0, e)** añade el elemento **e** en la posición **i** del array **a**
    - El array tendrá un elemento más. Todos los elementos después de **i** se retrasan una posición

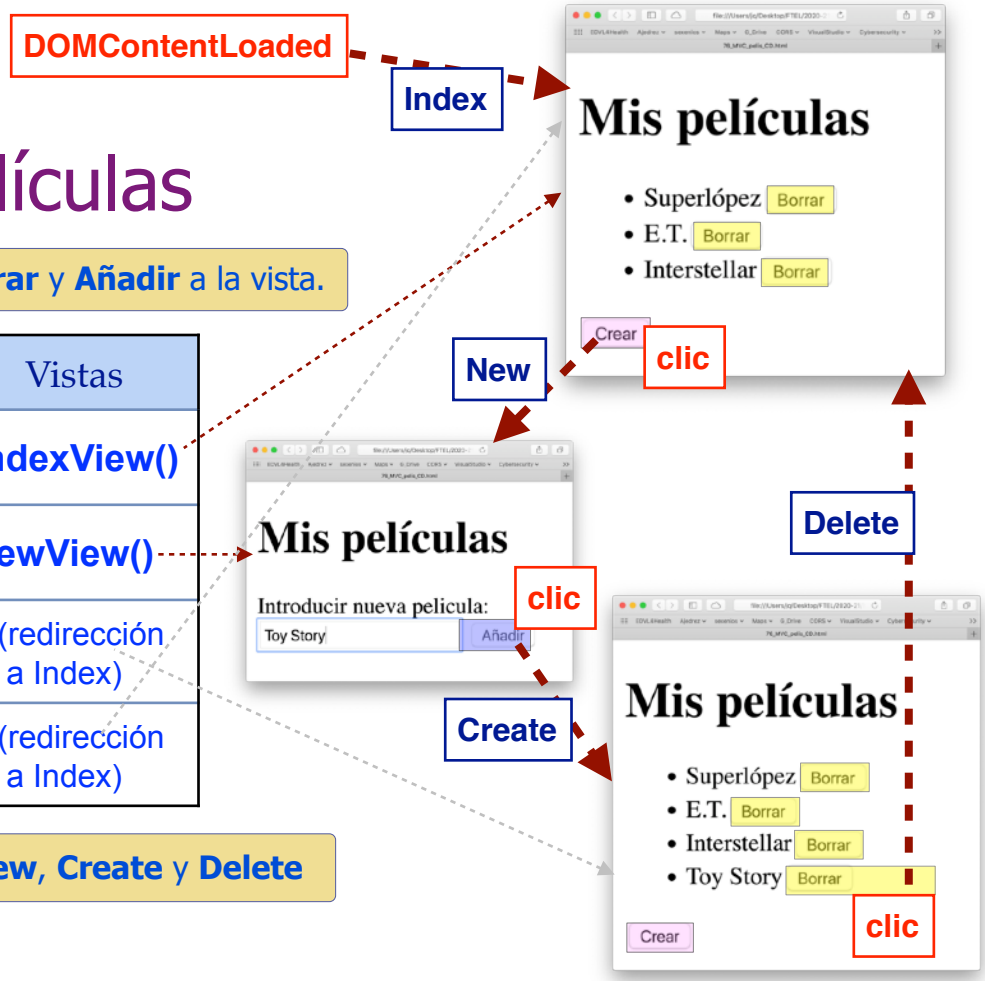
```
//                                  Lista de nombres
var a = ["Eva", "Juan", "Rafa"];
a.length      // => 3
a     // => ["Eva", "Juan", "Rafa"]

//                                  Actualizar
a[0] = "Ana"
a     // => ["Ana", "Juan", "Rafa"]

//                                  Eliminar
a.splice(1, 1)
a     // => ["Ana", "Rafa"]

//                                  Añadir al final
a.push("José")
a     // => ["Ana", "Rafa", "José"]

//                                  Añadir  en i=2
a.splice(2, 0, "Eva")
a     // => ["Ana", "Rafa", "Eva", "José"]
```

Ejemplo III: Añadir y borrar películas

Index es igual que en Ejemplo II, salvo que se añaden los botones **Borrar** y **Añadir** a la vista.

Primitiva	Eventos	Controladores	Vistas
Index: mostrar lista de películas	DOMContentLoaded	indexContr()	indexView()
New: formulario de creación de película	click$_{Crear}$	newContr()	newView()
Create: añadir película al modelo	click$_{Añadir}$	createContr()	-- (redirección a Index)
Delete: borrar una película del modelo	click$_{Borrar}$	deleteContr(i)	-- (redirección a Index)

Se añaden las primitivas **New**, **Create** y **Delete**



- **Index**
  - Muestra la lista de recursos (películas)
- **New** (complementaria de **Create)**
  - Muestra el formulario para teclear el nuevo recurso (película)
- **Create** modifican el contenido de un recurso (película) en el modelo
  - Añade la nueva película al modelo (al final del array)
- **Delete** elimina un recuso (película) del modelo

Ejemplo III: primitiva Index

```
<!doctype html><html><head><meta charset="utf-8">
    <script type="text/javascript">
        // MODELO
        var peliculas = ["Superlópez", "E.T.", "Interstellar"];

        // VISTAS
        function indexView (pelis) {
            var    i=0,    html = "<ul>";
            while(i < pelis.length) {
              html = html + `<li> ${pelis[i]} <button id="delete" data-my-id="${i}">Borrar</button></li>`;
              i = i + 1;
            };
            return html + `</ul> <button id="new">Crear</button>`;
        };

        function newView () {
            return `Introducir nueva pelicula: <input type="text" id="peli"> <button id="create">Añadir</button>`
        };

        // CONTROLADORES
        function indexContr() { document.getElementById("main").innerHTML = indexView(peliculas);};
        function newContr() { document.getElementById("main").innerHTML = newView();};
        function createContr() {
            peliculas.push(document.getElementById("peli").value);
            indexContr();
        };
        function deleteContr(i) {
            peliculas.splice(i,1);
            indexContr();
        };

        // EVENTOS
        document.addEventListener('DOMContentLoaded', ev => indexContr());
        document.addEventListener('click', ev => {
            if      (ev.target.matches('#new'))    newContr();
            else if (ev.target.matches('#create')) createContr();
            else if (ev.target.matches('#delete')) deleteContr(ev.target.dataset.myId);
        })
    </script>
</head>
<body>
    <h1> Mis películas </h1>
    <div id="main"></div>
</body>
</html>
```

Modelo: array con una lista inicial de películas.

Se añade el botón **Borrar** detras de cada película, con los atributos id="delete" para identificar el clic en este botón y data-my-id="${i}" con el índice **i** de la película a borrar en el array.

Se añade el botón **Añadir** al final de la vista, con el atributo id="new" que identifica el clic en este botón.

indexContr() no se modifica respecto al Ejemplo II.

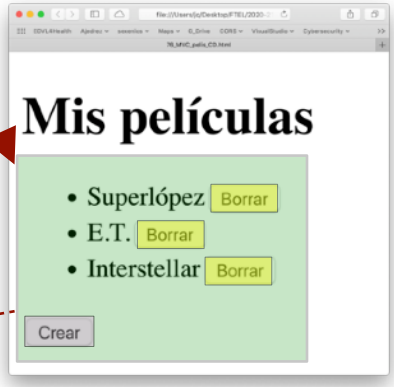
indexContr() se invoca igual que en el Ejemplo II.

Bloque **<div** id="**main**"**>** donde se inserta la vista.

El código resaltado en crema es el de la primitiva **Index.** Es igual al del Ejemplo II, salvo los botones **Crear** y **Borrar**.

DOMContentLoaded

Index



Router de Eventos

- Instala los manejadores asociados a los diferentes tipos de eventos en **document**
  - Los eventos (clicks en botones) burbujean hasta ser capturados en **document**
    - El **atributo id** de cada botón indica el controlador asociado, por ejemplo **<button id="new">...**
- El manejador de eventos recibe el objeto DOM del elemento clicado en **ev.target**
  - El método **matches(<selector-CSS>)** es true si <selector-CSS> casa con el objeto DOM clicado
- HTML permite definir atributos de datos **data-***, por ejemplo **data-my-id="2"**
  - Los valores asignados a estos atributos estarán en propiedades (en **camel-case**) de **ev.target.dataset**
    - Por ejemplo, **data-id="2"** y **data-my-id="2"** generan las propiedades **{.., id:2, myId:2, ..}** en **ev.target.dataset**
  - Ver: https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Use_data_attributes

El manejador del evento **DOMContentLoaded** que indica página cargada se instala en document. Invoca **indexContr()**, para mostrar la lista de películas al cargar la página.

```
// EVENTOS
document.addEventListener('DOMContentLoaded', ev => indexContr());
document.addEventListener('click', ev => {
    if      (ev.target.matches('#new'))    newContr();
    else if (ev.target.matches('#create')) createContr();
    else if (ev.target.matches('#delete')) deleteContr(ev.target.dataset.myId);
})
```

Si el **id** del objeto DOM del elemento clicado es **new** se invoca **newContr()**.

Si el **id** del objeto DOM del elemento clicado es **create** se invoca **createContr()**.

Si el **id** del objeto DOM del elemento clicado es **delete** se invoca **deleteContr(i)**.

ev.target es el objeto DOM del elemento HTML clicado.

El manejador del evento **click** atiende clics sobre cualquier elemento HTML de la página cargada (document). El manejador utiliza el método **matches('#..')** para determinar el atributo **id** del botón clicado.

ev.target.dataset contiene propiedades con los atributos definidos por el usuario con el nombre en "camel-case", es decir el atributo **data-my-id** esta en la propiedad **myId**.

Ejemplo III: primitiva New

```
<!doctype html><html><head><meta charset="utf-8">
    <script type="text/javascript">
        // MODELO
        var peliculas = ["Superlópez", "E.T.", "Interstellar"];

        // VISTAS
        function indexView (pelis) {
            var    i=0,    html = "<ul>";
            while(i < pelis.length) {
              html = html + `<li> ${pelis[i]}<button id="delete" data-my-id="${i}">Borrar</button></li>`;
              i = i + 1;
            };
            return html + `</ul> <button id="new">Crear</button>`;
        };

        function newView () {
            return `Introducir nueva pelicula: <input type="text" id="peli"> <button id="create">Añadir</button>`
        };

        // CONTROLADORES
        function indexContr() { document.getElementById("main").innerHTML = indexView(peliculas);};
        function newContr() { document.getElementById("main").innerHTML = newView();};
        function createContr() {
            peliculas.push(document.getElementById("peli").value);
            indexContr();
        };
        function deleteContr(i) {
            peliculas.splice(i,1);
            indexContr();
        };

        // EVENTOS
        document.addEventListener('DOMContentLoaded', ev => indexContr());
        document.addEventListener('click', ev => {
            if      (ev.target.matches('#new'))    newContr();
            else if (ev.target.matches('#create')) createContr();
            else if (ev.target.matches('#delete')) deleteContr(ev.target.dataset.myId);
        })
    </script>
</head>
<body>
    <h1> Mis películas </h1>
    <div id="main"></div>
</body>
</html>
```

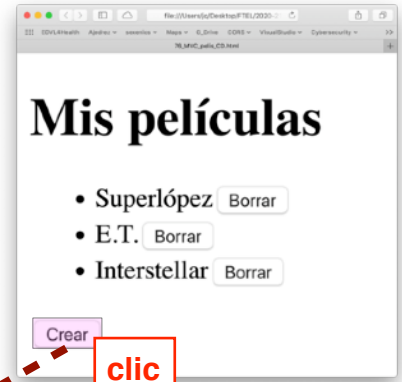
El código resaltado en rosa añade la primitiva **New** al Ejemplo II.

Botón **Crear** con el atributo id="new" que identifica el clic y dispara la primitiva **New**.

newView () genera el formulario de **Añadir** nueva película.

newContr() muestra el formulario de añadir nueva película.

newContr() se activa si el botón clicado tiene id="new".



New

clic

Bloque **<div** id="**main**"**>** donde se inserta la vista.

Ejemplo III: primitiva Create

```
<!doctype html><html><head><meta charset="utf-8">
    <script type="text/javascript">
        // MODELO
        var peliculas = ["Superlópez", "E.T.", "Interstellar"];

        // VISTAS
        function indexView (pelis) {
            var    i=0,    html = "<ul>";
            while(i < pelis.length) {
              html = html + `<li> ${pelis[i]}<button id="delete" data-my-id="${i}">Borrar</button></li>`;
              i = i + 1;
            };
            return html + `</ul> <button id="new">Crear</button>`;
        };

        function newView () {
            return `Introducir nueva pelicula: <input type="text" id="peli"> <button id="create">Añadir</button>`
        };

        // CONTROLADORES
        function indexContr() { document.getElementById("main").innerHTML = indexView(peliculas);};
        function newContr() { document.getElementById("main").innerHTML = newView();};
        function createContr() {
            peliculas.push(document.getElementById("peli").value);
            indexContr();
        };
        function deleteContr(i) {
            peliculas.splice(i,1);
            indexContr();
        };

        // EVENTOS
        document.addEventListener('DOMContentLoaded', ev => indexContr());
        document.addEventListener('click', ev => {
            if      (ev.target.matches('#new'))    newContr();
            else if (ev.target.matches('#create')) createContr();
            else if (ev.target.matches('#delete')) deleteContr(ev.target.dataset.myId);
        })
    </script>
</head>
<body>
    <h1> Mis películas </h1>
    <div id="main"></div>
</body>
</html>
```

El **código resaltado en azul** añade la primitiva **Create** al Ejemplo II.

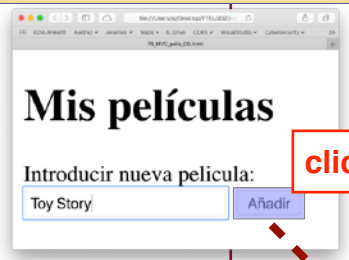
Botón **Añadir** con el atributo id="create" que identifica el clic y dispara la primitiva **Create**.

createContr() añade la película tecleada en el cajetín, como último elemento del array del modelo y muestra la nueva lista de películas invocando **indexContr()**.

peliculas.push(document.detElementById("peli").value) añade la película tecleada en el cajetín, como último elemento del array.

indexContr() redirecciona a la primitiva **Index** que muestra la lista de películas una vez actualizado el modelo.

createContr() se invoca si id="create".



clic

Create

Bloque **<div** id="**main**"**>** donde se inserta la vista.

Ejemplo III: primitiva Delete

```
<!doctype html><html><head><meta charset="utf-8">
    <script type="text/javascript">
        // MODELO
        var peliculas = ["Superlópez", "E.T.", "Interstellar"];

        // VISTAS
        function indexView (pelis) {
            var    i=0,    html = "<ul>";
            while(i < pelis.length) {
              html = html + `<li> ${pelis[i]}<button id="delete" data-my-id="${i}">Borrar</button></li>`;
              i = i + 1;
            };
            return html + `</ul> <button id="new">Crear</button>`;
        };

        function newView () {
            return `Introducir nueva pelicula: <input type="text" id="peli"> <button id="create">Añadir</button>`
        };

        // CONTROLADORES
        function indexContr() { document.getElementById("main").innerHTML = indexView(peliculas);};
        function newContr() { document.getElementById("main").innerHTML = newView();};
        function createContr() {
            peliculas.push(document.getElementById("peli").value);
            indexContr();
        };
        function deleteContr(i) {
            peliculas.splice(i,1);
            indexContr();
        };

        // EVENTOS
        document.addEventListener('DOMContentLoaded', ev => indexContr());
        document.addEventListener('click', ev => {
            if      (ev.target.matches('#new'))    newContr();
            else if (ev.target.matches('#create')) createContr();
            else if (ev.target.matches('#delete')) deleteContr(ev.target.dataset.myId);
        })
    </script>
</head>
<body>
    <h1> Mis películas </h1>
    <div id="main"></div>
</body>
</html>
```

El botón **Borrar** de cada película tiene el atributos id="delete" para identificar el clic en este botón y data-my-id="${i}" con el índice **i** de la película en el array.

deleteContr() borra la película identificada por **i** del modelo y muestra la nueva lista de películas invocando **indexContr()**.

peliculas.splice(i,1) elimina la película indicada por **i** del array del modelo.

indexContr() redirecciona ala primitiva **Index** que muestra la lista de películas una vez actualizado el modelo.

deleteContr() se invoca si id="create".

El índice **i** de la película en el array se extrae de la propiedad **e.target.dataset.myId.**

Bloque **<div** id="**main**"**>** donde se inserta la vista.

El **código resaltado en amarillo** añade la primitiva **Create** al Ejemplo II.

Delete

Ejemplo IV: Mostrar detalles de películas

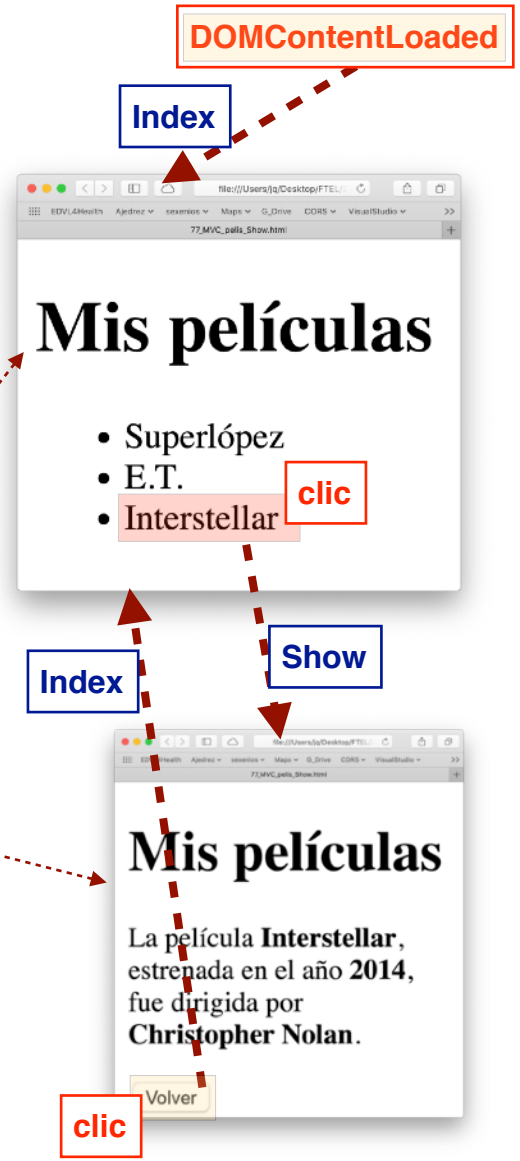
Ejemplo IV: Mostrar detalles de películas

- **Modelo** se enriquece
  - Array de objetos con datos películas, por ejemplo
    - { titulo: **"E.T."**, director: **"Steven Spielberg"**, estreno: **1982**};
- Se añade la primitiva **Show**
  - Para mostrar los detalles de una película

Index es igual que en Ejemplo II, salvo que las películas se hacen **clicables**.

Primitiva	Eventos	Controladores	Vistas
Index: mostrar lista de películas	**DOMContentLoaded** y **click**$_{Volver}$	**indexContr()**	**indexView()**
Show: mostrar película	**click**$_{<pelicula>}$	**showContr(i)**	**showView(i)**

Se añade la primitiva **Show**



- **Index**
  - Muestra la lista de recursos (películas)
- **Show**
  - Muestra los detalles de un recurso (película)

Ejemplo IV: Primitiva Index

```
<!doctype html><html><head><meta charset="utf-8">
    <script type="text/javascript">
        // MODELO
        var peliculas = [
            {titulo: "Superlópez",   director: "Javier Ruiz Caldera", estreno: "2018"},
            {titulo: "E.T.",         director: "Steven Spielberg",    estreno: "1982"},
            {titulo: "Interstellar", director: "Christopher Nolan",   estreno: "2014"}
        ];

        // VISTAS
        function indexView (pelis) {
            var    i=0,   html = "<ul>";
            while(i < pelis.length) {
              html = html + `<li id="show" data-my-id="${i}"> ${pelis[i].titulo}</li>`;
              i = i + 1;
            };
            return html + "</ul>";
        };

        function showView (peli) {
            return `
            La película <b> ${peli.titulo}</b>, estrenada
            en el año <b> ${peli.estreno}</b>, fue
            dirigida por <b> ${peli.director}</b>.

            <p><button id="index"> Volver </button></p>`
        }

        // CONTROLADORES
        function indexContr() { document.getElementById("main").innerHTML = indexView(peliculas);};
        function showContr(i) { document.getElementById("main").innerHTML = showView(peliculas[i]);};

        // EVENTOS
        document.addEventListener('DOMContentLoaded', ev => indexContr());
        document.addEventListener('click', ev => {
            if      (ev.target.matches('#index')) indexContr();
            else if (ev.target.matches('#show'))  showContr(ev.target.dataset.myId);
        })
    </script>
</head>
<body>
    <h1> Mis películas </h1>
    <div id="main"></div>
</body>
</html>
```

Modelo: array de objetos con datos de películas.

indexView (pelis) genera el HTML de la lista de títulos de películas. Cada título tiene un evento **onclick**, para que al clicar en él, muestre los detalles de la película.

Botón con atributo **id="index"**, que genera un evento capturado en el router de eventos.

indexContr() muestra la lista de películas utilizando la vista **indexView(pelis)**.

El evento **DOMContentLoaded** es atendido con **indexContr()**.

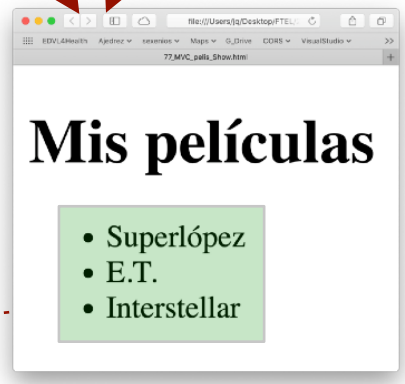
Clicar en el botón **Volver** es atendido con **indexContr()**.

Bloque ** ** donde se inserta la vista.

DOMContentLoaded

clic

Index



Ejemplo IV: Primitiva Show

```
<!doctype html><html><head><meta charset="utf-8">
    <script type="text/javascript">
        // MODELO
        var peliculas = [
            {titulo: "Superlópez",   director: "Javier Ruiz Caldera", estreno: "2018"},
            {titulo: "E.T.",         director: "Steven Spielberg",    estreno: "1982"},
            {titulo: "Interstellar", director: "Christopher Nolan",   estreno: "2014"}
        ];

        // VISTAS
        function indexView (pelis) {
            var    i=0,   html = "<ul>";
            while(i < pelis.length) {
              html = html + `<li id="show" data-my-id="${i}"> ${pelis[i].titulo}</li>`;
              i = i + 1;
            };
            return html + "</ul>";
        };

        function showView (peli) {
            return `
            La película <b> ${peli.titulo}</b>, estrenada
            en el año <b> ${peli.estreno}</b>, fue
            dirigida por <b> ${peli.director}</b>.

            <p><button id="index"> Volver </button></p>`
        }

        // CONTROLADORES
        function indexContr() { document.getElementById("main").innerHTML = indexView(peliculas);};
        function showContr(i) { document.getElementById("main").innerHTML = showView(peliculas[i]);};

        // EVENTOS
        document.addEventListener('DOMContentLoaded', ev => indexContr());
        document.addEventListener('click', ev => {
            if      (ev.target.matches('#index')) indexContr();
            else if (ev.target.matches('#show'))  showContr(ev.target.dataset.myId);
        })
    </script>
</head>
<body>
    <h1> Mis películas </h1>
    <div id="main"></div>
</body>
</html>
```

Modelo: array de objetos con datos de películas.

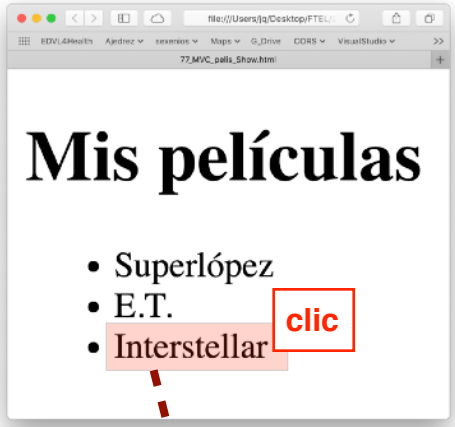
Título con atributos **id="show"** y **data-my-id="i"**, que genera un evento capturado por el router de eventos.

showView (peli) muestra los detalles de la película. Incluye el botón **Volver**, que al clicar invoca la primitiva **Index**, para volver a mostrar la lista de películas.

showContr() muestra los detalles de la película con índice **i**, utilizando la vista **showView(peli)**.

El evento generado al clicar en el título de una película es atendido aquí invocando **showContr(i)**, donde **i** se extrae del atributo **data-my-id="i"**.

Bloque ** ** donde se inserta la vista.



Show

Las 7 primitivas del interfaz CRUD

Interfaz CRUD Web: las 7 primitivas de gestión de una lista

- **Index** y **Show** muestran la lista de recursos (Index) o los detalles de un recurso (Show)
  - Primitivas **R**ead: muestran al usuario datos del modelo (El Ejemplo III solo tiene Index)
- **New** y **Create** añaden un nuevo recurso al modelo
  - Primitivas **C**reate: primero se muestra el formulario (New) y luego se actualiza el modelo (Create)
- **Edit** y **Update** modifican el contenido de un recurso en el modelo
  - Primitivas **U**pdate: mostrar el formulario (Edit) y luego actualizar el modelo (Update)
- **Delete** elimina un recuso del modelo
  - Primitiva **D**elete: suele pedir confirmación antes de eliminar el recurso

Primitiva	Eventos	Controladores	Vistas
Index: mostrar lista de recursos (películas)	**DOMContentLoaded y onclick$_{Volver}$**	**indexContr()**	**indexView()**
Show: mostrar recurso	**click$_{<pelicula>}$**	**showContr(i)**	**showView(i)**
New: mostrar formulario de creación de nuevo recurso	**click$_{Crear}$**	**newContr()**	**newView()**
Create: añadir nuevo recurso al modelo (película)	**click$_{Añadir}$**	**createContr()**	**--** (redirección a Show o Index)
Edit: mostrar formulario de edición de recurso existente	**click$_{Editar}$**	**editContr(i)**	**editView(i)**
Update: actualizar recurso existente en el modelo	**click$_{Actualizar}$**	**updateContr(i)**	**--** (redirección a Show o Index)
Delete: borrar un recurso del modelo (película)	**click$_{Borrar}$**	**deleteContr(i)**	**--** (redirección a Index)

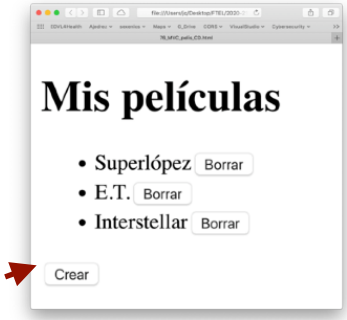
Index se ilustra en Ejemplos II, III y IV.

Show se ilustra en Ejemplo IV.

New y **Create** se ilustran en Ejemplo III.

Edit y **Update** no se ilustran en ningún Ejemplo.

Delete se ilustra en Ejemplo III.



JSON: JavaScript Object Notation

Juan Quemada, DIT - UPM
Santiago Pavón, DIT - UPM

Serialización de datos: JSON

- **Serialización de datos**
  - transformación **reversible** de valores en un string equivalente
  - Facilita el almacenamiento y envío de datos, por ejemplo
    - Almacenar datos en un fichero
    - Enviar datos a través de una línea de comunicación
    - Paso de parámetros en interfaces REST
- **JSON** - JavaScript Object Notation
  - Formato de serialización de valores y objetos JavaScript
    - http://json.org/json-es.html
- Existen otros formatos de serialización: XML, HTML, XDR(C), ...
  - Estos formatos están siendo desplazados por JSON, incluso XML
    - Existen bibliotecas de JSON para los lenguajes más importantes

Objeto global JSON

- JavaScript tiene el objeto global JSON con métodos de conversión
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON
- **JSON.stringify(<object>)**
  - El método **stringify** transforma un objeto (**<object>)** en un string JSON equivalente
    - Lanza excepciones cuando hay errores
- **JSON.parse(<string>)**
  - El método **parse** transforma un **<string>** JSON en el objeto o valor equivalente
    - Lanza excepciones cuando hay errores

```
JSON.stringify(null)                    // =>   'null'
JSON.parse('null')                      // =>    null

JSON.stringify(127)                     // =>   '127'

JSON.stringify('hola')                  // =>   '"hola"'

JSON.stringify([1, 2, 3])               // =>   '[1, 2, 3]'

JSON.stringify({a:27, b:"hola"})        // =>   '{"a":27,"b":"hola"}'
```

Características de JSON

- JSON puede serializar
  - objetos, arrays, strings, números finitos, true, false y null
    - NaN, Infinity y -Infinity se serializan por defecto a null
    - Los objetos Date se serializan como un string en formato ISO 8601
      - la reconstrucción devuelve un string y no el objeto Date original que se serializó
  - No se puede serializar
    - Funciones, RegExp, errores, undefined
- parse y stringify admiten filtros para los elementos no soportados
  - ver doc de APIs JavaScript:
    - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON

```
JSON.stringify(new Date())      => '"2013-08-08T17:13:10.751Z"'

JSON.stringify(NaN)             => 'null'

JSON.stringify(Infinity)        => 'null'
```

Ejemplo de datos en JSON

- **JSON** es un formato flexible y legible de datos muy utilizado
  - permite insertar espacios en blanco y retorno de línea entre los símbolos
    - El siguiente ejemplo muestra un array con 4 objetos en JSON

```
'[ {  "title": "E.T.",
      "director": "Steven Spielberg"
   },
   {   "title": "Star Wars",
       "director": "George Lucas"
   },
   {   "title": "Psicosis",
       "director": "Alfred Hitchcock"
   },
   {   "title": "Placido",
       "director": "Luis García Berlanga"
   }
]'
```

Boolean y operadores relacionados: !, && y ||, ===, !==, <, >, >=, <=, -?-:-

Juan Quemada, DIT - UPM

El tipo boolean y operadores asociados

- El tipo **boolean** tiene dos valores: **true** y **false**

```
Boolean("")     => false
Boolean(4)      => true
```

- **Boolean(a)** es la función que convierte otros valores a boolean
  - **0, -0, NaN, null, undefined, "", '', `` se convierten a **false** y el **resto** a **true**
    - **truthy** o **falsy**: denominación de los valores equivalentes a **true** o **false**

- **!x** (operador negación)
  - Convierte **x** a booleano y lo invierte

```
!!""      => false
!!4       => true
```

```
!4        => false
!"4"      => false
```

```
!null     => true
!0        => true
```

- **x && y** (operador: *y*; inglés: *and*)

  x && y => **y** (si **Boolean(x) === true**)
  => **x** (si **Boolean(x) === false**)

```
true  && true    => true
true && false    => false
undefined && x   => undefined
0 && true        => 0
1 && "5"         => "5"
```

!No convierte a booleano!

- **x || y** (operador: *o*; inglés: *or*)

  x || y => **x** (si **Boolean(x) === true**)
  => **y** (si **Boolean(x) === false**)

```
true  || false   => true
false || false   => false
undefined || 1   => 1
13 || 1          => 13
```

!No convierte a booleano!

- **z ? x : y** (operador ternario condicional)

  z ? x : y => **x** (si **Boolean(z) === true**)
  => **y** (si **Boolean(z) === false**)

```
true  ? 1 : 7    => 1
 3    ? 1 : 7    => 1
false ? 1 : 7    => 7
 ""   ? 1 : 7    => 7
```

!No convierte a booleano!

Mas info: https://javascript.info/logical-operators

Comparación e identidad de valores

- **<, <=, >, >=** (comparaciones)
  - Se suele utilizar solo con number

```
1.2 < 1.3    => true
1e2 < 1e3    => true
1 <  1       => false
```

```
1 <= 1       => true
1 >  1       => false
1 >= 1       => true
```

- **===, !==** (identidad y no identidad)
  - Determina (devuelve true) si **2 valores** son iguales y del mismo tipo
    - Indica igualdad de valores solo con: **number**, **boolean**, **strings, symbol** y **undefined**
  - Con **objetos** la identidad compara **referencias** (punteros)
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Sameness

```
0 === 0      => true
0 === 0.00   => true
2 === 02     => true
2 === 0x2    => true
0 === 1      => false
```

```
'2' === "2"     => true
'2' === "02"    => false

`` === ""       => true
`` === " "      => false
```

```
let x = {},  y = {};
let z = x;   // copia puntero a x

x === y      => false
x === z      => true
```

! === compara punteros !

- Existe una igualdad débil (**==**, **!=**), pero **no debe utilizarse**

Almacenamiento en cliente: localStorage, sessionStorage y "Same Origin Policy"

Juan Quemada, DIT - UPM

Almacenamiento persistente en el cliente

- JavaScript soporta **persistencia de datos** en el navegador
  - Con los objetos de window: **localStorage** y **sessionStorage**
    - Doc: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage
  - **localStorage** y **sessionStorage** solo permiten crear propiedades que guarden **strings**
- **localStorage**
  - Permite crear **contenedores** de datos **permanentes**
    - Solo se eliminan si se borran desde JavaScript
- **sessionStorage**:
  - Permite crear **contenedores** de datos asociados a la **sesión**
    - **Comienzo de sesión:** apertura de navegador o pestaña
    - **Final de sesión:** cierre de navegador o pestaña

Storage API

- **localStorage** y **sessionStorage** son objetos "especiales"
  - Inicialmente se usaban **propiedades dinámicas** como contenedores de datos
    - Realiza una **conversión automática a string** al asignarles un valor

```
localStorage.s =  'hola';   //          - Crea la contenedor s y le asigna 'hola'
.......
localStorage.s;             //  => 'hola'  - Obtiene el string guardado en el contenedor s
.......
delete localStorage.s;      //           - Elimina el contenedor 's'
```

- Hoy se recomienda usar la Web Storage API

```
localStorage.setItem('s', 'hola');   //          - Crea un contenedor 's' y le asigna 'hola'
.......
localStorage.getItem('s');           //  => 'hola'  - Obtiene el string guardado en 's'
.......
localStorage.removeItem('s');        //           - Elimina el contenedor 's'
.......
localStorage.clear();                //          - Elimina todos los contenedores
```

Same Origin Policy

- Los contenedores de **localStorage** y **sessionStorage**
  - Siguen la política de seguridad **same-origin-policy**
    - Doc:https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy
- El **origen** de un script son el **protocolo**, **dominio** y **puerto** del servidor
  - Un programa solo puede acceder a contenedores creados por otros programas que vinieron del mismo **origen**, es decir del mismo servidor
    - Por lo tanto, un programa solo podrá acceder a los contenedores creados por otros programas descargados de su servidor de origen
- Las razones para utilizar **same origin policy** son:
  - **Seguridad**: un programa solo confía en programas del mismo servidor
  - **Modularidad**: cada servidor tiene un espacio de nombres diferente

Ejemplo de localStorage (propiedades)

- Ejemplo que cuenta visitas
  - Contenedor creado con propiedades dinámicas


```
<!DOCTYPE html>
<html><head><meta charset="UTF-8">
<script type="text/javascript">

  localStorage.cuenta = (localStorage.cuenta || 0);

  let c = ++localStorage.cuenta;

  // Evento inicial
  document.addEventListener('DOMContentLoaded',
    () => document.getElementById('cuenta').innerHTML = c
  )
</script>
</head><body>
  <h3>Ejemplo de localStorage</h3>

  Número de visitas a esta página: <span id='cuenta'></span>!
</body>
</html>
```

Crea el contenedor **cuenta** de localStorage (solo si no existe) y lo inicializa con 0. Si existe no altera su valor.

Incrementa el contenido del contenedor **cuenta**. El número de visitas se guarda como un string, pero ++ fuerza la conversión a number.

Inserta el valor del contenedor **cuenta** en el HTML, cuando DOM ya esta construido.

Ejemplo de localStorage (Storage API)

- Ejemplo que cuenta visitas
  - Contenedor creado con Web Storage API


```
<!DOCTYPE html>
<html><head><meta charset="UTF-8">
<script type="text/javascript">

  let cuenta = localStorage.getItem('cuenta') || 0;

  localStorage.setItem('cuenta', ++cuenta);

  // Evento inicial
  document.addEventListener('DOMContentLoaded',
    () => document.getElementById('cuenta').innerHTML = cuenta
  )
</script>
</head><body>
  <h3>Ejemplo de localStorage</h3>

  Número de visitas a esta página: <span id='cuenta'></span>!
</body>
</html>
```

Crea el contenedor **cuenta** de localStorage (solo si no existe) y lo inicializa con 0. Si existe no altera su valor.

Además guarda el valor en una variable cuenta para uso posterior.

Incrementa el contenido del contenedor **cuenta**. El número de visitas se guarda como un string, pero ++ fuerza la conversión a number.

Inserta el valor del contenedor **cuenta** en el HTML, cuando DOM ya esta construido.

Ejemplo de storage API y JSON

- Se añade a Películas
  - Almacenar modelo en localStorage
  - cajetín para añadir películas

```
<!doctype html><html><head><meta charset="utf-8">
    <script type="text/javascript">

    // Contenedor 'c' de localStorage
    const inicial = JSON.stringify(["Superlópez", "Jurassic Park", "Interstellar"]);
    localStorage.setItem('c', localStorage.getItem('c') || inicial);

    // MODELO
    let mis_peliculas = JSON.parse(localStorage.getItem('c'));

    // VISTAS
    function showView () {
        let    i=0,    view = "";

        while(i < mis_peliculas.length) {
          view += `
              <li> ${mis_peliculas[i++]}</li>`;
        };

        return view;
    };

    // CONTROLADORES
    function showContr() {
        document.getElementById('lista').innerHTML = showView();
    };

    function addContr() {
        let p = document.getElementById('pelicula').value;
        mis_peliculas.push(p);
        localStorage.setItem('c', JSON.stringify(mis_peliculas));
        document.getElementById('lista').innerHTML = showView();
    };

    // Eventos
    document.addEventListener('click', ev => {
        if (ev.target.matches('#add')) addContr ();
    })

    document.addEventListener('DOMContentLoaded', showContr);
    </script>
</head><body>

    <h1>Películas</h1>
    Mis películas favoritas:
    <ul id="lista"> </ul>

    Añadir película:
    <input type="text" id="pelicula">
    <button id="add">Añadir</button>
</body></html>
```

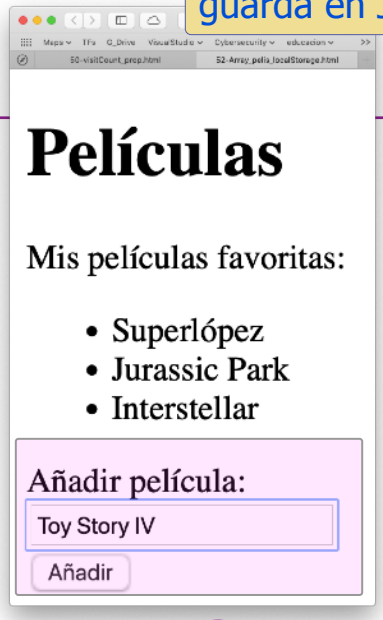
Se crea constante con array inicial en JSON y se crea e iniciativa contenedor (solo si no existe)

se trae el string JSON guardado en localStorage y se transforma en array.

Controlador que añade películas. El array se guarda en JSON.

Se configura evento clic en botón añadir.

Se añade cajetín con botón.



Final del tema