JAVASCRIPT

FORMULARIOS

INTRODUCCIÓN

- Los formularios y controles, como <input>, tienen muchos eventos y propiedades especiales.
- Los **formularios** del documento son miembros de la **colección** especial **document.forms**.
- Esa es la llamada “Colección nombrada”: es nombrada y ordenada. Podemos usar el nombre o el número en el documento para conseguir el formulario.

```
document.forms.my; // el formulario con name="my"
document.forms[0]; // el primer formulario en el documento
```

- Cuando tenemos un **formulario**, **cualquier elemento** se encuentra **disponible en** la colección nombrada **form.elements**.

NAVEGACIÓN: FORMULARIOS Y ELEMENTOS

- Por ejemplo:

```
<form name="my">
    <input name="one" value="1">
    <input name="two" value="2">
</form>
<script>
    // obtención del formulario
    let form = document.forms.my; // elemento <form name="my">
    // get the element
    let elem = form.elements.one; // elemento <input name="one">
    alert(elem.value); // 1
</script>
```

NAVEGACIÓN: FORMULARIOS Y ELEMENTOS

- Puede haber múltiples elementos con el mismo *name*. Esto es típico en el caso de los botones de radio y checkboxes.
- En ese caso form.elements[name] es una colección. Por ejemplo:

```
<form>
    <input type="radio" name="age" value="10">
    <input type="radio" name="age" value="20">
</form>
<script>
    let form = document.forms[0];
    let ageElems = form.elements.age;
    alert(ageElems[0]); // [object HTMLInputElement]
</script>
```

NAVEGACIÓN: FORMULARIOS Y ELEMENTOS

- Todos los controles están disponibles en form.elements.
- **Fieldsets como “sub-formularios”**
- Un formulario puede tener uno o varios elementos <fieldset> dentro. Estos **también tienen la propiedad elements que lista los controles del formulario dentro de ellos.**

NAVEGACIÓN: FORMULARIOS Y ELEMENTOS

- Por ejemplo:

```
<form id="formulario">
    <fieldset name="userFields">
        <legend>info</legend>
        <input name="login" type="text">
    </fieldset>
</form>

<script>
    let form = document.forms[0];
    alert(form.elements.login); // <input name="login">
    let fieldset = form.elements.userFields;
    alert(fieldset); // HTMLFieldSetElement
    /* podemos obtener el input por su nombre tanto desde el formulario como desde el
fieldset*/
    alert(fieldset.elements.login == form.elements.login); // true
</script>
```

NAVEGACIÓN: FORMULARIOS Y ELEMENTOS

- Hay una notación corta: podemos acceder al elemento como **form[index/name]**. En lugar de form.elements.login podemos escribir form.login.
- Esto también funciona, pero **tiene un error menor: si accedemos a un elemento, y cambiamos su name, se mantendrá disponible mediante el nombre anterior (así como mediante el nuevo).**
- Ejemplo:

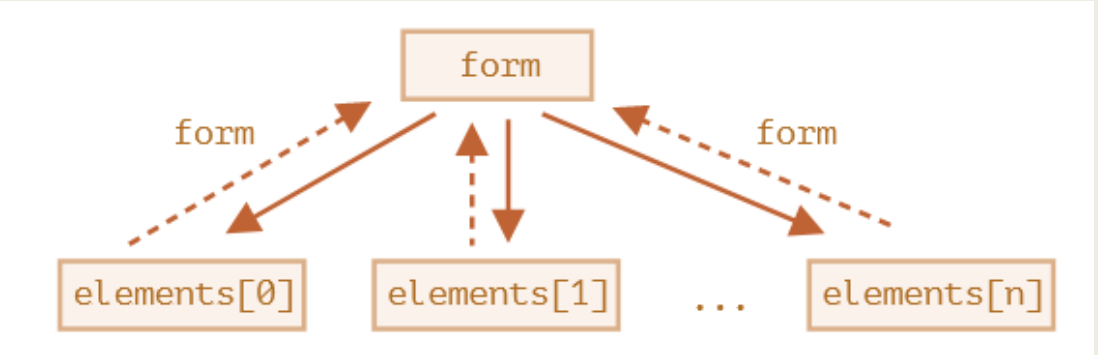
```
<form id="form">
    <input name="login">
</form>
<script>
    let form = document.forms[0];
    alert(form.elements.login == form.login); // true, el mismo <input>
    form.login.name = "username"; // cambiamos el name al <input>
    // form.elements actualiza el nombre:
    alert(form.elements.login); // undefined
    alert(form.elements.username); // input
    // form permite ambos nombres: el nuevo y el viejo
    alert(form.username == form.login); // true
</script>
```

Esto no suele ser un problema, porque raramente se cambian los names de los elementos de un formulario.

NAVEGACIÓN: FORMULARIOS Y ELEMENTOS

- Para cualquier elemento, el formulario está disponible como **element.form**.
- Por ejemplo:

```
<form id="form">
    <input type="text" name="login">
</form>
<script>
    // form -> element
    let form = document.forms[0];
    let login = form.login;
    // element -> form
    alert(login.form); // HTMLFormElement
</script>
```



ELEMENTOS DEL FORMULARIO

- **input y textarea**

- Podemos acceder a sus valores como **input.value (cadena) o input.checked (booleano) para** casillas de verificación (**checkboxes**) **y** botones de opción (**radio buttons**).
- De esta manera:

```
input.value = "New value";

textarea.value = "New text";

input.checked = true; // para checkboxes o radios
```

- **Usa textarea.value, no textarea.innerHTML**
  - Incluso aunque <textarea>...</textarea> contenga su valor como HTML anidado, nunca deberíamos usar textarea.innerHTML para acceder a él.
  - Esto solo guarda el HTML que había inicialmente en la página, no su valor actual.

ELEMENTOS DEL FORMULARIO

➢ select y option

- Un elemento <select> tiene 3 propiedades importantes:
  - **select.options:** la colección de subelementos <option>,
  - **select.value:** el valor del <option> seleccionado actualmente, y
  - **select.selectedIndex:** el número del <option> seleccionado actualmente.
- Ellas proveen **tres formas diferentes de asignar un valor para un elemento <select>**:
  1. Encontrar el elemento <option> correspondiente (por ejemplo entre select.options) y asignar a su option.selected un true.
  2. Si conocemos un nuevo valor: asignar tal valor a select.value.
  3. Si conocemos el nuevo número de opción: asignar tal número a select.selectedIndex.

ELEMENTOS DEL FORMULARIO

- **select y option**
- Ejemplo:

```
<select>
    <option value="apple">Apple</option>
    <option value="pear">Pear</option>
    <option value="banana">Banana</option>
</select>
<script>
    /* las tres líneas hacen lo mismo
    cargar la página con la última opción seleccionada*/
    let select = document.querySelector('select');
    select.options[2].selected = true;
    select.selectedIndex = 2;
    select.value = 'banana';
    /*Recuerda que las opciones comienzan en cero, así que index 2 significa la tercera opción. */
</script>
```

EVENTOS DE ENFOQUE/DESENFOQUE

- **Un elemento se enfoca** cuando el **usuario hace click sobre él o al pulsar Tab** en el teclado. Existen también un **atributo autofocus de HTML** que enfoca un elemento por defecto cuando una página carga, ...
- Enfocarse sobre un elemento generalmente significa: “prepárate para aceptar estos datos”, por lo que es el momento de ejecutar código para inicializar la funcionalidad requerida.
- El momento de **desenfoque** (“**blur**”) ocurre cuando **un usuario clica en otro punto o presiona Tab para ir al siguiente campo** de un formulario.
- Perder el foco o desenfocarse generalmente significa: “los datos ya han sido introducidos”, entonces podemos ejecutar código para comprobarlo, o para guardarlo en el servidor, etc.
- Existen importantes **peculiaridades al trabajar con eventos de enfoque**.

EVENTOS DE ENFOQUE/DESENFOQUE

- El evento **focus** es llamado al **enfocar**, y el **blur** cuando el elemento **pierde el foco**.
- En el ejemplo a continuación el **manejador blur** comprueba si se ha introducido un correo, y en caso contrario muestra un error. El **manejador focus** esconde el mensaje de error (en blur se volverá a comprobar):

```
<form action="programa.php" method="get">
    <label for="correo">Su correo por favor:</label>
    <input type="email" id="correo">
    <div id="error"></div>
</form>

<script>
    let entrada = document.querySelector('input');
    entrada.onblur = function() {
        if (!entrada.value.includes('@')) {
            // not email
            entrada.classList.add('invalid');
            let error = document.getElementById('error');
            error.innerHTML = 'Por favor introduzca un correo válido.'
        }
    };
    entrada.onfocus = function() {
        if (this.classList.contains('invalid')) {
            // quitar estilos "error", porque el usuario quiere reintroducir algo
            this.classList.remove('invalid');
            error.innerHTML = "";
        }
    };
</script>
```

Estilos css:

```
.invalid { border-color: red; }
#error { color: red }
```

MÉTODOS focus/blur

- Los métodos **elem.focus()** y **elem.blur()** ponen/quitan el foco sobre el elemento.

```
    <form action="programa.php" method="get">
        <label for="correro">Su correo por favor:</label>
        <input type="email" id="correo">
        <input type="text" placeholder="hacer que el correo sea inválido y tratar de enfocar aquí">
    </form>
<script>
    let entrada = document.querySelector('input');
    entrada.onblur = function() {
        if (!this.value.includes('@')) {
            // no es un correo // mostrar error
            this.classList.add("error");
            // ...y volver a enfocar
            entrada.focus();
        } else {
            this.classList.remove("error");
        }
    };
</script>
```

Estilos css:

```
.error { background-color: red; }
```

No funciona en Firefox (bug).

MÉTODOS focus/blur

- **No podemos** “prevenir perder el foco” llamando a **event.preventDefault() en blur**, porque blur funciona después de que el elemento perdió el foco.
- Una pérdida de foco puede ocurrir por diversas razones, como que el usuario haga click en algún otro lado. El propio JavaScript también puede causarlo, por ejemplo:
  - Un alert traslada el foco hacia sí mismo, lo que causa la pérdida de foco sobre el elemento (evento blur). Y cuando el alert es cerrado, el foco vuelve (evento focus).
  - Si un elemento es eliminado del DOM, también causa pérdida de foco. Si es reinsertado el foco no vuelve.
- Estas situaciones a veces causan que los manejadores focus/blur no funcionen adecuadamente y se activen cuando no son necesarios.
- Es recomendable **tener cuidado al utilizar estos eventos**. Si queremos monitorear pérdidas de foco iniciadas por el usuario deberíamos evitar causarlas nosotros mismos.

ENFOCAR SOBRE CUALQUIER ELEMENTO

- Por defecto, muchos elementos no permiten enfoque.
- La lista varía un poco entre navegadores, pero, por lo general **focus/blur está garantizado para elementos con los que el visitante puede interactuar: <button>, <input>, <select>, <a>**, etc.
- En cambio, elementos que existen para formatear algo, tales como , , <table>, por defecto no son posibles de enfocar. El método elem.focus() no funciona en ellos, y los eventos focus/blur no son desencadenados.
- Esto puede ser modificado usando el atributo **HTML tabindex**.
- **Cualquier elemento se vuelve enfocable si contiene tabindex**. El valor del atributo es el número de orden del elemento cuando Tab (o algo similar) es utilizado para cambiar entre ellos.

EVENTO change

- **Se activa cuando el elemento finaliza un cambio.**
- Para ingreso de texto significa que el evento ocurre cuando se pierde foco en el elemento.
- Por ejemplo, mientras estamos escribiendo en el siguiente cuadro de texto, no hay evento. Pero cuando movemos el focus a otro lado, por ejemplo hacemos click en un botón, entonces ocurre el evento change:

```
<input type="text">
<input type="button" value="Button">
<script>
    let entradas = document.getElementsByTagName('input');
    entradas[0].addEventListener('change', contenido);
    function contenido(){
        alert (this.value);
    }
</script>
```

EVENTO change

- Para otros elementos: select, input type=checkbox/radio se dispara inmediatamente después de cambiar la opción seleccionada:

```
<select>
    <option value="">Select something</option>
    <option>Option 1</option>
    <option>Option 2</option>
    <option>Option 3</option>
</select>
<script>
    let listaSelect = document.querySelector('select');
    listaSelect.addEventListener('change', mostrarSeleccion);
    function mostrarSeleccion(){
        alert(this.value);
    }
</script>
```

EVENTO input

- El evento input se dispara cada vez que un valor es modificado por el usuario.
- A diferencia de los eventos de teclado, ocurre con el cambio a cualquier valor, incluso aquellos que no involucran acciones de teclado: copiar/pegar con el mouse.
- Por ejemplo:

```
<input type="text"> oninput: <span id="result"></span>

<script>

    let entrada = document.querySelector('input');

    entrada.oninput = function() {

        document.getElementById('result').innerHTML = entrada.value;

    };

</script>
```

- Si queremos manejar cualquier modificación en un <input> entonces este evento es la mejor opción.
- Por otro lado, **el evento input no se activa con entradas del teclado u otras acciones que no involucren modificar un valor**, por ejemplo presionar las flechas de dirección ⇦ ⇨ mientras se está en el input.

EVENTO Y MÉTODO submit

- El **evento submit** se activa cuando el formulario es enviado. Normalmente se utiliza para validar el formulario antes de ser enviado al servidor o bien para abortar el envío y procesarlo con JavaScript.
- El **método form.submit()** permite iniciar el envío del formulario mediante JavaScript. Podemos utilizarlo para crear y enviar nuestros propios formularios al servidor.

EVENTO submit

- Un formulario puede enviarse haciendo click en <input type="submit">
- Esta acción causa que el evento submit sea activado en el formulario. El handler puede comprobar los datos, y si hay errores, mostrarlos e invocar event.preventDefault(), entonces el formulario no será enviado al servidor.
- En el formulario siguiente haz click en <input type="submit">. Esta acción muestra alert y el formulario no es enviado debido a la presencia de return false:

EVENTO submit

```
<form action="programa.php">
        <p>
            <lable for="segundo">Click en "submit": </lable>
            <input id="segundo" type="submit" value="Submit">
        </p>
</form>
<script>
    function enviar(){
        alert('¡submit!');
        event.preventDefault();
    }
    document.forms[0].addEventListener('submit', enviar);
</script>
```

MÉTODO submit

- Para **enviar un formulario al servidor manualmente**, podemos usar **form.submit()**.
- Entonces el evento submit no será generado. Se asume que si el programador llama form.submit(), entonces el script ya realizó todo el procesamiento relacionado.
- A veces es usado para crear y enviar un formulario manualmente:

```
let form = document.createElement('form');

form.action = 'https://google.com/search';

form.method = 'GET';

form.innerHTML = '<input name="q" value="test">';

// el formulario debe estar en el document para poder enviarlo

document.body.append(form);

form.submit();
```