Tabla de contenido

Formularios, objeto form.

La mayor interactividad de los usuarios se produce a la hora de introducir información en los controles de formulario. JS permite acceder tanto al formulario como a cada control de una página web y establece para cada uno de estos objetos una serie de métodos y propiedades cuyo objetivo es poder dotar de dinamismo casi cualquier característica HTML de los mismos. Por lo tanto es importante conocer cómo son estos objetos.

Comencemos por el objeto form. Por cada formulario existente en una página web, <form …> se crea en el DOM un objeto Form. Este objeto está disponible como una propiedad del objeto document en la colección forms (recuerda que si en la misma página web contamos con varios formularios éstos se encontrarán accesibles en la colección **document.forms**).

document.forms[0]: será el primer formulario de la página.

Objetos HTML DOM: Tanto los formularios como todos sus controles (input, select, textarea, …) son considerados objetos del Modelo de Objetos del Documento (DOM). Esto significa que cada objeto tiene propiedades únicas definidas por el DOM. Por ejemplo: los formularios tienen la propiedad exclusiva **action** y los **select** la propiedad **selectedIndex**.

La API básica para un objeto form es la siguiente:

elements[]: Retorna un array con todos los elementos del formulario.

PROPIEDADES

acceptCharset: Permite leer/establecer el atributo accept-charset de un formulario. Los valores más comunes son UTF-8 y ISO-8859-1. Es el juego de caracteres aceptado en el formulario.

action: Permite leer/establecer el atributo action de un formulario.

enctype: Permite leer/establecer el atributo enctype de un formulario. Este atributo indica cómo deben codificarse los datos antes de ser enviados al servidor. Por defecto se usa application/x-www-form-urlencoded que indica que los datos se codifican en formato URL. También se puede usar text/plain para no codificar salvo los espacios en blanco que se cambian por +. Y si estamos subiendo ficheros al servidor debemos desactivar la codificación por lo que usamos multipart/form-data.

length: Devuelve el número de elementos de un formulario.

method: Permite leer/establecer el atributo method de un formulario. Como sabes, los valores pueden ser get o post.

name: Permite leer/establecer el atributo name de un formulario.

target: Permite leer/establecer el atributo target de un formulario. Este atributo indica dónde se abrirá el action del formulario. Los valores aceptados son: _blank (nueva ventana), _self (esta ventana), _parent (marco padre), _top(página principal que define los marcos) o bien el atributo name de un marco existente. MÉTODOS

reset(): Provoca el reseteo (limpieza) de todos los datos en sus controles. Es equivalente a pulsar un <input type=”reset”> dentro del formulario.

submit(): Provoca el envío de información al recurso especificado en el atributo action. Es equivalente a pulsar un <input type=”submit”> dentro el formulario.

EVENTOS

onreset = función: Evento que se dispara al pulsar un botón de tipo reset en el formulario.

onsubmit = función: Evento que se dispara al pulsar un botón de tipo submit en el formulario. Habitualmente se emplea para realizar la validación en el lado del cliente. Si la función retorna true se produce el submit(), en caso contrario no se produce.

Propiedades estándar.

Tanto los formularios como sus controles pueden utilizar también desde JS el conjunto de propiedades estándar. Estas propiedades, que mostramos a continuación son susceptibles de ser utilizadas en todos los elementos HTML, y están definidas por el W3C.

Muchas de estas propiedades son atributos en HTML, como es el caso de **accesskey** o **tabindex**. Por lo tanto podríamos fijarlos en el propio HTML o bien hacerlo mediante la propiedad equivalente en JS. Recuerda que en JS se sigue la notación **lowerCamel**, luego el atributo **accesskey** de html equivale a la propiedad JS **accessKey**

accessKey: Permite leer/modificar el atributo html accesskey, utilizado para indicar una tecla de acceso directo. Ejemplo, si hacemos control.accesskey = “p” podemos colocar el foco directamente en este control pulsando alt+p. También podríamos hacerlo simplement en html colocando el atributo accesskey = “p” en el elemento.
className: Permite leer/cambiar una clase CSS para un elemento. elemento.className = “titulo” aplicará la clase .titulo {…}. Mediante JS podremos cambiar las clases dinámicamente.
dir: Permite leer/establecer la dirección del texto en un elemento. El valor por defecto es “ltr” (left to rigth). También podemos utilizar “rtl” (rigth to left).
id: Permite leer/establecer el atributo id de un elemento.
innerHTML: Permite leer/establecer contenido Html de un elemento.
lang: Permite leer/establecer el atributo lang de un elemento.
tabIndex: Valor numérico que permite leer/establecer el orden de tabulación de un elemento. Existe el atributo html tabindex equivalente. Esto nos permite que los controles obtenga el foco por orden si se pulsa el tabulador en un formulario.
title: Permite leer/establecer el atributo title de un elemento. Cada navegador interpretará el atributo title a su manera. Habitualmente es similar al uso de alt=”” e las imágenes. Se puede agregar a cualquier elemento.

¿Cómo acceder a los elementos de un formulario?

FORMA 1

Podemos identificar un control de formulario utilizando las colecciones. Observa cómo accedemos al primer elemento <input type=”text” name=”Nick”> del primer formulario <`form name=”miform”>` de la página:

```
var formulario = document.forms[0]; // accedemos al primer formulario con la colección forms
```

`var control = formulario.elements[0]; //` accedemos al primer input con la colección elements

FORMA 2

Si nuestros controles de formulario disponen de un atributo name también podríamos combinar este atributo con las colecciones como si fueran arrays asociativos. El ejemplo de la forma anterior, es equivalente a:

`var formulario = document.forms["miform"];` // accedemos al form con name="miform"

`var control = formulario["nick"];` //accedemos al elemento con name="Nick"

FORMA 3

El atributo name también nos permite que accedamos a los elementos pero en este caso especificando toda la jerarquía en el árbol de objetos del DOM. El ejemplo anterior quedaría como:

`var control = document.miform.nick` // accedemos al <input name="Nick"> dentro de miform

Es decir, si utilizamos el atributo html name podemos identificar los elementos mediante la ruta. Así: `document.miform.nick` obtiene el `<input type="text" name="nick">` que está dentro del formulario `<form name="miform">`.

Como ves JS es muy flexible y permite el acceso a los componentes de una página web de múltiples formas. En algunos casos estaremos empleando la API de los objetos y en otros casos estaremos empleando el Modelo de Objetos del Documento (DOM).

Controles de formulario.

La mayor interactividad en las páginas web tiene lugar cuando el usuario hace uso de los controles de formulario para agregar información.

Tanto los formularios como todos sus controles (input, select, textarea, …) son considerados objetos del Modelo de Objetos del Documento (DOM). Esto significa que cada objeto tiene propiedades únicas definidas por el DOM. Por ejemplo: los formularios tienen la propiedad exclusiva action y los select la propiedad exclusiva selectedIndex.

Simplemente recuerda que cada aspecto de HTML de un control tendrá su propiedad o método equivalente en JS, aunque puede que no se llamen exactamente igual.

Objeto input type="text".

Representa una entrada de texto simple. Ejemplo:

```
<input type="text" name="nombre" size="8" maxlength="30" />
```

API de Javascript para este objeto:

accessKey: Permite leer/modificar el atributo html accesskey, utilizado para indicar una tecla de acceso directo. Ejemplo, si hacemos control.accesskey = "p" podemos

colocar el foco directamente en este control pulsando alt+p. También podríamos hacerlo simplement en html colocando el atributo accesskey = "p" en el elemento.
defaultValue: Permite leer/escribir el texto por defecto
disabled: Propiedad booleana. Permite leer/escribir la desactivación del control. Un control desactivado no se envía cuando se acepta el formulario.
form: Devuelve una referencia al formulario que contiene este control.
maxLength: Permite leer/escribir el número máximo de caracteres permitidos en el texto.
name: Permite leer/escribir el valor del atributo name.
readOnly: Propiedad booleana. Permite fijar el control como de sólo lectura. Un control de sólo lectura no puede ser modificado por el usuario, pero sí se envía cuando se acepta el formulario.
size: Permite leer/escribir el ancho del control. Este ancho está reflejado en un número, que será el número de caracteres de texto.
type: Retorna el tipo de control. En los controles de texto este tipo será "**text**".
Otros valores representarán otros controles de tipo <input>. Los más utilizados (ya sabes que HTML5 incluye nuevos tipos) son los siguientes:
"**password**" (cuadro de contraseña), "**checkbox**"(casilla de verificación), "radio" (casilla de opción), "**submit**" (botón de envío del formulario), "**reset**" (botón de reseteo de formulario), "**file**" (botón para seleccionar ficheros), "**hidden**" (campo oculto), "**image**"(botón de imagen), "**button**" (botón genérico).
Ejemplo de type nuevo en HTML5: email.
value: Permite leer/escribir el valor del control (el texto escrito en él).

MÉTODOS

select(): Selecciona el contenido del control.

Otros objetos input.

Hay muy pocas diferencias en el resto de objetos input.

Objeto **input type = "password":** tiene exactamente las mismas propiedades y métodos que el input type="text".

Objetos **input type = "button", input type="submit"** e **input type="reset":** al tratarse de un botones, habitualmente utilizados para programarlos con el evento onclick dispone de un subconjunto de propiedades básicas. Sólo son **disabled, form, name, type** y **value**.

Objetos **input type = "checkbox"** y **input type="radio":** además de las cinco propiedades básicas (disabled, form, name, type y value) disponen de dos propiedades exclusivas, que son:

- **checked**: propiedad booleana para activar/desactivar el control desde JS.
- **defaultChecked**: también propiedad booleana para establecer la activación predeterminada del control desde JS.

Objeto **input type = "file":** además de las cinco propiedades básicas dispone de una propiedad exclusiva, que es:

- **accept**: permite establecer una lista de formatos aceptados separados por comas. Mucho cuidado porque algunas versiones IE y Safari pueden no aceptar esta propiedad. Por este motivo no es muy utilizada.

Objeto **input type = "hidden"**: al tratarse de un campo de texto oculto sólo dispone de las propiedades básicas pero no de disabled (no se puede desactivar). Son por tanto cuatro propiedades: form, name, type y value.

NOTA: recordar algunos aspectos de los input *type="radio"* y de los input *type="checkbox*. La primera cuestión es el valor que se envía con el formulario. Cada control debe tener su propio atributo value diferenciado. Ejemplo:

```
<input type="checkbox" value="PR" name="interes01" id="interes01"/>
```

Si no utilizamos atributos value al enviar el formulario podríamos obtener valores del tipo on/off. Esto no es muy adecuado porque estamos dependiendo de valores que no generamos nosotros mismos (cada servidor podría diferir en los mismos. La segunda cuestión es el uso del atributo name. En el caso de los input type="radio" si queremos que sean excluyentes entre sí tendremos que asignarles el mismo atributo name. No obstante los atributos id sí deben ser diferentes. No deben existir dos elementos con el mismo id.

Objetos select y option.

Representa una lista desplegable formada por objetos <option>:

```
<select name="isla" id="isla" size="3" multpile="multiple">…</select>
```

API de Javascript para este objeto:

PROPIEDADES

options: Se trata de un array o colección que permite acceder al conjunto de opciones definidas dentro del select. Cada opción está formada por un objeto option.
disabled: Propiedad booleana. Permite leer/escribir la desactivación del control. Un control desactivado no se envía cuando se acepta el formulario.
form: Devuelve una referencia al formulario que contiene este control.
length: Retorna el número de opciones que hay dentro del select. También podríamos acceder a este dato utilizando la colección options de la forma options.length.
multiple: Propiedad booleana. Permite activar/desactivar el atributo multiple en el select. Un select multiple permite seleccionar varias opciones a la vez.
name: Permite leer/escribir el valor del atributo name.
selectedIndex: Retorna el índice de la opción seleccionada. Este índice es la posición dentro de la colección option de la opción que esté seleccionada.
size: Permite leer/escribir el alto del control. Representa el número de opciones que será visible a la vez en la representación del select en la pantalla.
type: Retorna el tipo de control. En los objetos select el tipo por defecto es "select-one". Sin embargo si activamos el atributo multiple el tipo del control cambia a "select-multiple".

MÉTODOS

add(opción, antesDe): Permite agregar una opción nueva dinámicamente. Los dos parámetros son obligatorios.

Opción: será un objeto option o bien optgroup.

AntesDe: será una opción existente en el select. La opción se insertará justo antes de la opción indicada. Si indicamos null se insertará al final del select.

Ejemplo: insertamos una opción con el texto "Gato" al final del select con id="Animales"

```
var miselect = document.getElementByID("Animales");
var miopcion = document.createElement("option");
miopcion.text = "Gato";
miopcion.value= 25;
miselect.add(miopcion, null);
```

remove(posicion): Borra la opción cuyo índice en el array options sea posicion.

El objeto select está directamente relacionado con el objeto option. Utilizando la Api del **objeto option** podremos acceder y modificar cada opción en particular:

PROPIEDADES

- **defaultSelected**: Propiedad booleana que retorna true si esta opción está seleccionada por defecto, es decir, si en el código HTML incluye un atributo selected="selected". En caso contrario retorna false.
- **disabled**: Propiedad booleana. Permite leer/escribir la desactivación del control. Un control desactivado no se envía cuando se acepta el formulario.
- **form**: Devuelve una referencia al formulario que contiene este control.
- **index**: Permite leer/modificar el índice o posición de esta opción en el array de opciones de un select.
- **selected**: Propiedad booleana. Permite seleccionar/deseleccionar la opción desde JS. **text**: Permite leer/escribir el texto de la opción.
- **value**: Permite leer/escribir el atributo value de la opción. Recuerda que este atributo value es lo que enviamos cuando se acepte el formulario en caso de encontrarse esta opción seleccionada.

Objeto textarea.

Representa un cuadro de texto multilínea:

```
<textarea name="comentarios" id="comentarios" cols="30",
rows="5">
Aquí su texto
</textarea>
```

Este objeto es bastante parecido a los cuadros de texto simples. De hecho cuenta con los mismos métodos y propiedades que los input type="text". No obstante también cuenta con la propiedad **cols** y **rows** para leer/establecer desde JS estos atributos.

Otros objetos DOM.

El DOM incluye objetos para cualquier etiqueta HTML. Cada uno de estos objetos dispone de un conjunto propio de métodos y propiedades. Aquí te muestro enlaces a algunos de los más utilizados:

Objeto anchor: representa a un enlace o etiqueta <a>. Todos los enlaces de una página están disponibles desde JS mediante la colección document.anchors. También

podríamos acceder a ellos mediante su atributo name o su atributo id. Una guía de referencia sobre su API se encuentra en:
http://www.w3schools.com/jsref/dom_obj_anchor.asp

Objeto area: representa a cada área definida dentro de un mapa sensible o etiqueta <area>. Puedes consultar su API en:
http://www.w3schools.com/jsref/dom_obj_area.asp

Objeto body: representa a la etiqueta <body>. En JS puedes acceder a este objeto mediante document.body, y puedes modificar por ejemplo el color de fondo, el color genérico para los enlaces activo, los enlaces visitados, etc. Puedes consultar su API en:
http://www.w3schools.com/jsref/dom_obj_body.asp

Objeto image: representa a cada etiqueta <img> de una página. Las etiquetas <img> son contenedores de imágenes. Tienen propiedades exclusivas como src para indicar la ruta del fichero a cargar. Puedes consultar su API en:
http://www.w3schools.com/jsref/dom_obj_image.asp

Otros objetos populares, que puedes consultar también en la web de w3schools son frame, link (cada etiqueta link para enlazar ficheros en una página web), meta (cada etiqueta meta), table (cada tabla), etc.

Para terminar citar el **objeto style**. Cualquier objeto del DOM incluye una propiedad style. Esta propiedad nos permite acceder a los estilos CSS del objeto y modificar sus propiedades mediante JS. Eso sí, los nombres en JS no suelen coincidir con las propiedades CSS. Así, por ejemplo, para poner un color de fondo en CSS utilizamos la propiedad background-color mientras que en JS haríamos `objeto.style.backgroundColor = "color"`. Recuerda además que JS es case-sensitive.

Cambiar todas las classes con una nueva o más:
```
document.getElementById("MyElement").className = "MyClass";
```
(Puedes agregar más clases separándolas con un espacio)

Agregar clases adicionales sin afectar las existentes:
```
document.getElementById("MyElement").className += " MyClass";
```

Element.classList

classList es una forma práctica de acceder a la lista de clases de un elemento como una cadena de texto delimitada por espacios a través de element.className

Métodos

add(String [, String]): Añade las clases indicadas. Si estas clases existieran en el atributo del elemento serán ignoradas.
remove(String [, String]): Elimina las clases indicadas.
Nota: Eliminar una clase que no existe NO produce un error.

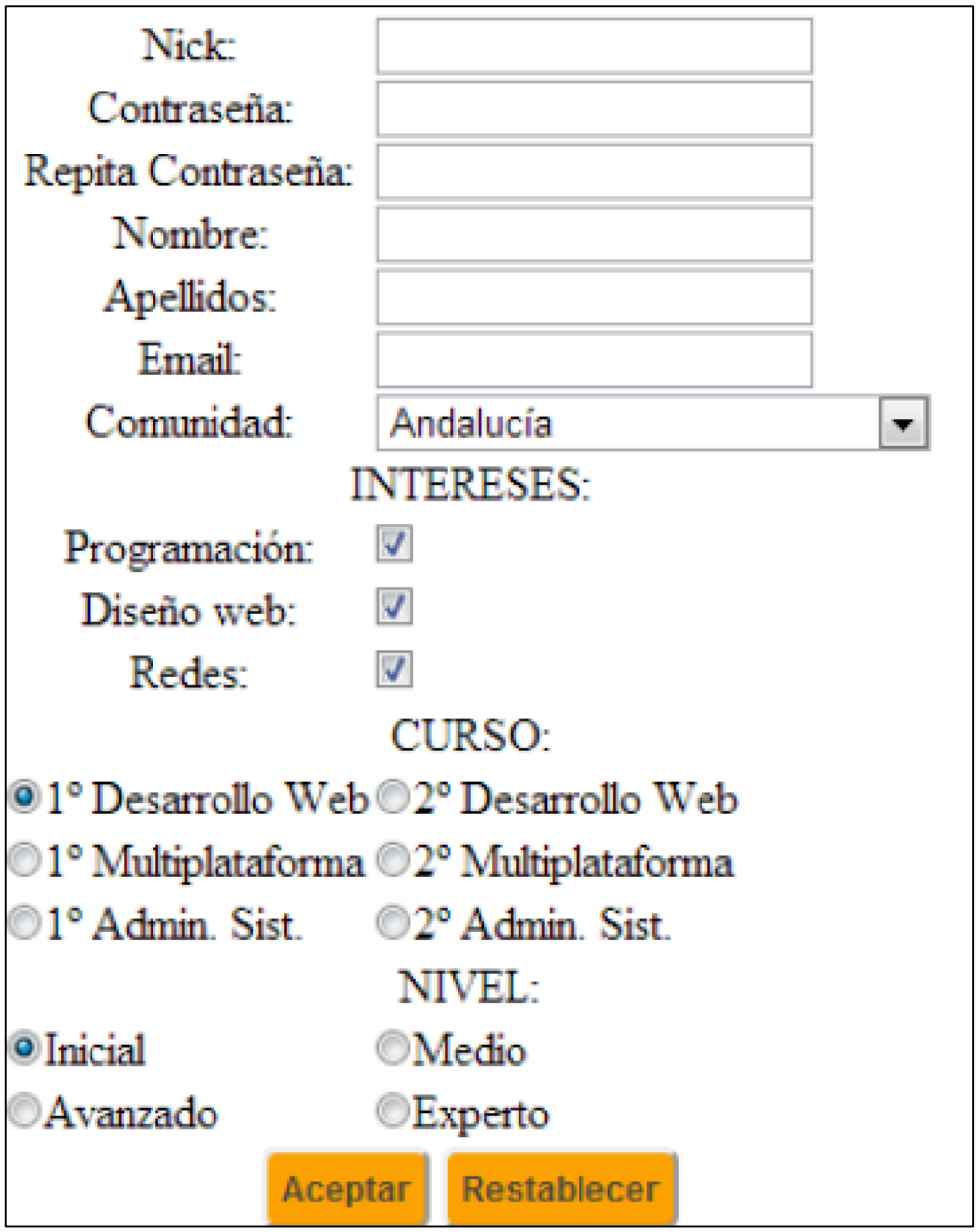
item(Number): Devuelve el valor de la clase por índice en la colección.
toggle(String [, force]): Cuando sólo hay un argumento presente: Alterna el valor de la clase; ej., si la clase existe la elimina y devuelve false, si no, la añade y devuelve true.
Cuando el segundo argumento está presente: Si el segundo argumento se evalúa como true, se añade la clase indicada, y si se evalúa como false, la elimina.
contains(String): Comprueba si la clase indicada existe en el atributo de clase del elemento.
replace(oldClass, newClass): Reemplaza una clase existente por una nueva.

```
//div es una referencia de objeto al elemento <div> con class="foo
bar"
div.classList.remove("foo");
div.classList.add("anotherclass");

// si visible está presente la elimina, de lo contrario la añade
div.classList.toggle("visible");

//añadir/eliminar visible, dependiendo de la condición, i menor que 10
div.classList.toggle("visible", i < 10 );

alert(div.classList.contains("foo"));

// añadir o eliminar varias clases
div.classList.add("foo", "bar");
div.classList.remove("foo", "bar");

// reemplazar la clase "foo" por "bar"
div.classList.replace("foo", "bar");
```

Actividad 1

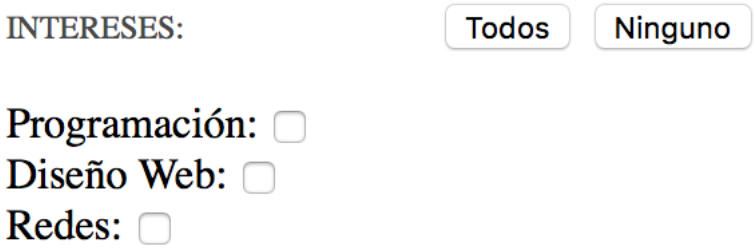
Necesitarás un servidor de páginas web, puedes utilizar aquel con el que te sientas a gusto.

1. Crea una BD para este ejercicio con el nombre Formulario1 y con un cotejamiento igual a utf8_general_ci
2. Crea un usuario con todos los permisos para trabajar sobre esta base de datos.
3. Crea una tabla auxComunidades, rellene ésta con el sql adjunto en la plataforma.
4. Cree un formulario en una página php, la comunidad se obtendrá del servidor y se cargará en el select.

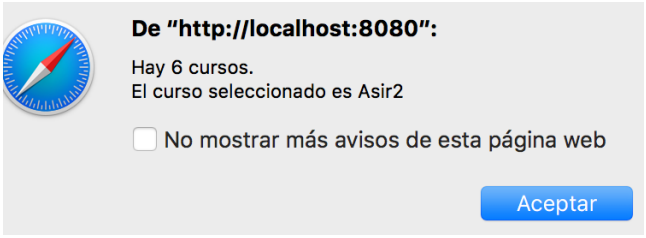


5. Todos los controles dispondrán de los atributos **name** e **id**. Además hay dos grupos de radio excluyentes: un grupo para seleccionar el curso y otro grupo para seleccionar el nivel.
6. Crea una función JS llamada **preparar()**. Su cometido será recorrer los elementos del formulario haciendo lo siguiente [NOTA: La función no podrá utilizar en ningún momento los atributos name o id de los controles del formulario. Utiliza elements]:
    a. A los controles de tipo password les fijará un color de fondo gris #C7C7C7
    b. A los controles de tipo text les fijará un color de letra rojo.
    c. A los controles de tipo checkbox se les desactivará el marcado.
    d. A los controles de tipo radio les colocará un margen de 10px.
    e. A los controles de tipo submit y reset les colocará letra blanca.
    f. La función se disparará en el evento onload.

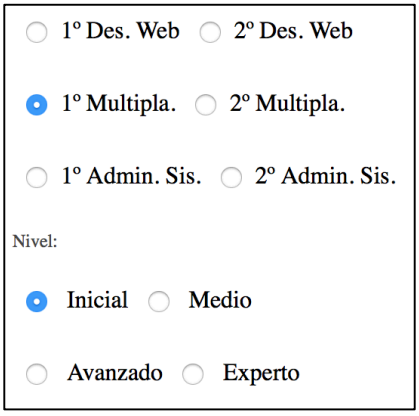
7. Agrega unos botones junto a INTERESES con los valores TODOS y NINGUNO. Cuando se pulsen ambos llamarán a la función **seleccionar(valor)** que recibirá true si queremos marcar los intereses o false en caso contrario.

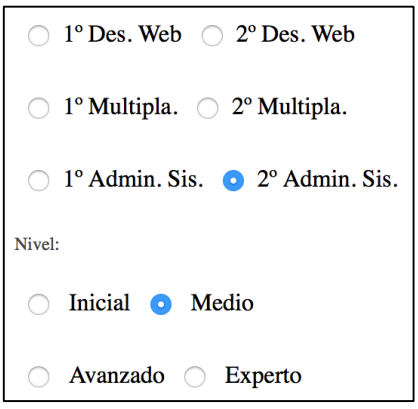


8. Tenemos un conjunto de radios cuyo nombre es curso. Asegúrate de dar los valores siguientes a cada campo (Daw1, Daw2, Dam1, Dam2, Asir1 y Asir2). Ahora coloca a todos estos radio el evento **onchange="cambioCurso();"** Programa la función cambioCurso(). Esta función debe utilizar únicamente accesos por nombre (atributo name). La función mostrará una alerta como la siguiente:

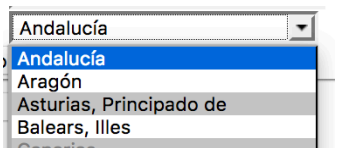


Si además este valor termina por un "2" tendremos que marcar el segundo radio de nivel (el de nivel medio). En caso contrario marcaremos el primer radio de nivel (el de nivel básico). NOOO puede utilizar getElementById().



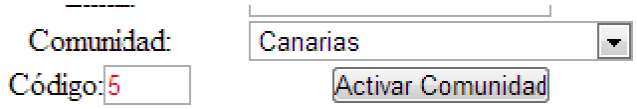


9. Programa la función **estiloCebra()**. Su comedito es colocar un fondo gris claro #C7C7C7 a las comunidades que estén en posiciones impares. Llama a esta función al final de **preconfigurar()**.



10. Programa la función **desactivaOpciones(caracter).** Su cometido será recibir un carácter y desactivar aquellas comunidades que comiencen por el mismo. Para probarla agrega una llamada en **preconfigurar()** como la siguiente: **desactivaOpciones("C");**

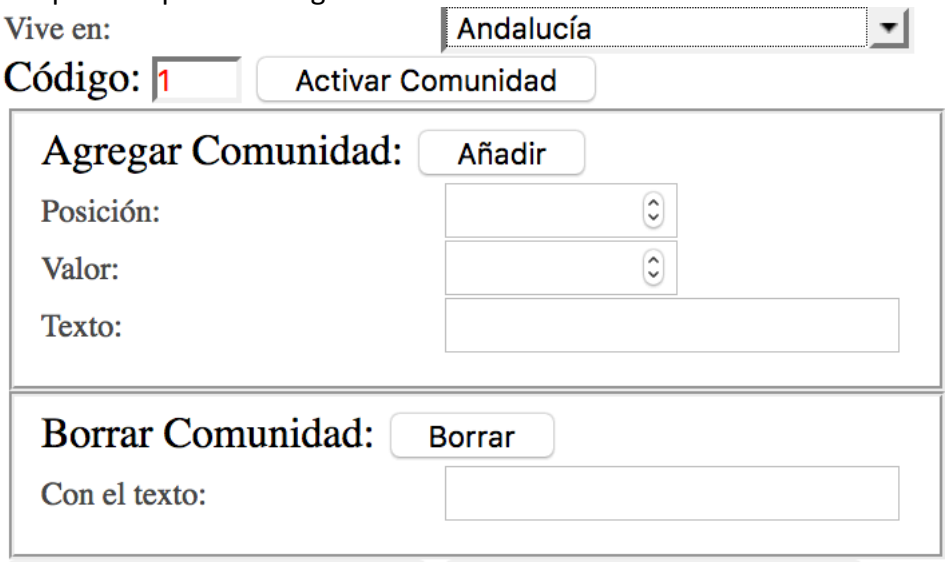
11. Modifica tu formulario para que incluya lo siguiente. Cuando se pulsa "Activar Comunidad" se lee el código escrito y se selecciona automáticamente la comunidad con ese valor en el atributo value. En el ejemplo se ha seleccionado Canarias porque es la que tiene value="5". Programa este efecto.



12. Ahora agregaremos/eliminaremos objetos de nuestra lista desplegable. A la hora de agregar una comunidad tendremos que indicar el valor y el texto. También indicaremos la posición donde queremos insertarla. Si no se indica ninguna posición se insertará por el final.

    NOTA: el parámetro AntesDe del método add() es un objeto option. Si quiero por ejemplo insertar en la posición 2 tendré que obtener una referencia a la opción que esté en la posición 2 y pasarla al método add().

    Por otra parte a la hora de borrar una comunidad sólo pediremos el texto y eliminaremos esa comunidad en concreto. Si la comunidad no está en la lista informaremos con una alerta.
    Comprueba que tus código funcionan revisando el select de comunidad.



Fuentes

http://www.maestrosdelweb.com/que-es-javascript/
http://librosweb.es/libro/javascript/
http://librosweb.es/
http://www.aprenderaprogramar.es/index.php?option=com_content&view=article&id=788:tipos-de-datos-javascript-tipos-primitivos-y-objeto-significado-de-undefined-null-nan-ejemplos-cu01112e-&catid=78:tutorial-basico-programador-web-javascript-desde-&Itemid=206
https://www.todojs.com/ref/javascript/global/
https://jherax.wordpress.com/2014/07/08/javascript-poo-1/