Unidad 6 - Formularios

Formularios y controles de formulario

Acceder a los formularios y a los controles de formulario

Los formularios y controles en HTML pueden ser manipulados como cualquier otro elemento del DOM, pero también tienen propiedades específicas que facilitan su uso.

Formularios

El objeto `document` incluye la propiedad `forms`, que devuelve una `HTMLCollection` con todos los formularios del documento.

Propiedad	Descripción
`document.forms`	`HTMLCollection` con todos los formularios
`document.forms[0]`	Primer formulario del documento
`document.forms.id`	Formulario con el `id` especificado
`document.forms["name"]`	Formulario con el atributo `name` especificado

Acceso desde controles internos:

Propiedad	Descripción
`control.form`	Referencia al formulario al que pertenece el control

Controles

Los formularios tienen una propiedad `elements`, una colección con todos sus controles.

Propiedad	Descripción
`formulario.elements`	`HTMLFormControlsCollection` con los controles
`formulario.elements[0]`	Primer control del formulario
`formulario.elements.id`	Control con el `id` especificado
`formulario.id`	Acceso directo al control usando el `id`

Acceso a controles dentro de un `<fieldset>`:

- Los `<fieldset>` tienen su propia propiedad `elements`.

Acceso a controles con el mismo `name` (ej. radios o checkboxes):

- `form.elements["name"]` devuelve una colección con los controles que comparten ese `name`.

Propiedades de controles de formulario

Tipos comunes de `<input>`:

Propiedad	Descripción
`input.value`	Contenido del campo de texto
`input.checked`	Indica si está seleccionado (`checkbox/radio`)

Ejemplo: `<input type="radio">`

```
let seleccionado = document.querySelector('input[name="status"]:checked');
if (seleccionado) console.log(seleccionado.value);
```

`<select>` y `<option>`:

Propiedad	Descripción
`select.value`	Valor del `<option>` seleccionado
`select.options`	Colección con todos los `<option>`
`select.selectedIndex`	Índice del `<option>` seleccionado
`select.selectedOptions`	Colección con todos los `<option>` seleccionados
`option.selected`	Booleano que indica si el `<option>` está seleccionado
`option.text`	Contenido del `<option>` (como `textContent`)

Cambiar selección:

- `select.value = "nuevoValor"`
- `select.selectedIndex = nuevoIndice`
- `option.selected = true`

Crear fácilmente elementos `<option>` → `new Option(text, value, defaultSelected, selected)`

```
let option = new Option("Text", "value");
// <option value="value">Text</option>
```

Eventos de formularios

Eventos de controles (`input`, `textarea`, `select`)

Evento	Descripción
`change`	Ocurre cuando el valor cambia y pierde el foco (`checkbox`, `radio`).
`input`	Ocurre cada vez que el usuario modifica el valor.

Eventos de foco (`focus`, `blur`)

Evento	Propagación	Descripción
`focus`	No	Elemento obtiene el foco
`blur`	No	Elemento pierde el foco
`focusin`	Sí	Similar a `focus`, pero se propaga

Evento	Propagación	Descripción
`focusout`	Sí	Similar a `blur`, pero se propaga

Evento `submit`

Se activa al enviar un formulario. Puede interceptarse para validar datos antes de enviarlos.

```
form.onsubmit = function(event) {
    event.preventDefault(); // Detiene el envío
    console.log("Formulario validado");
};
```

Métodos clave

Método	Descripción
`form.submit()`	Envía el formulario manualmente
`elemento.focus()`	Coloca el foco sobre el elemento
`elemento.blur()`	Quita el foco del elemento

Expresiones regulares

Las expresiones regulares (`RegExp`) permiten buscar, validar y manipular texto utilizando patrones.

Sintaxis básica

```
let expReg = /patrón/modificadores;
let expReg2 = new RegExp("patrón", "modificadores");
```

Métodos de String con expresiones regulares

Método	Retorno	Descripción
`cadena.match(/expReg/)`	`[String]` o `null`	Devuelve un array con el primer `substring` que coincide o `null` si no hay coincidencias.
`cadena.replace(/expReg/, reemplazo)`	`String`	Sustituye el primer `substring` que coincide con el patrón por el valor de `reemplazo`.

```
"Es hoy".match(/hoy/) // ["hoy"]
"Número: 142719".replace(/1/, "x") // "Número: x42719"
```

Método `.test()`

Método	Retorno	Descripción
`expReg.test(cadena)`	`boolean`	Devuelve `true` si la cadena coincide con el patrón.

```
/[0-9]{8}[a-zA-Z]/.test("012345678A"); // true
```

Construcción de expresiones regulares

Patrones básicos

Patrón	Descripción
`^` y `$`	Principio (`^`) o final (`$`) de cadena.
`.`	Cualquier carácter excepto fin de línea.
`[xyz]`	Coincide con cualquier carácter en el conjunto.
`[a-z]`	Coincide con cualquier carácter dentro del rango.
`[^xyz]`	Coincide con cualquier carácter fuera del conjunto.
`\d`, `\w`, `\s`	Dígito, carácter alfanumérico o espacio en blanco.
`\D`, `\W`, `\S`	No dígito, no alfanumérico o no espacio en blanco.
`\t`, `\r`, `\n`	Tabulador horizontal, salto de línea o nueva línea.
`\`	Escapa un carácter con significado especial (`[` , `]` , `/` , `\`).
`\uffff`	Carácter Unicode mediante su código hexadecimal.

```
"Es hoy".match(/^hoy/) // null (no está al principio)
"Es hoy".match(/hoy$/) // ["hoy"] (está al final)
```

Operadores de repetición

Patrón	Significado
`*`	Cero o más veces.
`+`	Una o más veces.
`?`	Cero o una vez.
`{n}`	Exactamente `n` veces.
`{n,}`	`n` o más veces.
`{n, m}`	Entre `n` y `m` veces.

```
let cp = /^5[0-2][0-9]{3}$/;
cp.test("50000"); // true
cp.test("53345"); // false
```

Patrones alternativos y modificadores

Patrón o modificador	Significado
`x|y`	Busca `x` o con `y`.
`i`	Ignora mayúsculas y minúsculas.

Patrón o modificador	Significado
`g`	Busca todas las coincidencias.
`m`	Búsqueda en múltiples líneas.

```
/^(green|blue)$/i.test("Green"); // true
```

Almacenar datos en el equipo cliente

Web Storage

Los objetos de almacenamiento web, `localStorage` y `sessionStorage`, permiten guardar pares clave/valor directamente en el navegador. A diferencia de las cookies, estos datos no se envían al servidor, lo que permite almacenar información de mayor tamaño, con un límite mínimo de 5 MB en navegadores modernos. Ambos objetos comparten los mismos métodos clave:

- `.setItem(clave, valor)` → Guarda un par clave/valor.
- `.getItem(clave)` → Recupera un valor usando su clave.
- `.removeItem(clave)` → Elimina una clave y su valor asociado.
- `.clear()` → Borra todos los datos almacenados.
- `.key(indice)` → Obtiene una clave por su índice.
- `.length` → Número total de elementos almacenados.

Características principales

- **Solo admite `strings`:** Cualquier dato no `string` se convierte automáticamente. Para almacenar otros tipos de datos, usa `JSON.stringify` para guardar y `JSON.parse` para recuperar.
- **Acceso alternativo (no recomendado):** También es posible acceder mediante propiedades, como `localStorage.dato`, pero no es una práctica aconsejada.

Diferencias entre `localStorage` y `sessionStorage`

`localStorage`

- Comparte los datos entre todas las pestañas y ventanas del mismo origen.
- Los datos persisten incluso tras cerrar y volver a abrir el navegador.

`sessionStorage`

- Solo disponible en la pestaña actual del navegador (aunque compartido entre iframes del mismo origen).
- Los datos sobreviven a recargas de página, pero no a cerrar la pestaña.

Eventos relacionados con el documento y la ventana

`DOMContentLoaded`: Se activa cuando el DOM está completamente cargado, pero antes de que se carguen recursos como imágenes o estilos.

```
document.addEventListener("DOMContentLoaded", function () {
    // Código inicial
```

```
});
```

`load`: Se dispara cuando toda la página y sus recursos están completamente cargados.

```
window.onload = function () {
    // Código a ejecutar
};
```

`beforeunload`: Se ejecuta cuando el usuario intenta salir de la página.

- Permite guardar datos:

```
window.onbeforeunload = function () {
    localStorage.setItem("clave", "valor");
    return false; // Solicita confirmación al usuario
};
```

`unload`: Se activa cuando el usuario finalmente abandona la página. Está limitado a operaciones simples, sin demoras ni confirmaciones.

Estados del documento con `document.readyState`

- `loading` → Cargando el documento.
- `interactive` → DOM listo, pero aún cargando recursos.
- `complete` → Documento y recursos completamente cargados.

Carga de scripts: `async` y `defer`

`defer`: Indica que el script debe cargarse en segundo plano y ejecutarse cuando el DOM esté listo.

- Ventajas:
  - No bloquea la carga del DOM.
  - Se ejecuta antes de `DOMContentLoaded`.
- Solo aplicable a scripts externos.

`async`: Permite que el script se cargue y ejecute de forma independiente del resto del código.

- Es ideal para scripts que no interactúan con el DOM.
- Solo aplicable a scripts externos.