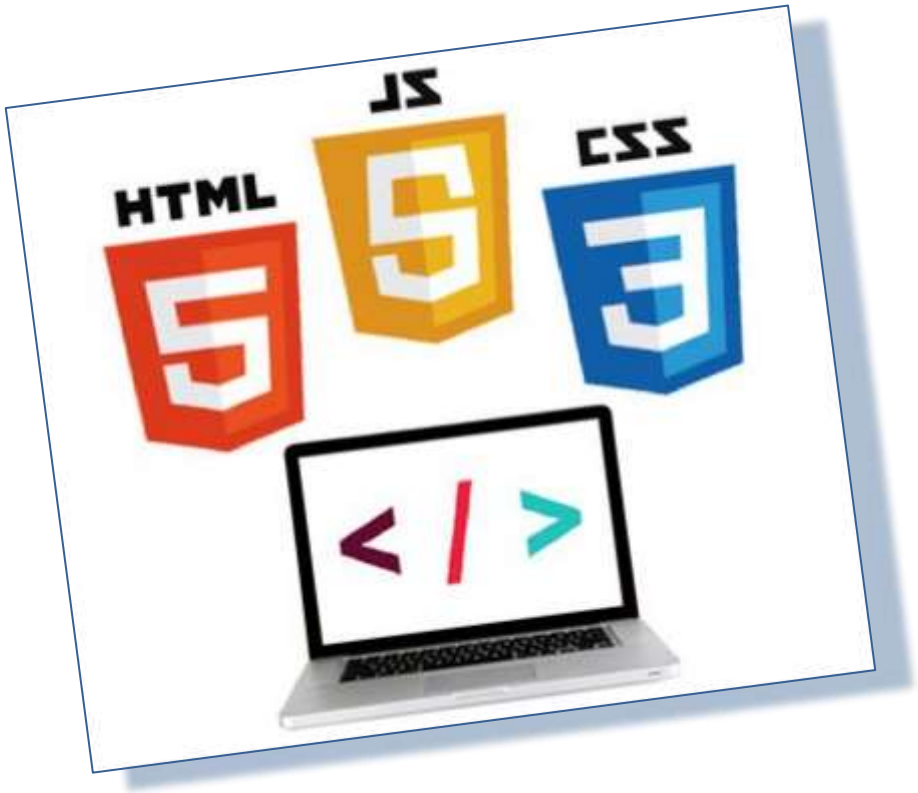
Desarrollo web en entorno cliente



Tema 6: Eventos y Formularios

1. Eventos

Los eventos permiten reaccionar a acciones que realiza el usuario o cambios que suceden en la página. Javascript permite ejecutar código como respuesta a un evento, asociando al mismo una función. Además, una sola acción puede desencadenar más de un evento. Por ejemplo, al pulsar un botón se puede capturar el evento click pero también mousedown, mouseup, etc. Los siguientes son algunos de los eventos más comunes.

Eventos de página: Se producen en el documento HTML, normalmente en el body:

- **load**: cuando termina la carga de la página y está construido el árbol DOM.
- **unload**: al destruirse la página.
- **beforeUnload**: antes de destruirse el documento pudiendo, por ejemplo, mostrar un mensaje de confirmación.
- **resize**: al cambiar el tamaño del documento, al redimensionar la ventana.

Eventos de ratón: Se producen:

- **click** / **dblclick**: cuando se hace click/doble click sobre un elemento.
- **mousedown** / **mouseup**: al pulsar/soltar cualquier botón del ratón.
- **mouseenter** / **mouseleave** o **mouseover/mouseout**: cuando el puntero del ratón entra/sale del elemento (en el segundo caso también aplica a sus descendientes).
- **mousemove**: continuamente mientras el puntero se mueva dentro del elemento.

Eventos de teclado: Se producen:

- **keydown**: al presionar una tecla y se repite mientras la tecla siga pulsada.
- **keyup**: en el momento en que se deja de presionar una tecla.
- **keypress**: cuando se pulsa y suelta una tecla alfanumérica.

Eventos de toque: Se producen al usar una pantalla táctil:

- **touchstart**: cuando se detecta un toque en la pantalla.
- **touchend**: cuando se deja de pulsar la pantalla táctil.
- **touchmove**: cuando un dedo es desplazado a través de la pantalla.
- **touchcancel**: cuando se interrumpe un evento táctil.

Eventos de formulario: Se producen en los formularios:

- **focus / blur**: al obtener/perder el foco un elemento.
- **change**: al perder el foco un <input> o <textarea> si ha cambiado su contenido o al cambiar de valor un <select> o un <checkbox>.
- **input**: al cambiar el valor de un <input> o <textarea> (por cada letra)
- **select**: al cambiar el valor de un <select> o al seleccionar texto de un <input> o <textarea>
- **submit** / **reset**: al enviar/recargar un formulario.

1.1. *Registro de eventos en línea*

La opción más sencilla pero menos recomendable, es registrar un evento en línea. Es decir, se trata de incluir directamente javascript en las etiquetas de elementos HTML. Se hace añadiendo un atributo con el nombre del evento(precedido por la palabra on) a escuchar en el elemento HTML que lo vaya a producir, asociando en ese mismo punto el código a ejecutar.

Por ejemplo, para ejecutar cierto código al producirse el evento 'click' sobre un botón:

```
<input type="button" id="botonAceptar" onclick="alert('Aceptar');"/>
<h1 onclick="this.innerHTML='Nuevo Texto'"
onmouseover="this.style.background='blue'"
onmouseout="this.style.background='white'">Elemento de ejemplo</h1>
```

Una mejor práctica es asociar una llamada a una función en lugar del código directamente:

```
<input type="button" id=" botonAceptar " onclick="aceptar()" />
<script>
    function aceptar() {
        alert('Aceptar');
    }
</script>
```

Evitar acciones por defecto

Cuando un evento se produce, puede pasar que ya tenga acciones por defecto asignadas. Por ejemplo, cuando se pincha sobre un enlace, ese click desencadena la acción que lleva a abrir la página que tenga asociada. Si se quiere evitar esa acción, se puede incluyendo un return false; al final de la acción que indique el programador:

```
<a href="https://www.w3schools.com/js/" onclick="alert('Aceptar');
return false;">
        Acceso a la página de JS </a>
```

Si lo que se quiere es controlar la acción por defecto dependiendo de la intención del usuario, se puede, por ejemplo, preguntar antes de ejecutar:

```
<a href="https://www.w3schools.com/js/" onclick="return
confirmar();"> Acceso a la página de JS </a>
    <script>
        function confirmar(){
            return confirm("Desea abandonar la página");
        }
    </script>
```

1.2. Registro de eventos según el modelo tradicional

Como opción más recomendable que el registro en línea, aunque no la óptima, ya que separa el código HTML del JavaScript, se puede asociar a un elemento HTML una propiedad con el nombre del evento (precedido de on) a escuchar mediante DOM:

```
document.getElementById(' botonAceptar ').onclick = function () {
        alert('Aceptar');
    }
//O, con una función no anónima. En este caso la función se indica //
sin ()
document.getElementById(' botonAceptar ').onclick = aceptar;
function aceptar() {
    alert('Aceptar');
}
```

Si en algún momento se quiere anular esa captura de evento:

```
document.getElementById(' botonAceptar ').onclick = null;
```

Un problema que puede surgir al utilizar este modelo de registro es que al asociar la captura de un evento con un elemento html puede darse el caso de que se asocie a un elemento que no se haya creado. Para evitar esto es conveniente poner el código que atiende a los eventos dentro de una función que se ejecute cuando se haya terminado de cargar la página y se haya creado el árbol DOM. Esto aplica para cualquier acción o código que vaya a modificar el árbol DOM. Siguiendo el ejemplo anterior:

```
window.onload = function() {
    document.getElementById('botonAceptar').onclick = aceptar;
}

function aceptar() {
    alert('Aceptar');
}
```

1.3. Registro de eventos según el modelo avanzado del W3C

Se trata de la forma de registrar eventos óptima y la que se debería de usar. Este modelo permite añadir a un elemento un listener de eventos mediante el método **addEventListener**, el cual recibe como parámetros:

- primer parámetro: el nombre del evento a escuchar (esta vez sin *on* delante).
- segundo parámetro: la función a ejecutar cuando se produzca el evento.
- tercer parámetro: un booleano para indicar lo llamado fase de burbujeo (se verá en un punto más adelante en el tema). Este último parámetro, de momento, se dejará siempre a false.

Cuando se utilizan funciones con nombre, éste se pasa como parámetro sin paréntesis:

```
document.getElementById('botonAceptar').addEventListener('click',
aceptar,false);
function aceptar() {
    alert('Aceptar');
}
// O como función flecha
document.getElementById('botonAceptar').addEventListener('click', ()
=> alert('Aceptar'), false);
```

Utilizar la opción de función anónima será obligatorio siempre que se quiera pasar algún parámetro a la función listener, aunque este caso no es muy habitual.

Capturando eventos a través de listeners permite poner varios para un mismo evento, ya que se ejecutarán todos ellos.

```
document.getElementById('botonAceptar').addEventListener('click',
aceptar,false);
document.getElementById('botonAceptar').addEventListener('click',
comprobarDatos,false);
//Siendo aceptar y comprobarDatos dos funciones diferentes
```

Si se quiere eliminar uno de esos listeners se hará mediante el método **removeEventListener**, pero solo en caso de que se haya utilizado una función con nombre cuando se haya declarado, ya que hay que pasarle dicha función como segundo parámetro.

```
document.getElementById('botonAceptar').removeEventListener('click',
aceptar);
```

EJERCICIO 1

- A partir del resultado del ejercicio 7 de la unidad DOM/BOM, modifica todas las capturas de eventos realizadas para que el registro de los eventos se haga siguiendo el modelo avanzado del W3C.
- Añade, a la funcionalidad del ejercicio, los cambios necesarios para que cuando se haga doble click sobre el primer párrafo de la página, además de lo que ya hace, cambie el color del encabezado h1 a rojo si es negro y viceversa.
- La primera vez que se produzca cada uno de los eventos, se eliminará el listener asociado al elemento, salvo para los casos de invisibilizar párrafos, eliminar o asignar el atributo name de los radio buttons, eliminar la última fila de la lista nueva y cambiar estilos.
- El evento para eliminar la última fila de la nueva lista creada se asignará cuando se haya creado dicha lista.

1.4. Los objetos this y event

Al producirse un evento se generan automáticamente en su función manejadora 2 objetos: this e event.

this

Hace referencia al elemento que contiene el código en donde se encuentra la variable this. En el caso de una función listener será el elemento que tiene el escuchador que ha recibido el evento.

event

Es un objeto y la función listener de cualquier evento lo recibe como parámetro.

```
document.getElementById('botonAceptar').addEventListener('click',
aceptar,false);
document.getElementById('botonAceptar').addEventListener('dblclick',a
ceptar,false);

function aceptar(event) {
    if (event.type == "click") {
        this.style.color = "red";
    }
    if (event.type == "dblclick") {
        this.style.color = "blue";
    }
}
```

Tiene propiedades y métodos que dan información sobre el evento. Algunos de ellos son:

- **.type**: indica qué evento se ha producido.
- **.target**: referencia el elemento donde se produjo (será this o un descendiente).
- **.currentTarget**: el elemento que contiene el escuchador del evento lanzado (normalmente el mismo que this). Por ejemplo si hay un <p> al que se le asigna un listener de 'click', y dentro tiene un elemento , si se hace click sobre el event.target será el que es donde se ha hecho click (está dentro de <p>) pero.currentTarget será <p> (que es quien tiene el listener que se está ejecutando).
- **.cancelable**: indica si el evento puede cancelarse. En caso afirmativo se puede llamar a event.preventDefault() para cancelarlo.
- **.preventDefault()**: si un evento tiene un listener asociado se ejecuta el código de dicho listener y después el navegador realiza la acción que correspondería por defecto al evento. Con preventDefault() se cancela la acción por defecto del navegador para el evento. Por ejemplo si el evento era el submit de un formulario éste no se enviará o si era un click sobre un hiperenlace no se irá a la página indicada en él.
- **.stopPropagation**: un evento se produce sobre un elemento y todos su padres. Por ejemplo, si hacemos click en un que está en un <p> que está en un que está en el BODY el evento se va propagando por todos estos elementos y saltarían los escuchadores asociados a todos ellos en caso de que los hubiera. Si alguno llama a este método el evento no se propagará a los demás elementos padre.

Otras propiedades que se pueden encontrar dependiendo del tipo de evento son:

- **.button**: qué botón del ratón se ha pulsado (0: izq, 1: rueda; 2: dcho).
- **.screenX** / **.screenY**: las coordenadas del ratón respecto a la pantalla.
- **.clientX** / **.clientY**: las coordenadas del ratón respecto a la ventana cuando se

produjo el evento.

- **.pageX** / **.pageY**: las coordenadas del ratón respecto al documento (si se ha hecho un scroll será el clientX/Y más el scroll).
- **.offsetX** / **.offsetY**: las coordenadas del ratón respecto al elemento sobre el que se produce el evento.
- **.detail**: si se ha hecho click, doble click o triple click.

Los eventos de teclado son los más incompatibles entre diferentes navegadores. En el teclado hay teclas normales y especiales (Alt, Ctrl, Shift, Enter, Tab, flechas, Supr, …). En la información del teclado hay que distinguir entre el código del carácter pulsado (e=101, E=69, €=8364) y el código de la tecla pulsada (para los 3 caracteres es el 69 ya que se pulsa la misma tecla). Las principales propiedades de event son:

- **.key**: nombre de la tecla pulsada.
- **.which**: código de la tecla pulsada.
- **.keyCode** / **.charCode**: código de la tecla pulsada y del carácter pulsado (según navegadores)
- **.shiftKey** / **.ctrlKey** / **.altKey** / **.metaKey**: si está o no pulsada la tecla SHIFT / CTRL / ALT / META. Esta propiedad también la tienen los eventos de ratón.

EJERCICIO 2

Siguiendo con el proyecto del ejercicio 1:

- Añade un nuevo elemento (p, span, div, el que quieras) que sea paralelo al título principal de la página.
- Registra un evento para la etiqueta <body> de tal manera que al mover el ratón en cualquier punto de la ventana del navegador, se muestre, en ese elemento añadido, la posición del puntero respecto de la pantalla y respecto de la página.
- Registra otro evento para <body> para que al pulsar cualquier tecla se muestre en un alert el key y el keyCode de la tecla pulsada.

1.5. Bindeo del objeto this

Como ya se ha visto en unidades anteriores, cuando se entra en un método el elemento this pasa a ser el del contexto de esa función y no el que hubiera antes de entrar. En cuanto a las funciones manejadoras de eventos, hay veces que se necesita que this no sea el elemento sobre quien se produce el evento sino que se quiere conservar el valor que tenía antes de entrar. Por ejemplo si la función listener es un método de una clase, al entrar en la función se pierde la referencia al objeto(que sería el this que habría antes de entrar).

El método .bind() permite pasarle a una función el valor que quiera dársele a this dentro de dicha función. Por defecto a una función listener de eventos se le bindea el valor de event.currentTarget. Si se quiere que tenga otro valor hay que indicarlo con **.bind()**.En el siguiente ejemplo el valor de this dentro de la función aceptar será variable:

```
document.getElementById('botonAceptar').addEventListener('click',
aceptar.bind(variable));
```

Si fuese el caso de un listener dentro de una clase, para mantener el valor de this y que haga referencia al objeto sobre el que se está actuando habría que hacer:

```
document.getElementById('botonAceptar').addEventListener('click',
aceptar.bind(this));
```

1.6. Propagación de eventos (bubbling)

Teniendo en cuenta que los elementos de una página web es normal que se aniden unos en otros(Hola), si se crea un listener del evento click para cada uno de ellos se ejecutarán todos ellos, pero ¿en qué orden?.

El W3C estableció un modelo en el que primero se disparan los eventos de fuera hacia dentro (primero el), llamado fase de captura, y al llegar al más interno (el) se vuelven a disparar de nuevo pero de dentro hacia afuera, lo que se conoce como fase de burbujeo. Aquí es donde entra en juego el tercer parámetro del método addEventListener:

- **false** (valor por defecto): los eventos se disparan en fase de burbujeo. Por lo que por defecto se ejecutará primero el listener del , después del y después del .
- **true**: indica que se ha de disparar el evento en fase de captura, y los listener se ejecutarán en orden inverso al anterior.

En cualquier momento es posible evitar que se siga propagando un evento, ejecutando el método .stopPropagation() en el código de cualquiera de los listeners. Es decir, si tras la ejecución del listener del , no queremos que se ejecuten los de y , se puede indicar mediante este método.

1.7. innerHTML y listeners de eventos

Cuando se modifica la propiedad innerHTML de un elemento del árbol DOM, ya se sabe que todos sus elementos desaparecen y, con ellos, desaparecerán todos los listeners asociados, ya que de incluirse elementos nuevos los listeners hay que asociarlos después de crearlos.

En el siguiente ejemplo se añade una nueva fila a una tabla y a esa nueva fila se le asocia un listener por el cual al hacer click sobre la misma se mostrará su id:

```
function crearNuevaFila(tabla) {
    let numFilas = tabla.childElementCount;
    let fila = "<tr id='" + numFilas + "'><td>" + (numFilas + 1) +
"</td><td>" + (numFilas + 2) + "</td></tr>";
    tabla.innerHTML += fila;
    document.getElementById(numFilas).addEventListener('click',
        event => {
            alert('Id: ' + event.currentTarget.id);
            event.stopPropagation();
        },false);
}
```

EJERCICIO 3

- En el Aula Virtual encontrarás el proyecto “innerHTML y Eventos”. En él está el código del ejemplo anterior. Intenta comprender que hace en cada línea.
- Modifica el código del ejemplo de tal forma que no se actualice todo el innerHTML de la tabla si no que se añada la última fila creada únicamente. De esa forma:
  - El código será óptimo, porque el navegador no tendrá que renderizar toda la tabla, solo la nueva fila.
  - No se perderán los listeners de filas ya existentes.

2. Formularios

Una de las acciones que permite llevar a cabo JavaScript, es la validación de formularios en el lado del cliente. Esto proporciona una primera validación que no debería sustituir a la validación del lado del servidor, ya que en el lado del cliente es posible manipular el código desde la consola y saltarse así las validaciones establecidas.

Antes de ver como validar los datos introducidos en un formulario, se va a hacer un recorrido por elementos del mismo, operaciones básicas con ellos y expresiones regulares.

2.1. Formularios y sus elementos

Cuando un navegador carga una página, automáticamente crea un array llamado forms con una referencia a todos los formularios de la misma, y un array llamado elements, para cada uno de esos formularios, que contendrá la referencia a todos sus elementos.

```
document.forms[0]; //Primer formulario de la página
document.forms[0].elements[0]; //Primer elemento del primer
//formulario
```

Esta forma de acceder a los formularios no es muy recomendable ya que, dado el carácter dinámico de las webs, puede que el primer formulario de la página no sea siempre el mismo. Por lo tanto, la mejor manera de acceder a un formulario, o a uno de sus elementos, es a través de su atributo name o su atributo id, o utilizando las funcionesDOM de acceso a nodos:

```
<form name="formularioAlta" id="idFormularioAlta">
   <input type="text" name="nombreUsuario" id="nomUser"/>
</form>
//.. y en el documento de scripts …
let formularioAlta = document.formularioAlta;
let inputNombre = document.nombreUsuario;
let mismoFormAlta = document.getElementById("idFormularioAlta");
```

Propiedades de elementos de un formulario

Cada elemento de un formulario dispone de las siguientes propiedades:

- type: indica de que tipo es el elemento.
  - En un elemento <input>, su valor coincidirá con el del atributo type.
  - Para las listas desplegables su valor es select-one en caso de que se permita seleccionar solo una opción, o select-multiple en caso contrario.
  - En los elementos de tipo <textarea>, el valor de type es textarea.
- form: referencia al formulario al que pertenece el elemento:
  ```
  let formularioAlta = document.getElementById("nomUser").form;
  ```
- value: permite leer y modificar el valor del atributo value.
  - Para los campos de texto (<input type="text"> y <textarea>) recupera el valor introducido por el usuario.
  - Para los botones obtiene el texto que se muestra en el botón.

Eventos de formulario más comunes

- **click**: evento que se produce cuando se pincha con el ratón sobre un elemento.
- **change**: evento que se produce cuando:
  - el usuario cambia el valor de un elemento de texto (<input type="text"> o <textarea>).
  - el usuario selecciona una opción en una lista desplegable (<select>) y, seguidamente, pasa al siguiente campo del formulario.

- **focus**: el usuario selecciona un elemento del formulario.
- **blur**: el usuario ha deseleccionado un elemento por haber seleccionado otro elemento del formulario. El elemento pierde el foco.
- **submit**: cuando se envía un formulario.

2.2. Operaciones básicas con formularios

Obtener el valor de los campos

Mediante la propiedad value, se puede recuperar y establecer el valor de los **elementos de tipo texto** (textarea e input de tipo text).

En caso de un grupo de **radiobuttons**, la propiedad checked devuelve true para el elemento seleccionado y false en cualquier otro caso:

```
<input type="radio" value="si" name="preg" id="pregunta_si"/> SI
<input type="radio" value="no" name="preg" id="pregunta_no"/> NO

//El siguiente código determina si cada radiobutton se ha
seleccionado o no:
var elementos = document.getElementsByName("preg");
for(var i=0; i<elementos.length; i++) {
      alert(" Elemento: " + elementos[i].value + "\n Sel.: " +
      elementos[i].checked);
}
```

Para los elementos **checkbox**, se comprueba si estos están seleccionados de forma individual, ya que, en este caso, los checkbox de un grupo no son mutuamente excluyentes:

```
<input type="checkbox" value="cond" name="cond" id="cond"/> Condiciones
<input type="checkbox" value="priv" name="priv" id="priv"/> Política de
privacidad

var elem = document.getElementById("cond");
alert(" Elemento: " + elem.value + "\n Seleccionado: "+elem.checked);
elem = document.getElementById("priv");
alert(" Elemento: " + elem.value + "\n Seleccionado: "+elem.checked);
```

Para obtener el valor de una **lista desplegable**, hay que recuperar el valor del atributo value de la opción seleccionada. Para ello se hace uso del array options que es creado automáticamente por el navegador para cada lista. También está disponible la propiedad selectedIndex, la cual actualiza automáticamente el navegador cada vez que se cambia la selección, que almacena el índice, del array options, donde se encuentra la opción seleccionada.

```
        <select id="opciones" name="opciones">
            <option value="Sal">Salamanca</option>
            <option value="Bil">Bilbao</option>
            <option value="Sant">Santander</option>
            <option value="Alb">Albacete</option>
        </select>

//Y en el script...
```

```
var lista = document.getElementById("opciones");
// Obtener el valor y el texto de la opción seleccionada
var indiceOpcionSel = lista.selectedIndex;
var opcionSeleccionada = lista.options[indiceOpcionSel];
var textoSel = opcionSeleccionada.text;
var valorSel = opcionSeleccionada.value;
//Las 4 sentencias anteriores se podrían sustituir por:
var textoSel = lista.options[lista.selectedIndex].text;
var valorSel = lista.options[lista.selectedIndex].value;
alert("Ciudad seleccionada: " + textoSel + "\n Valor de la opción: "
+ valorSel );
```

Establecer el foco en un elemento

Que un elemento de un formulario tenga el foco quiere decir que está seleccionado y se puede escribir en el o modificar alguna propiedad. El foco se puede pasar de un elemento a otro mediante la tecla de tabulación, y no solo entre elementos de un formulario sino que afecta a cualquier elemento de la página como enlaces, imágenes, etc. Por ejemplo, si el foco lo tiene una lista desplegable, se podrá mover entre opciones con las flechas del teclado.

El foco de una página web se suele dar al elemento que se considere más importante. Por ejemplo, si pones www.google.es en tu navegador, cuando se abra el buscador el foco estará directamente en la caja de búsqueda.

Como ya se ha visto anteriormente, para dar el foco se hace a través del método focus():

```
//Para dar el foco a la lista desplegable de ciudades:
document.getElementById("opciones").focus();
```

Para asignar el foco a un elemento es recomendable comprobar, primero, que existe y después que no esté oculto (cuando el atributo hidden está a true).

Limitar el tamaño de caracteres de un textarea

Limitar el tamaño máximo de un elemento de tipo textarea, no es posible a través de algún atributo html como en los cuadros de texto. Pero, con JavaScript se puede añadir este dato mediante la captura de eventos. Por ejemplo, si al capturar el evento keypress se devuelve el valor false, la tecla pulsada no se tiene en cuenta. Por lo tanto, podría hacerse:

```
<textarea id="descripcion" onkeypress="return maximoLetras(100);">
</textarea>

//Y en el script ...
function maximoLetras(maxCaracteres) {
      var elemento = document.getElementById("descripcion");
      if(elemento.value.length >= maxCaracteres ) {
          return false;
      }
      else {
          return true;
      }
}
```

Restringir los caracteres permitidos en un cuadro de texto

Es bastante común que, por los requisitos establecidos para el desarrollo de una página, sea necesario bloquear el uso de ciertos caracteres en los cuadros de texto. Otra vez, haciendo uso de la captura de eventos, mediante JavaScript es posible indicar los caracteres prohibidos:

```
<input type="text" id="tlfn" onkeypress="return validarNum (evnt)"
/>
//Y en el script ...
function validarNumeros(event) {
    var permitidos = "0123456789";
    // Obtener la tecla pulsada
    var codigoCaracter = event.charCode || event.keyCode;
    var caracter = String.fromCharCode(codigoCaracter);
    // Si no se trata de un valor permitido no se escribirá en la
caja de texto
    return permitidos.indexOf(caracter) != -1;
}
```

EJERCICIO 4

- Crea una página web con un formulario. Dicho formulario contendrá el campo input del último ejemplo (sin capturar evento alguno). Además, se completará el formulario:
  - Incorporando tres checkbox para seleccionar franja horaria (6:00 a 14:00, de 14:00 a 20:00 y de 20:00 a 01:00), dos radiobutton para indicar si es de prepago o no, una lista desplegable con 5 nombres de compañías de teléfono para seleccionar una y un área de texto para introducir observaciones. El área de texto admitirá, como máximo, 140 caracteres.
  - Añadiendo, a la etiqueta form, los atributos action(sin url) y method.
  - Incluyendo, por último, un botón para enviar el formulario. Escoge el tipo de declaración adecuado.
- Captura el evento de envío de formulario y comprueba:
  - Que todos los campos tienen valor o están seleccionados, menos los checkbox que puede no haberse seleccionado ninguno.
  - En caso de que no lo estén, se avisará mediante un alert de los que faltan y no se seguirá con el envío del formulario.
  - En caso de que estén completo se mostrarán en un alert el valor de cada uno de ellos (para los checkbox y radiobutton solo de los seleccionados).
- Por último, para el campo teléfono añade, ahora sí, la captura del evento onkeypress. Cuando se capture ese evento se comportará como en el último ejemplo salvo que:
  - El evento se registrará siguiendo el modelo avanzado del W3C.
  - Y se permitirá el uso de las flechas especiales Backspace y Supr para borrar caracteres y Flecha Izquierda y Flecha Derecha para moverse por el cuadro de texto.

2.3. Objeto RegExp

El objeto RegExp de JavaScript permite, mediante su instanciación, crear expresiones regulares. Éstas permiten buscar un patrón dado en una cadena de texto. Es posible también, y más recomendable, declarar una expresión regular de forma literal incluyendo el patrón entre símbolos / :

```
let texto = "Desarrollo web en entorno cliente";
let expr = new RegExp("en");
let expr2 = /tr/;
//Aunque los métodos de comprobación se verán más adelante, las
//siguientes comprobaciones devolverán:
// true la primera porque existe la expresión dentro de la cadena
//false la segunda.
expr.test(texto);
expr2.test(texto);
```

Patrones

Lo más común es crear expresiones regulares haciendo uso de patrones, y no mediante una cadena literal. Los más comunes (y que se usan en combinaciones entre ellos) son:

- [..] : Se incluye entre corchetes una serie de caracteres o un rango:
  - [abc]: se busca que la cadena contenga alguno de esos caracteres.
  - [^abc]: se busca que contenga cualquiera excepto los indicados.
  - [a-z]: se busca cualquier letra minúscula ('-' indica rango).
  - [a-zA-Z]: se busca cualquier letra ya sea minúscula o mayúscula.
- | : El carácter pipe indica OR:
  - (x|y): se busca que la cadena contenga x o y.
  - (http|https): se busca que la cadena contenga http o https.
- Meta-caracteres:
  - . : un punto indica un solo carácter(el que sea).
  - \d: buscará un dígito. Mientras que \D indica lo que no sea un dígito.
  - \s: espacio en blanco. Con \S se busca que no lo haya.
  - \w: buscará una palabra o carácter alfanumérico. \W lo contrario.
  - \n: nueva línea.
- Cuantificadores:
  - +: al menos 1 vez (ej. [0-9]+ al menos un dígito).
  - *: 0 o más veces.
  - ?: 0 o 1 vez.
  - {n}: n caracteres (ej. [0-9]{5} = 5 dígitos).
  - {n,}: n o más caracteres.
  - {n,m}: entre n y m caracteres.
  - ^: al ppio de la cadena (ej.: ^[a-zA-Z] = empieza por letra).
  - $: al final de la cadena (ej.: [0-9]$ = que acabe en dígito).
- Modificadores:
  - /i: que no distinga entre Maysc y minsc (Ej. /html/i = buscará html, Html, HTML, …)
  - /g: búsqueda global, busca todas las coincidencias y no sólo la primera.
  - /m: busca en más de 1 línea (para cadenas con saltos de línea).

Métodos

Para trabajar con expresiones regulares, JavaScript provee los siguientes métodos:

- expr.**test**(cadena): devuelve true si la cadena coincide con la expresión. Si se incluye el modificador /g en la expresión, cada vez que se llame a este método buscará desde la posición de la última coincidencia:

```
let cadena = "Desarrollo de aplicaciones web";
let reg = /es/g;
console.log(reg.test(cadena)); // Devuelve true
console.log(reg.test(cadena)); // Devuelve true
console.log(reg.test(cadena)); // Devuelve false
```

- expr.**exec**(cadena): Método que, buscando igual que el anterior, lo que devuelve es un objeto con la coincidencia encontrada, su posición y la cadena completa:

```
let cadena = "Desarrollo de aplicaciones web";
let reg = /es/g;
console.log(reg.test(cadena)); // test Devuelve true
console.log(reg.test(cadena)); // test Devuelve true
console.log(reg.test(cadena)); // test Devuelve false
console.log(reg.exec(cadena)); // exec Devuelve ["es",
//index: 1, input: "Desarrollo de aplicaciones web"]
console.log(reg.exec(cadena)); // exec Devuelve ["es",
//index: 24, input: "Desarrollo de aplicaciones web"]
console.log(reg.exec(cadena)); // exec Devuelve null
```

- cadena.**match**(expr): método que se aplica a la cadena pasándole como parámetro la expresión regular(al revés que en el método anterior). Si la expresión tiene el modificador /g, devolverá un array con todas las coincidencias:

```
console.log(cadena.match(reg));
// Devuelve ["es", "es"]
```

- cadena.**search**(expr): devuelve la posición donde se encuentra la coincidencia buscada o -1 si no se da.
- cadena.**replace**(expr, cadena2): devuelve una nueva cadena copia de la primera. En esta segunda cadena se han reemplazado las coincidencias con la expresión regular por la cadena pasada como 2º parámetro:

```
//La siguiente sentencia devuelve "Dddarrollo de
aplicaciondd web"
str.replace( /es/g, "dd");
// Y esta otra "DESarrollo de aplicacionES web"
str.replace(/es/g, coincidencia =>
coincidencia.toUpperCase() );
```

EJERCICIO 5

- Haz el ejercicio 4 del documento “Ejercicios de Eventos y Formularios” que encontrarás en el Aula Virtual de la asignatura.

2.4. Validación de formularios

La validación de formularios puede hacerse mediante las opciones de validación que ofrece HTML5 y que sea el navegador quien la haga o, proporcionando mayor control del proceso al desarrollador, mediante JavaScript.

Validación incorporada en HTML5

HTML5 proporciona algunos atributos para los campos de un formulario que exigirán ciertos contenidos o comportamientos. Los más usados son:

- **required**: indica que el campo es obligatorio, por lo que si no se introduce ningún valor la validación fallará. Si se trata de un grupo de radio buttons, con ponerlo en uno solo de ellos ya obligará a que se seleccione una opción.
- **pattern**: si se incluye este atributo, el navegador validará que el valor introducido cumpla con la expresión regular indicada.
- **minlength** / **maxlength**: atributos que indican la longitud mínima o máxima del valor que se introduzca.
- **min** / **max**: atributos que permiten indicar un valor mínimo o máximo para un campo numérico.

El navegador también validará que el contenido de un campo coincida con el tipo del mismo (email, number, etc.).

Cuando un elemento tiene contenido válido, el navegador asigna a ese campo la pseudoclase **:valid**, y en caso contrario la pseudoclase **:invalid**. Esto facilita que se puedan indicar propiedades concretas a campos en caso de ser correctos o incorrectos. Por ejemplo, para los campos input:

```
input:invalid {
  border: 2px dotted red;
}
```

Cuando se utiliza este tipo de validación, ésta se realiza antes de enviar el formulario y, en el momento que encuentra un error, lo muestra y no sigue con la validación ni con el envío del formulario.

Validación mediante Javascript

Validar mediante HTML5 es una forma correcta de hacerlo, pero puede quedarse algo corta, por lo que es muy común combinar esas opciones de validación con JavaScript.

Es posible validar con JavaScript directamente, pero es mucho menos tedioso hacerlo con la API de validación de restricciones(Constraint Validation API) que ofrece HTML5. Haciendo uso de esta API:

- los requisitos de validación de cada campo están como atributos HTML de dicho campo por lo que son fáciles de ver.
- Es el navegador quien comprueba si el contenido del campo cumple o no esos requisitos y el desarrollador, mediante la API, sólo pregunta si se cumplen o no.
- Automáticamente se asignan a los campos las clases :valid o :invalid por lo que no hay que añadirles clases para destacarlos.

Las propiedades y métodos más importantes que proporciona esta API son:

- **validationMessage**: contiene el texto del error de validación proporcionado por el navegador, o una cadena vacía en caso de no haberlo.
- **validity**: objeto que incluye propiedades booleanas para comprobar los distintos fallos de validación.
  - valid: indica si el campo es válido.
  - valueMissing: indica si no se cumple la restricción required.
  - typeMismatch: indica si el contenido del campo no cumple con su atributo type.
  - patternMismatch: indica si no se cumple con el pattern indicado.
  - tooShort / tooLong: indica si no se cumple el atributo minlength/maxlength.
  - rangeUnderflow / rangeOverflow: indica si no se cumple el atributo min / max.
- **checkValidity()**: devolverá true o false, indicando si el campo al que se aplica es o no válido. Si no lo es lanza, además, un evento invalid sobre el campo.
- **setCustomValidity(mensaje)**: Permite sustituir el mensaje del navegador por uno personalizado, para un elemento de tipo input.

Para validar un formulario haciendo uso de esta API es necesario indicarle al navegador que no va a encargarse de mostrar los mensajes de error ni de decidir si se envía o no el formulario (solo valida los campos), ya que se hará mediante JavaScript. Esto se hace añadiendo el atributo **novalidate** a la etiqueta form.

EJERCICIO 6

- Descárgate el proyecto "Ejemplo Validation API" del Aula Virtual y revísalo. Contiene el ejemplo visto en clase.
  - Comprueba si has entendido el ejemplo cuando se ha explicado en clase.
  - Incluye trazas de log en el código JavaScript para ver el contenido de cada variable o propiedad que se utiliza dentro de la función listener.
- Partiendo del formulario del documento "ejercicio6.html" incluye los cambios necesarios para que, utilizando la API vista, se hagan las siguientes validaciones:
  - El nombre y el teléfono son campos obligatorios. El teléfono tendrá el formato xxx xxx xxx, siendo x cualquier número entre el 0 y el 9 y estando cada tres caracteres separados por un espacio en blanco.
  - El correo electrónico no es obligatorio, pero de incluirse deberá tener un formato válido.
  - Sera obligatorio también seleccionar un tipo de vehículo mientras que los extras podrán no seleccionarse.
  - La fecha de reserva será obligatoria también y con el formato dd/mm/aaaa.
- Para todos los errores que aparezcan en la validación del punto anterior, se incluirá el texto informativo en un div, el cual se incluirá por encima del botón "Reservar".