Hoja de ejercicios Tema 7

Javascript en el cliente: jQuery

▷ **1.** Al igual que en otros muchos lenguajes, una matriz bidimensional puede representarse en Javascript mediante un array cuyas componentes son también arrays:

```
let m = [
            [ "Esto", "es"  , "una fila"  ],
            [ "aquí", "va"  , "otra fila" ],
            [ "y"   , "aquí", "otra más"  ]
        ];
```

(a). Escribe una función `insertMatrix(selector, matriz)` que añada la `matriz` pasada como parámetro en los elementos indicados por el `selector`. La matriz debe representarse mediante un elemento `<table>`, en el cada uno de sus elementos `<td>` contenga el elemento correspondiente de la matriz:

Esto	es	una fila
aquí	va	otra fila
y	aquí	otra más

(b). Modifica la función del apartado anterior para que reciba, además de los dos parámetros anteriores, una función *callback*. Cada vez que el usuario haga clic en una celda de la tabla mostrada anteriormente, se llamará a la función *callback* pasándole como parámetro el elemento de la matriz seleccionado.

▷ **2.** Supongamos un formulario que tiene solamente entradas de texto (es decir, no tiene *check boxes* ni *radio buttons*). Un formulario de este tipo puede representarse mediante un array con tantos objetos como entradas de texto tenga el formulario:

```
let miFormulario = [
    {
        label: "Nombre",
        property: "nombre"
    },
    {
        label: "Apellidos",
        property: "apellidos"
    },
    {
        label: "Edad",
        property: "edad"
    }
];
```

Cada uno de los objetos del array contiene un atributo `label` que almacena la descripción del campo del formulario, y un atributo `property` que almacena el valor que tendrá la propiedad `name` del elemento `<input>` correspondiente.

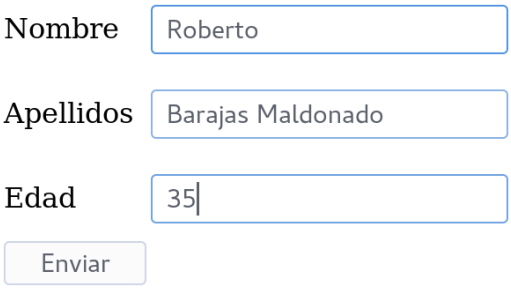
(a). Escribe una función `embedForm(selector, formulario)` que genere dinámicamente los componentes del `formulario` pasado como parámetro, suponiendo que este último venga en el formato de array de objetos mostrado anteriormente. Por ejemplo, si pasamos como segundo parámetro la variable `miFormulario` a esta función, debe generarse dinámicamente el siguiente subárbol DOM,

```
<div style="display: table;">
   <div style="display: table-row;">
      <div style="display: table-cell;">Nombre</div>
      <div style="display: table-cell; padding: 10px;">
         <input name="nombre" type="text">
      </div>
   </div>
   <div style="display: table-row;">
      <div style="display: table-cell;">Apellidos</div>
      <div style="display: table-cell; padding: 10px;">
         <input name="apellidos" type="text">
      </div>
   </div>
   <div style="display: table-row;">
      <div style="display: table-cell;">Edad</div>
      <div style="display: table-cell; padding: 10px;">
         <input name="edad" type="text">
      </div>
   </div>
</div>
```

que deberá insertarse dentro del elemento determinado por el `selector`:

(b). Modifica la función anterior para que reciba, además de los dos parámetros anteriores, una función *callback*. La función debe generar, además de los cuadros de texto, un botón que, al pulsarse, llame a la función *callback* pasándole un objeto que contenga la información del formulario. Los atributos del objeto han de coincidir con los atributos `property` de la representación del formulario.

Por ejemplo, si el usuario ha introducido la siguiente información:



Al pulsar el botón `[Enviar]`, la función *callback* deberá ser llamada con el siguiente objeto como parámetro:

```
{
    nombre: "Roberto",
    apellidos: "Barajas Maldonado",
    edad: "35"
}
```

(c). Extiende la solución del apartado anterior para que cada uno de los objetos del array que describe el formulario, además de los atributos `label` y `property` mostrados anteriormente, pueda contener un atributo `validator` cuyo valor sea una función booleana que indique si el valor introducido en el formulario es correcto. Por ejemplo:

```
let miFormulario = [
    ...
    {
        label: "Edad",
        property: "edad",
        validator: function(val) {
                        // Devuelve true si el valor pasado
                        // como parámetro es un número
                        return !isNaN(Number(val));
                    }
    }
];
```

Al pulsar el botón de *Enviar* se comprobará que la función validadora (en aquellos campos que la tengan) devuelva `true`. Si esto no es así para alguno de los campos del formulario, se mostrará un mensaje (`alert()`) con las etiquetas de los componentes del formulario que no hayan sido aceptados por la función validadora.

▷ **3.** Modifica el ejercicio 1 para que, además de visualizarse una matriz, pueda editarse. Al hacer clic en un elemento de la matriz, se sustituirá el elemento correspondiente por un cuadro de texto en el que el usuario puede cambiar el valor del elemento correspondiente de la matriz.

Esto	es	una fila
aquí	va	otra fila
y	aquí	la celda final

Cuando el cuadro de texto pierda el foco, se guardará el valor introducido en la posición de la matriz correspondiente, y se mostrará en la tabla correspondiente.
Indicación: Utiliza el evento `blur` para determinar cuándo un cuadro de texto pierde el foco. Utiliza el método `focus()` para hacer que un cuadro de texto adquiera el foco.

▷ **4.** En el Campus Virtual encontrarás una página HTML que simula una sencilla calculadora. Implementa su funcionalidad utilizando *jQuery*. El usuario introducirá el primer operando mediante los botones numéricos. Después seleccionará una de las operaciones (suma, resta, multiplicación, división) y, de nuevo, utilizará los botones numéricos para introducir el segundo operando. Al pulsar el botón = el display superior de la calculadora debe mostrar el resultado de aplicar la operación seleccionada a ambos operandos. Si, en cualquier momento, se pulsa el botón C, la calculadora volverá a mostrar el valor 0, y el usuario tendrá que comenzar de nuevo introduciendo el primer operando.

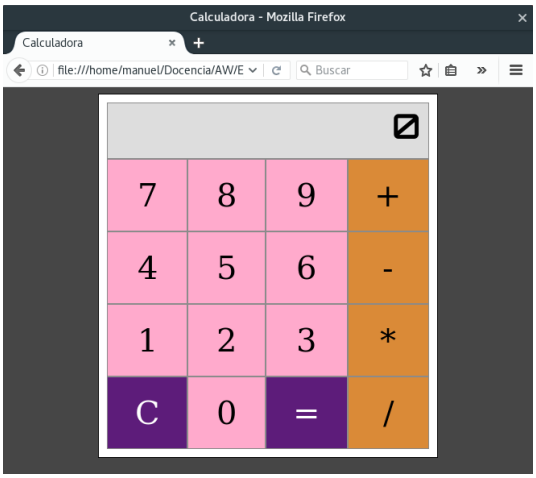


Figura 1: Aplicación calculadora