



## T6 - Ejercicios

Departamento de Lenguajes y Sistemas  
Informáticos

Universidad de Sevilla

UNIVERSIDAD DE SEVILLA

# Recordemos...



# El código JavaScript

---

```
function addElement(input) {
    let newItem = document.createElement("LI");
    let newPar = document.createElement("P");
    let remButton = document.createElement("BUTTON");

    remButton.innerHTML="X";
    remButton.addEventListener("click", removeElement);
    newPar.innerHTML = input.value;
    newPar.appendChild(remButton);
    newItem.appendChild(newPar);
    document.getElementById("lista").appendChild(newItem);
    document.getElementById("texto").value = "";
}

function removeElement(event) {
    let li = event.target.parentNode.parentNode;
    document.getElementById("lista").removeChild(li);
}
```

# Objetivos

---

- ◆ Desarrollar nuevas versiones de la lista de la compra, progresivamente más usables y amigables
- ◆ Aplicar estilos CSS según la metodología vista en clase de laboratorio (Bootstrap)

# Versiones

Versión 1:  
API DOM

Versión 2:  
jQuery

Versión 3:  
Entregable

Nuevo elemento:

**Lista de la compra:**

<input type="button" value="Eliminar"/>	pan
<input type="button" value="Eliminar"/>	cerveza

Nuevo elemento:

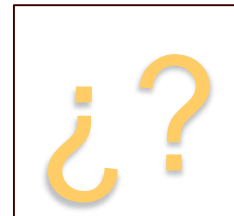
**Lista de la compra**

**Necesito**

pan	<input type="button" value="Eliminar"/>
hielo	<input type="button" value="Eliminar"/>

**Comprados**

pan	<input type="button" value="Restaurar"/>
-----	--



# Versión 1

---

- Requisitos de información:
  - RI1: El sistema deberá almacenar los productos que el usuario tiene pendiente comprar, mediante un texto descriptivo
- Requisitos funcionales:
  - RF1: El usuario podrá ver en todo momento el listado de productos pendientes
  - RF2: El usuario podrá añadir nuevos elementos al listado de productos pendientes, introduciendo el texto descriptivo del producto.
  - RF3: El usuario podrá eliminar cualquier de los elementos del listado.
- Notas:
  - Esta versión de la aplicación se implementará usando HTML, CSS (Bootstrap) y JavaScript

# Notas (I): Modelo

---

- ◆ Introducimos una clase MyList para modelar la lista de productos en memoria (en principio independiente de la lista de elementos HTML, aunque esta última va a reflejar la primera)

```
'use strict';

class MyList {

    constructor() {
        this.items = [];
    }

    addItem(element) {
        this.items.push(element);
    }

    removeItem(element) {
        this.items.splice(element,1);
    }
}
```

## Notas (II): Referencias

---

- ◆ Necesitamos agregar al elemento <head> las referencias necesarias para poder usar los estilos Bootstrap ya conocidos:

```
<head>
  <meta charset="utf8">
  <title>Ejemplos javascript</title>
  <!-- Scripts -->
  <script type="text/javascript" src="js/MyList.js"></script>
  <script type="text/javascript" src="js/main.js"></script>
  <!-- Styles -->
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.
com/bootstrap/4.4.1/css/bootstrap.min.css"
      crossorigin="anonymous">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/aj
ax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
</head>
```



# Notas (III): Estilos

---

- ◆ Usaremos el layout de tarjetas (Cards) de Bootstrap para organizar los contenidos:
  - ◆ `card-header`: Contenedor del cuadro de texto y el botón
  - ◆ `card-body`: Contenedor de la lista
  - ◆ `card-title`: Título (“Lista de la compra”)
  - ◆ `list-group list-group-flush`: Lista (`<ul>`)
  - ◆ `list-group-item list-group-item-info`: Elementos de la lista (`<li>`)
  - ◆ `btn btn-info`: Botón de añadir
  - ◆ `btn btn-outline-secondary btn-sm`: Botones de eliminar

## Notas (IV): Los manejadores de eventos

---

```
function addToShoppingList() {  
    list.addItem(txtItem.value);  
    txtItem.value= '';  
    refresh();  
}
```

- 1) Actualizar modelo
- 2) Actualizar interfaz

```
function removeFromShoppingList(e) {  
    list.removeItem(event.target.parentNode.id);  
    refresh();  
}
```

# Notas (V): Relación entre el modelo y la interfaz

---

- ♦ La lista del modelo (MyList) contiene los productos que el usuario va añadiendo, y se actualiza cuando se eliminan productos.
- ♦ Una vez que la lista del modelo se ha actualizado, hay que reflejar los cambios en la lista HTML de la interfaz, mediante el método refresh()

```
function refresh() {  
  lstItems.innerHTML = '';  
  let index = 0;  
  for (let i = 0; i < list.items.length; i++) {  
    let item = document.createElement('li');  
    let btnRemove = document.createElement('button');  
    item.id=index++;  
    item.className='list-group-item list-group-item-info';  
    btnRemove.innerText = 'Eliminar';  
    btnRemove.onclick = removeFromShoppingList;  
    btnRemove.className='btn btn-outline-secondary btn-sm';  
    item.append(btnRemove);  
    item.append(list.items[i]);  
    lstItems.appendChild(item);  
  }  
}
```

# Versión 2: Modificaciones

---

- Requisitos de información:
  - RI1: El sistema deberá almacenar los productos que el usuario tiene pendiente comprar, mediante un texto descriptivo
  - **RI2: El sistema deberá almacenar los productos que el usuario ya ha comprado.**
- Requisitos funcionales:
  - RF1: El usuario podrá ver en todo momento el listado de productos pendientes y **productos comprados**
  - RF2: El usuario podrá añadir nuevos elementos al listado de productos pendientes, introduciendo el texto descriptivo del producto.
  - RF3: El usuario podrá eliminar cualquiera de los elementos del listado de productos pendientes. **Los productos eliminados pasarán al listado de productos comprados.**
  - **RF4: El usuario podrá restaurar cualquiera de los productos del listado de productos comprados. Los productos restaurados pasarán al listado de productos pendientes.**
- Notas:
  - Esta versión de la aplicación se implementará usando HTML, CSS (Bootstrap), JavaScript y **JQuery**.

# Notas (I): Referencias

---

- ◆ Necesitamos agregar la referencia para usar JQuery:

```
<head>
  <meta charset="utf8">
  <title>Ejemplos javascript</title>
  <!-- Scripts -->
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
  <script type="text/javascript" src="js/MyList.js"></script>
  <script type="text/javascript" src="js/main.js"></script>
  <!-- Styles -->
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css"
        crossorigin="anonymous">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
</head>
```

# Notas (II): Estilos

---

- ◆ Usaremos el mismo layout de tarjetas de la versión anterior, pero distintos estilos:
  - ◆ `card-header`: Contenedor del cuadro de texto y el botón
  - ◆ `card-body`: Contenedor de cada lista
  - ◆ `card-title`: Título (“Lista de la compra”)
  - ◆ `list-group list-group-flush`: Lista (`<ul>`)
  - ◆ `list-group-item list-group-item-danger`: Elementos de la lista de pendientes
  - ◆ `list-group-item list-group-item-secondary`: Elementos de la lista de comprados
  - ◆ `btn btn-danger`: Botón de añadir
  - ◆ `btn btn-outline-secondary btn-sm`: Botones de eliminar
  - ◆ `btn btn-outline-danger btn-sm`: Botones de restaurar

# Notas (III): Dos listas

---

```
<div class="card-body">
  <h3 class="card-title">Lista de la compra</h3>
</div>
<div class="card-body">
  <h4>Necesito</h4>
  <ul id="shoppingList" class="list-group list-group-flush">
    <!--Items will be added here -->
  </ul>
</div>
<div class="card-body">
  <h4>Comprados</h4>
  <ul id="removedList" class="list-group list-group-flush">
    <!--Removed items will be added here -->
  </ul>
</div>
```

# Notas (IV): Los manejadores de eventos

---

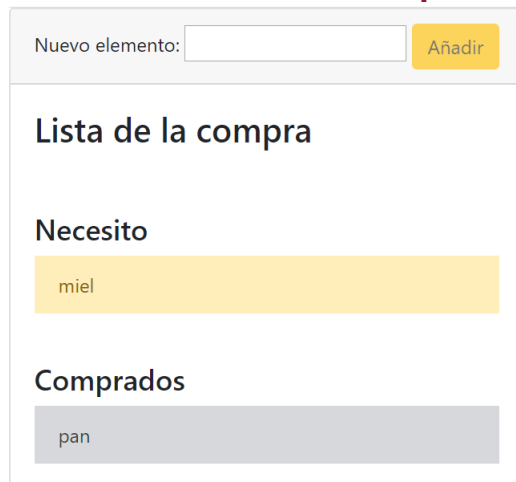
```
function addElement(e) {
    let text = $('#itemToAdd').val();
    shoppingList.addItem(text);
    refresh('#shoppingList');
    $('#itemToAdd').val('');
}
function removeElement() {
    shoppingList.removeItem(event.target.parentNode.id);
    let textValue = event.target.parentNode.childNodes[0].textContent;
    removedList.addItem(textValue);
    refresh('#removedList');
    refresh('#shoppingList');
}
function restoreElement() {
    removedList.removeItem(event.target.parentNode.id);
    let textValue = event.target.parentNode.childNodes[0].textContent;
    shoppingList.addItem(textValue);
    refresh('#removedList');
    refresh('#shoppingList');
}
```



# Versión 3: Modificaciones

---

1. Implementar una versión de la lista de la compra en la que eliminar/restaurar productos de la lista se haga interactuando directamente con el texto del elemento (sin botones)
2. Modificar el código para que el botón Añadir permanezca desactivado mientras que el usuario no ha introducido ningún producto
3. Optimizar la función `refresh()` para evitar tener que repintar toda la lista cada vez que se añade/elimina un objeto



Nuevo elemento:  Añadir

**Lista de la compra**

**Necesito**

miel

**Comprados**

pan

El estilo es orientativo,  
pero no podrá ser  
exactamente el mismo  
que los dos anteriores



**druiz@us.es**  
**inmahernandez@us.es**

**Departamento de Lenguajes y Sistemas  
Informáticos**  
**Universidad de Sevilla**

**UNIVERSIDAD DE SEVILLA**