

Documentacion:

1. [Opcion1](#)
2. [Opcion2](#)
3. [Opcion3](#)

Focus sucede cuando el usuario ingresa con el curso dentro de un campo input

Blur sucede cuando el cursor abandona el campo en donde se encuentra


change nos permite identificar cuando el valor de un campo cambio, podemos aplicarlo sobre cualquier campo del formulario

Submit identifica cuando clickeamos sobre un campo de tipo submit

Eventos dentro de un formulario


Compartimos este paso a paso para terminar de comprender los eventos que podemos implementar en un formulario para realizar las validaciones.

1




Capturar el formulario y los inputs que queremos validar.

2




Asignar el evento **onFocus** al input de Contraseña para que cuando el usuario haga clic sobre el mismo, aparezca un `<small>` que diga "Introducir por lo menos 8 caracteres"

3




Asignar el evento **onChange** al input de Contraseña para que una vez alcanzados los 8 caracteres, deje de aparecer el `<small>`

4




Asignar el evento **onBlur** al input de Dirección para que una vez que el usuario hace clic fuera del input, podamos mandar un alert en caso de que el input esté vacío.

5



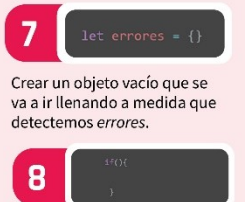
Asignar el evento **onSubmit** del formulario.

6




Prevenir que el formulario se mande usando `preventDefault()`.

7




Crear un objeto vacío que se va a ir llenando a medida que detectemos *errores*.

8



Crear condicionales que chequeen el valor o largo de los inputs. En caso de haber errores, crear una propiedad dentro del objeto *errores*.

9



En caso de que no haya errores, finalmente enviar el formulario.

Capturar el formulario

El primer objetivo será obtener el formulario. Para esto tenemos dos opciones:

```
JS let formulario = document.querySelector("form.reservation")
```

```
JS let formulario = document.forms["reservation"]
```

Eventos del formulario

El evento submit es aquel que se ejecuta cuando enviamos los datos.

```
JS formulario.addEventListener("submit", function(event){});
```

```
JS formulario.onsubmit = (event) => {}
```

Validando los campos

Podemos obtener nuestro **input** con **querySelector** para que finalmente preguntemos si el valor campo está vacío.

```
JS
event.preventDefault();
let campoNombre = document.querySelector("input.nombre");
if(campoNombre.value == ""){
    alert("El campo nombre no debe estar vacío");
}
```



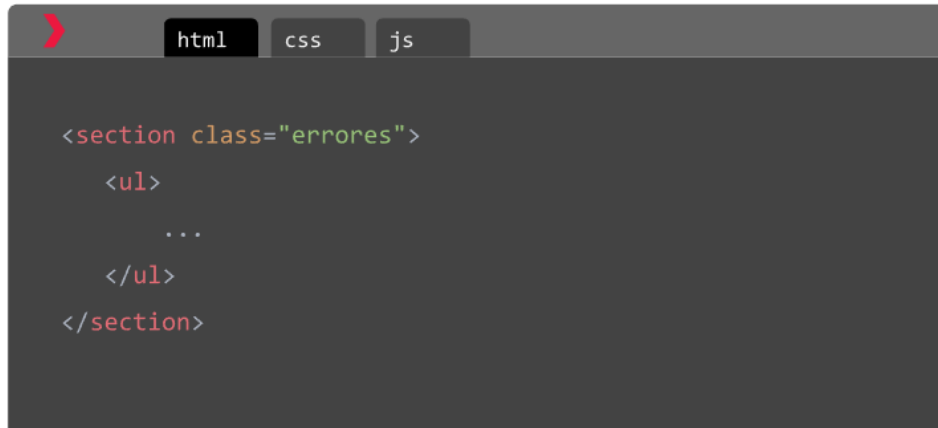
Para detener el envío de formulario usamos:
event.preventDefault()

Almacenar los errores

Creamos un **array** para acumular estos errores y cambiar nuestra lógica. Es decir, si el array no está vacío, entonces prevenimos el envío del formulario, caso contrario, el formulario se enviará.

```
JS
let errores = [];
let campoNombre = document.querySelector("input.nombre");
if(campoNombre.value == ""){
    errores.push("El campo nombre está vacío");
}
if(errores.length > 0){
    event.preventDefault();
}
```

Mostrando errores



```
<section class="errores">
  <ul>
    ...
  </ul>
</section>
```

En el javascript



```
if(errores.length > 0){
  event.preventDefault();
  let ulErrores = document.querySelector(".errores ul");
  errores.forEach(error => {
    ulErrores.innerHTML += `<li>${error}</li>`
  });
}
```

Lista de Eventos en un formulario:

1. submit:

- Descripción: Este evento se dispara cuando se envía un formulario, ya sea haciendo clic en un botón de envío o llamando al método `submit()` del formulario.
- Uso común: Validación de formularios antes del envío.

```
1 form.addEventListener('submit', function(event) {  
2     // Lógica de validación o manipulación antes de enviar el formulario  
3     // event.preventDefault(); // Evita el envío del formulario  
4 });  
5
```

2. reset:

- Descripción: Se activa cuando se hace **click** en un **botón de reinicio en un formulario**, lo que restablece todos los elementos del formulario a sus valores iniciales.
- Uso común: Realizar acciones adicionales al reiniciar el formulario.

```
1 form.addEventListener('reset', function(event) {  
2     // Lógica adicional al reiniciar el formulario  
3 });
```

3. change:

- Descripción: Se dispara cuando el **valor** de un elemento de formulario **cambia** y **pierde el foco**.
- Uso común: Validación en tiempo real o actualización de otros campos dependientes.



```
1 input.addEventListener('change', function(event) {  
2     // Lógica para manejar el cambio en el valor del input  
3 });
```

4. input:


- Descripción: Similar a change, pero **se activa inmediatamente** después de que cambia el valor de un elemento de formulario.
- Uso común: Para acciones que deben ocurrir inmediatamente después de cada cambio.



```
1 input.addEventListener('input', function (event) {  
2     // Lógica para manejar el cambio inmediato en el valor del input  
3 });
```

5. focus:

- Descripción: Se activa cuando un elemento de formulario **obtiene el foco**.
- Uso común: Cambios en la interfaz de usuario al seleccionar un campo.



```
1 input.addEventListener('focus', function (event) {  
2     // Lógica para manejar el foco en el input  
3 });
```

6. blur:

- Descripción: Se dispara cuando un elemento de formulario **pierde el foco**.
- Uso común: Validación o acciones después de que el usuario ha terminado de ingresar datos en un campo.



```
1 input.addEventListener('blur', function (event) {  
2     // Lógica para manejar la pérdida de foco en el input  
3 });
```