Uso de JavaScript (II): Eventos y formularios

Departamento de Lenguajes y Sistemas Informáticos

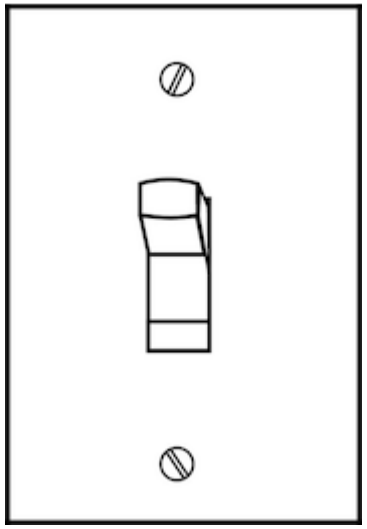
Universidad de Sevilla

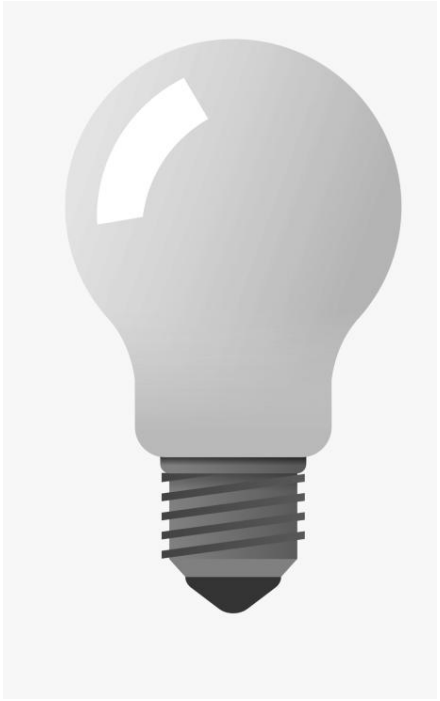
Daniel Ayala, Carlos Arévalo, José Calderón, Margarita Cruz, Inma Hernández, David Ruiz

Contenidos

Eventos de programación

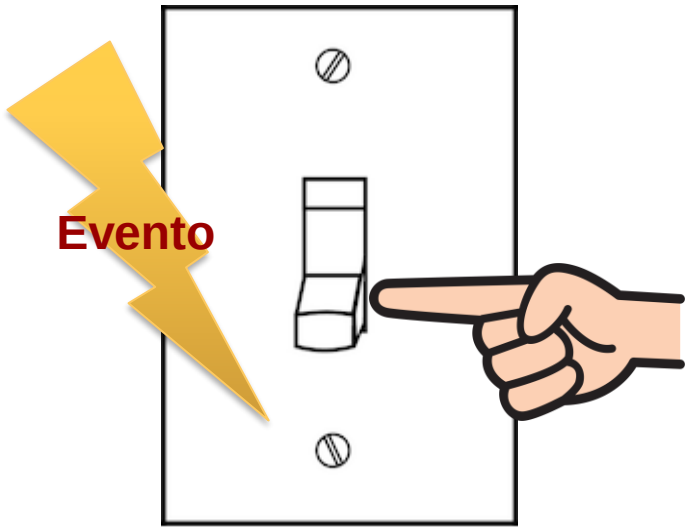
- Son “cosas” que les pasan a los elementos del lenguaje

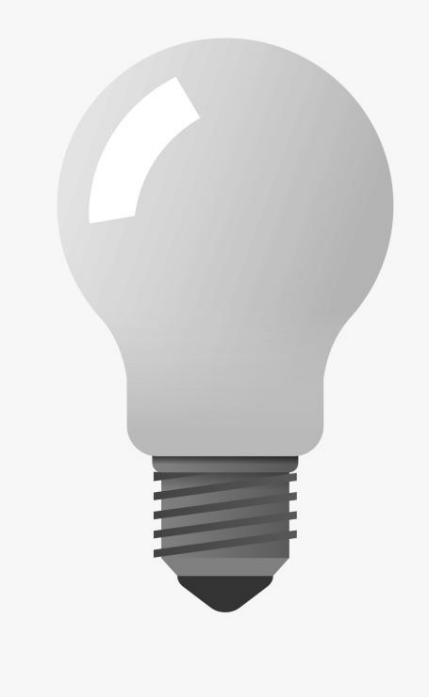




Eventos de programación

- Son “cosas” que les pasan a los elementos del lenguaje
- El lenguaje permite reaccionar a esas “cosas” ejecutando código





Eventos de programación

- Son “cosas” que les pasan a los elementos del lenguaje
- El lenguaje permite reaccionar a esas “cosas” ejecutando código

Código

encenderLuz()

Eventos en HTML

- En el caso de las páginas web, los eventos son “cosas” que les ocurren a los elementos HTML. Puede ser
  - A un único elemento
  - A un conjunto de elementos
  - A la página entera
- Los eventos se producen en el contexto del navegador que está mostrando la página, por lo que también afectan al propio navegador

Tipos de eventos

- Existe una gran variedad de tipos de eventos, según el elemento de que se trate. Algunos ejemplos comunes de cosas que se pueden controlar mediante eventos:
  - Qué hacer cada vez que se carga la página
  - Qué hacer cada vez que se cierra la página
  - Qué hacer cuando el usuario pulsa un elemento (o pasa el ratón por encima de él)
  - Qué hacer cuando se envía un formulario (por ejemplo, validar los datos que ha introducido el usuario antes de enviar a backend)
  - Qué hacer cuando el usuario introduce un valor usando el teclado (por ejemplo, para hacer la validación a medida que el usuario escribe, en vez de esperar a enviar el formulario)
  - Qué hacer cuando el usuario redimensiona la ventana del navegador
- Lista completa de tipos de eventos:
  - https://www.w3schools.com/jsref/dom_obj_event.asp
  - https://developer.mozilla.org/en-US/docs/Web/Events

Categorías de eventos

- Según el tipo de elemento que emite el evento, distinguimos:
  - Eventos del objeto **window**
    - redimensionar la ventana
  - Eventos del objeto **window.screen**
    - cambiar la orientación del dispositivo
  - Eventos del objeto **document**
    - carga de la página, interacción con el usuario, salida de la página
  - Objetos del árbol **DOM**
    - Interacciones o modificaciones de los elementos
  - Objetos **XMLHttpRequest**
    - Para peticiones a través de la red (lo veremos más adelante)
  - Objetos **multimedia** (audio, video)
    - Cambio en el estado de la reproducción

Contenidos

Como manejar los eventos

- Añadir un manejador al elemento
- Manipular el DOM para añadir el manejador al elemento
- Añadir un listener

Event handlers – atributo manejador

- Para definir cómo se debe reaccionar a un evento que afecta a un determinado elemento, se añade a dicho elemento un manejador de eventos (***event handler***), con la siguiente sintaxis:

```
<elemento manejador='código JavaScript'>
```

- El código JavaScript para manejar un determinado evento puede ser muy largo como para añadirlo directamente al valor del atributo. Lo habitual es encapsularlo en una función, y llamarla desde el manejador:

```
<elemento manejador='function()'>
```

- Asociar un trozo de código (o función) a un evento se conoce como registrar un manejador de eventos (*registering an event handler*).

Event handlers – manipulación del DOM

- Otra forma de definir los handlers es obteniendo una referencia al elemento como objeto. Cada objeto tiene una propiedad por cada tipo de evento:

```
let btn = document.getElementById("boton");
btn.onclick = function() {
    //código de la función
}
```

- O, si se ha definido la función previamente, asignando directamente a la propiedad el nombre de la función

```
function manejador() {
    //código de la función
}
btn.onclick = manejador;
```

En este caso, se usa el nombre del tipo de evento con el prefijo 'on' (onclick, onmouseover, …)

Event Listeners

- También se puede definir el comportamiento cuando se produce un evento con un *event listener*:

```
element.addEventListener(evento, función);
```

- Ejemplo:

```
btn.addEventListener("click", function(){
      alert("Event listener");
});
```

En este caso, se usa el nombre del tipo de evento sin 'on' (click, mouseover, …)

Event handlers vs Event listeners

- A los manejadores de eventos a veces se le llama escuchadores de eventos (*event listeners*), dado que en ambos casos se trata de **código que se ejecuta cuando se produce un evento**
- Conceptualmente no son lo mismo
  - El listener está pendiente de que ocurra el evento (el listener *notifica* al handler)
  - Un handler es la combinación entre
    - El listener que informa cuando se produce el evento
    - El código que se ejecuta como respuesta al evento (*callback function*)
- En la práctica, se usan ambos términos de forma intercambiable, aplicando una **metonimia** (se designa al listener por el nombre del handler para el que “trabaja”)

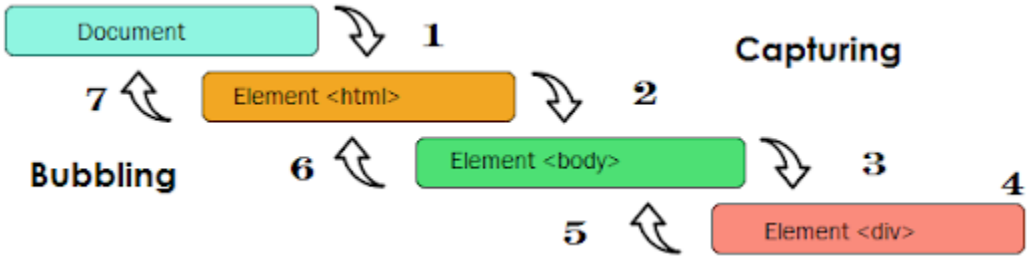
Diferencias entre handlers y listeners

- Si se añaden varios handlers al mismo evento**, sólo se ejecutará el último** (se sobreescriben)
- Si se añaden varios listeners al mismo evento, **todos ellos se ejecutarán** cuando se produzca el evento
- Lo mismo si se añade un handler y varios listeners
- Al método `addEventListener()` se le puede añadir un tercer parámetro booleano opcional que indica si el código se debe ejecutar en la fase de captura o de bubbling.
- Al asociar una función a un evento mediante el método `addEventListener()`, la función recibirá automáticamente un parámetro "`event`" que representa al evento y contiene propiedades útiles.

Modos de propagación de eventos

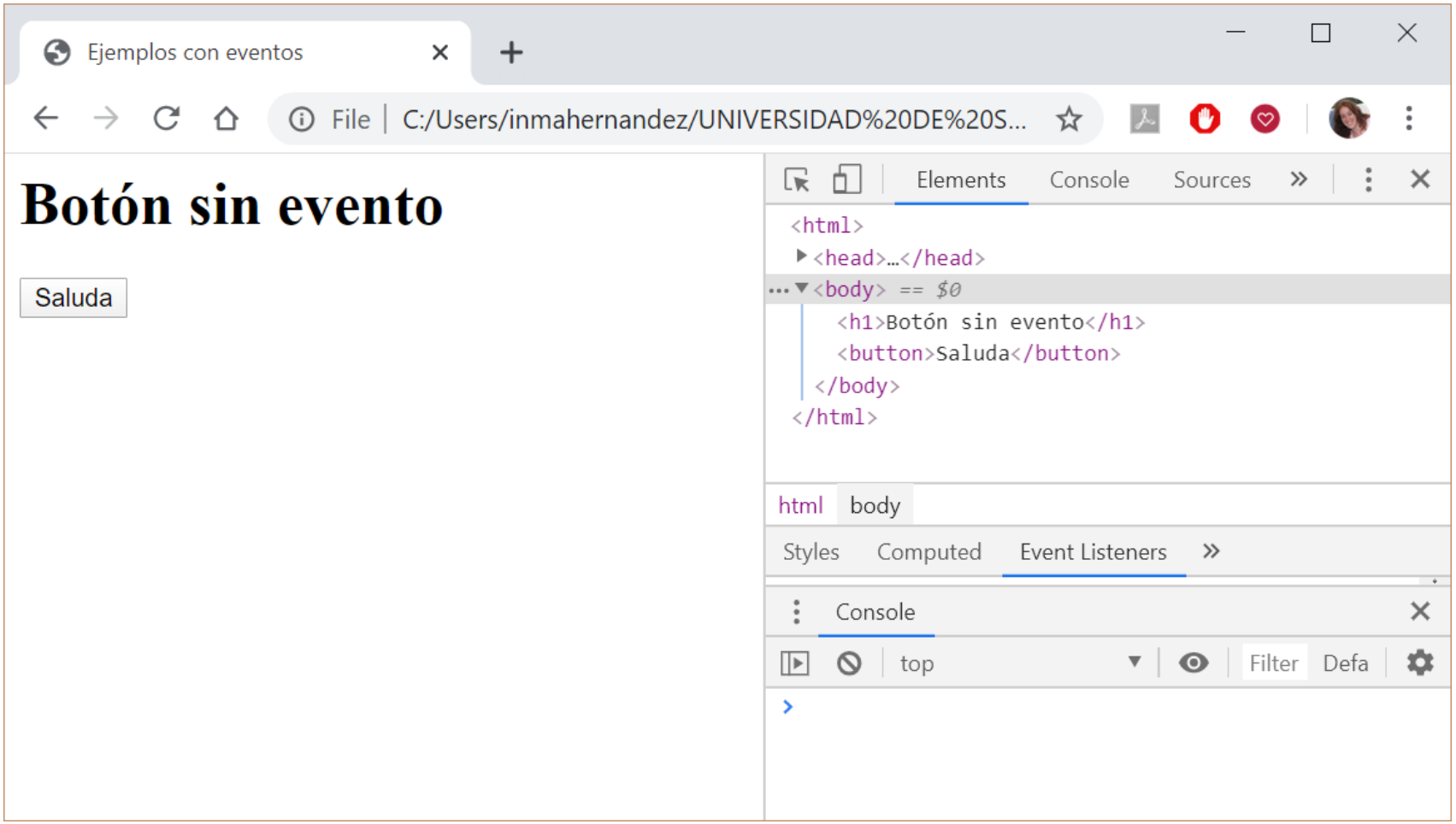
- En HTML, hay una estructura anidada de elementos
  - Ej: Un <p> dentro de un , dentro de <body>…
- Si hay varios elementos anidados, y cada uno de ellos tiene un manejador distinto para un tipo de evento (ej: click), ¿En qué orden se ejecuta el código asociado a los manejadores?
- Hay dos modos de propagación
  - Capturing: El orden es del elemento más externo al más interno
    - En el ejemplo, primero se ejecuta el código del manejador de <body>, luego el de y luego el de <p>
  - Bubbling: El orden es del elemento más interno al más externo
    - En el ejemplo, primero se ejecuta el código del manejador de <p>, luego el de y luego el de <body>

Modos de propagación de un evento

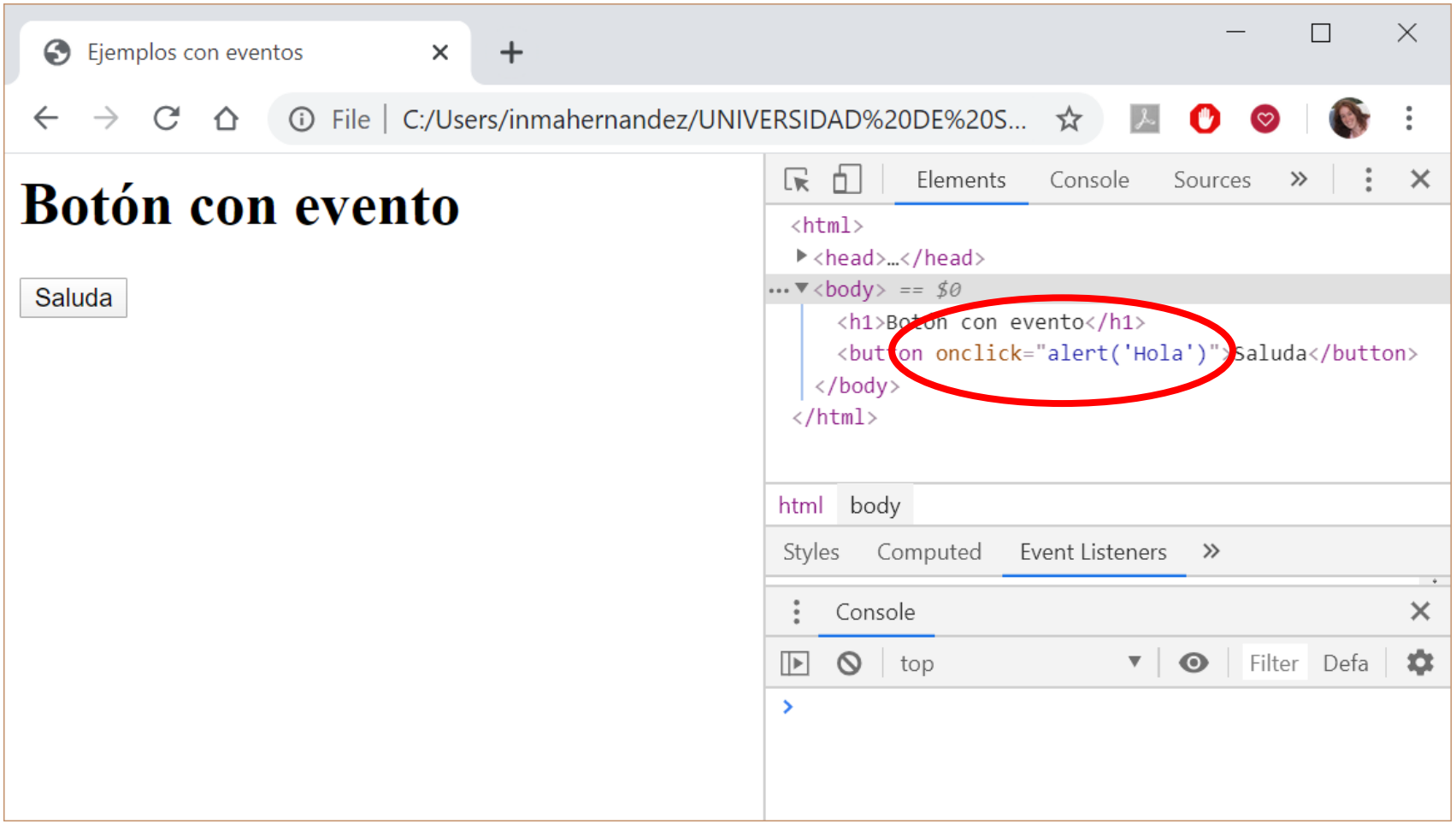


Contenidos

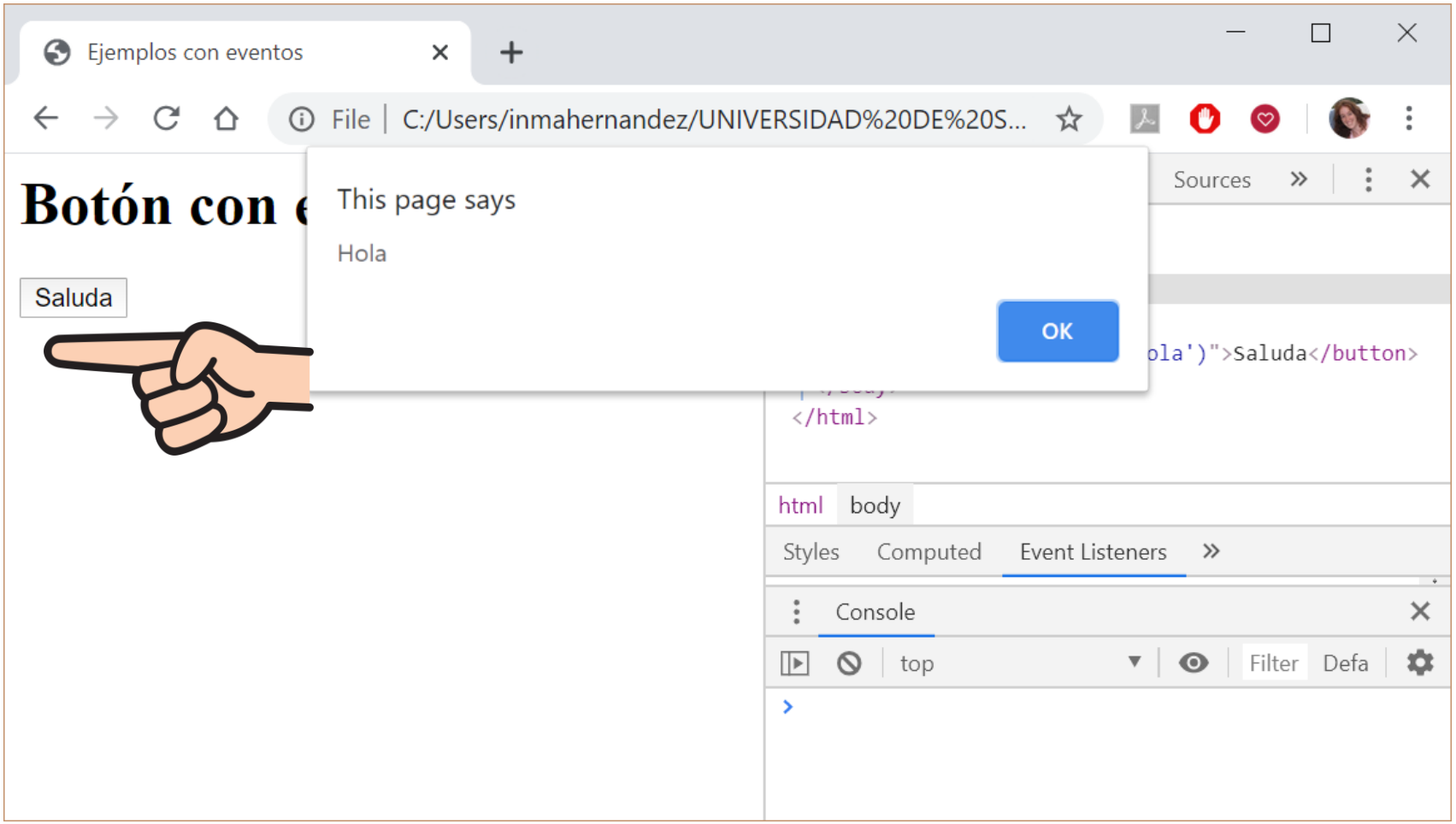
Eventos en JavaScript - Ejemplo



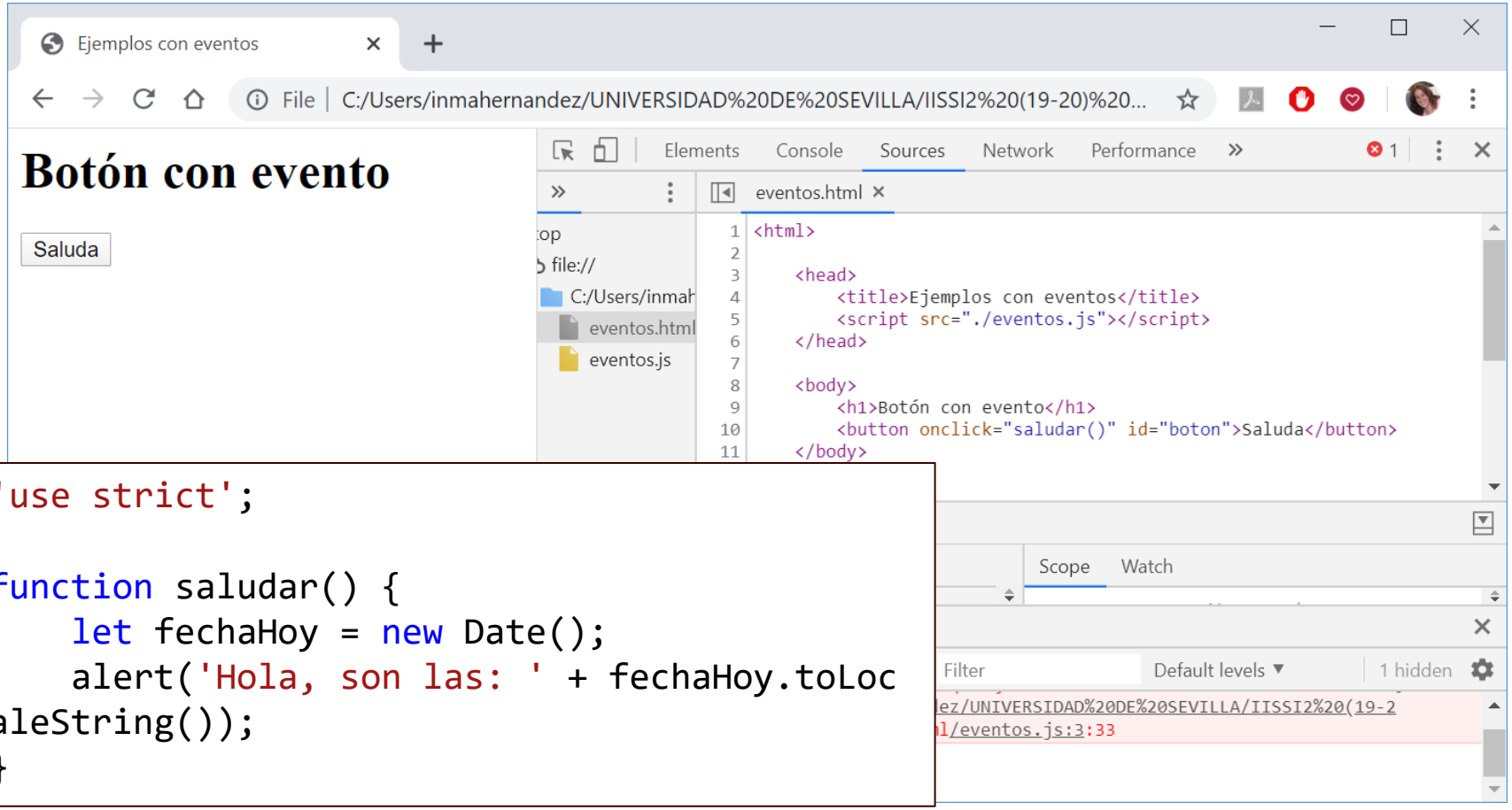
Eventos en JavaScript - Ejemplo



Eventos en JavaScript - Ejemplo

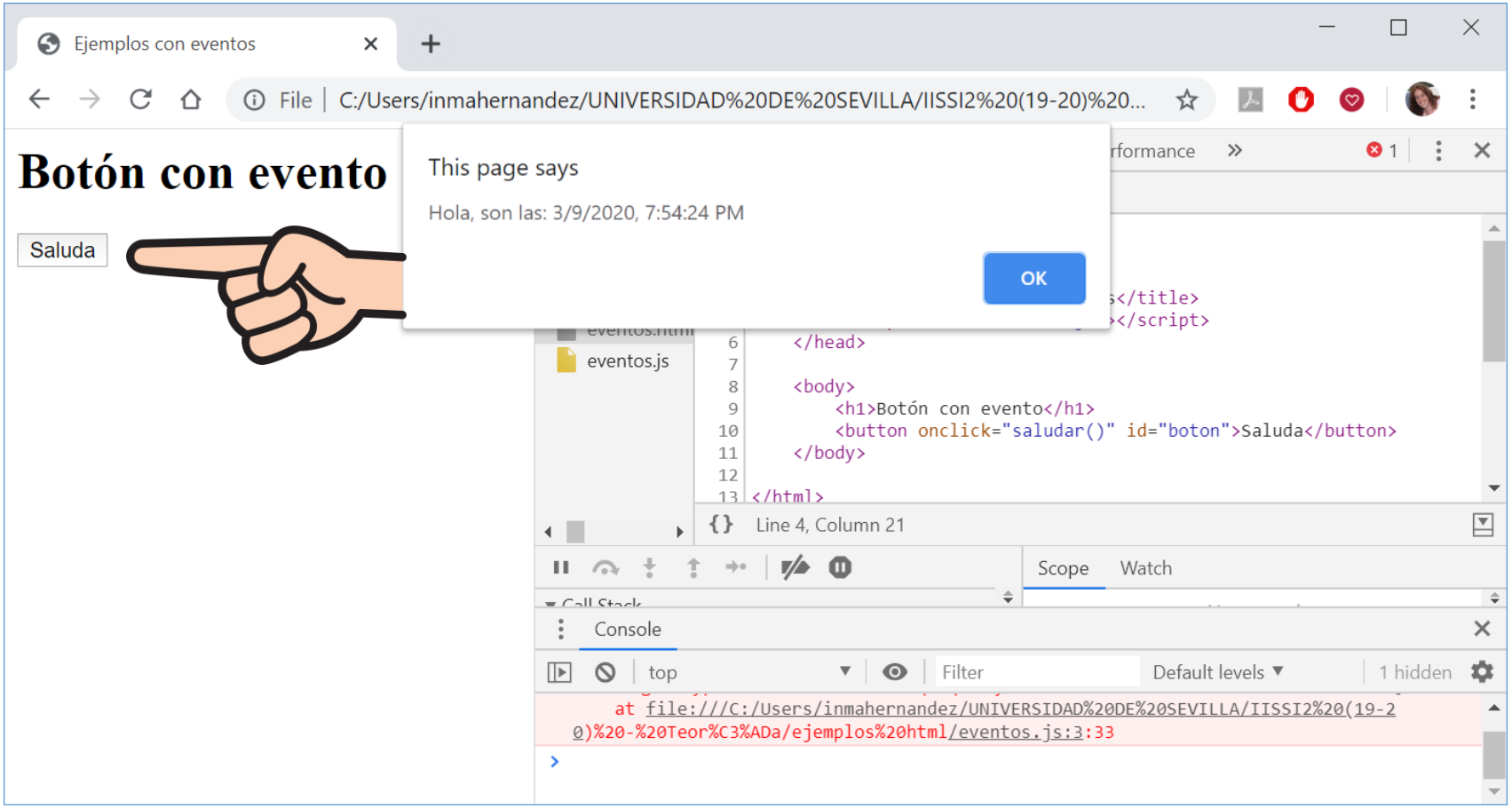


Eventos en JavaScript - Ejemplo

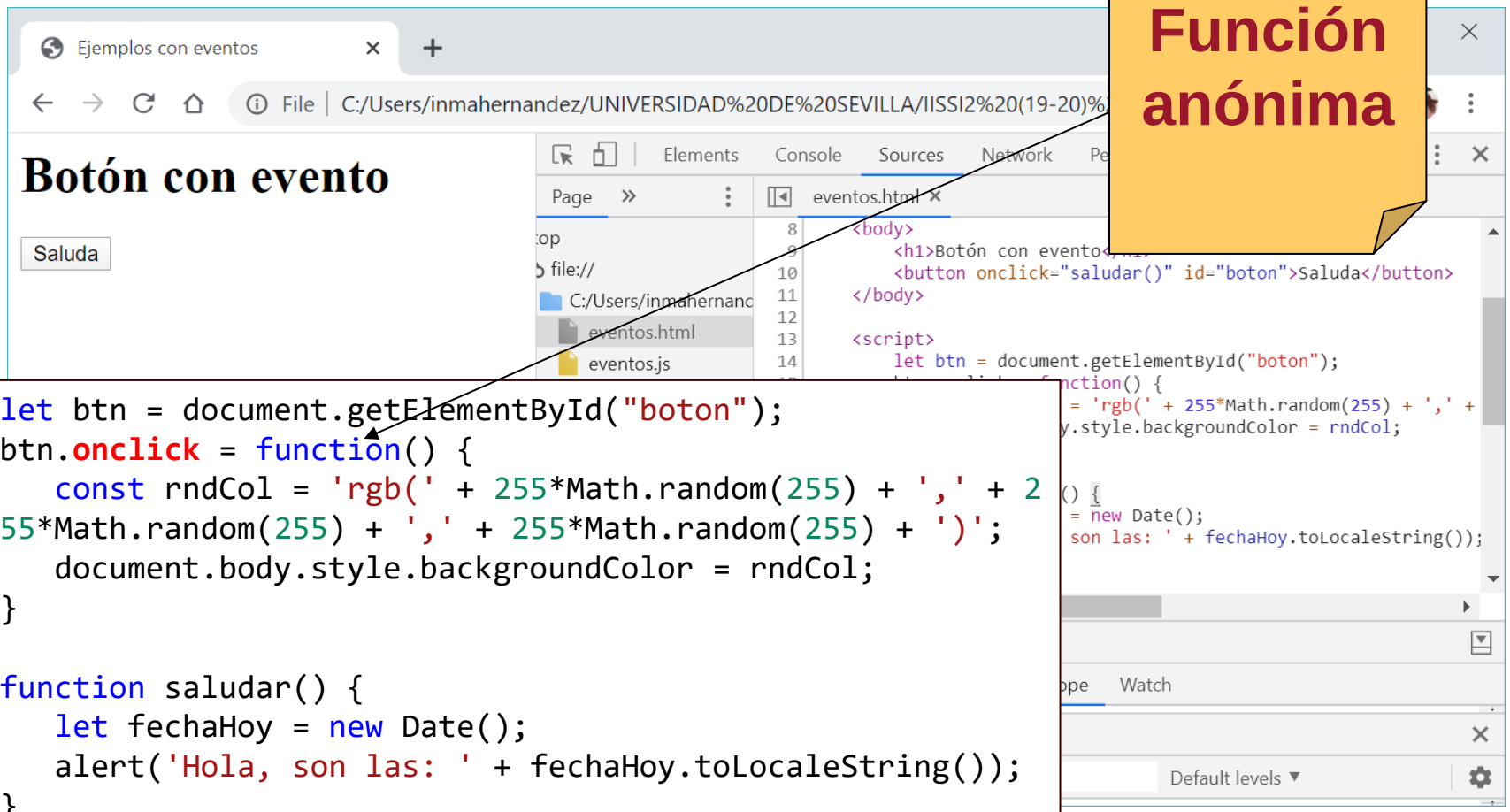



```
'use strict';

function saludar() {
    let fechaHoy = new Date();
    alert('Hola, son las: ' + fechaHoy.toLoc
aleString());
}
```

Eventos en JavaScript - Ejemplo

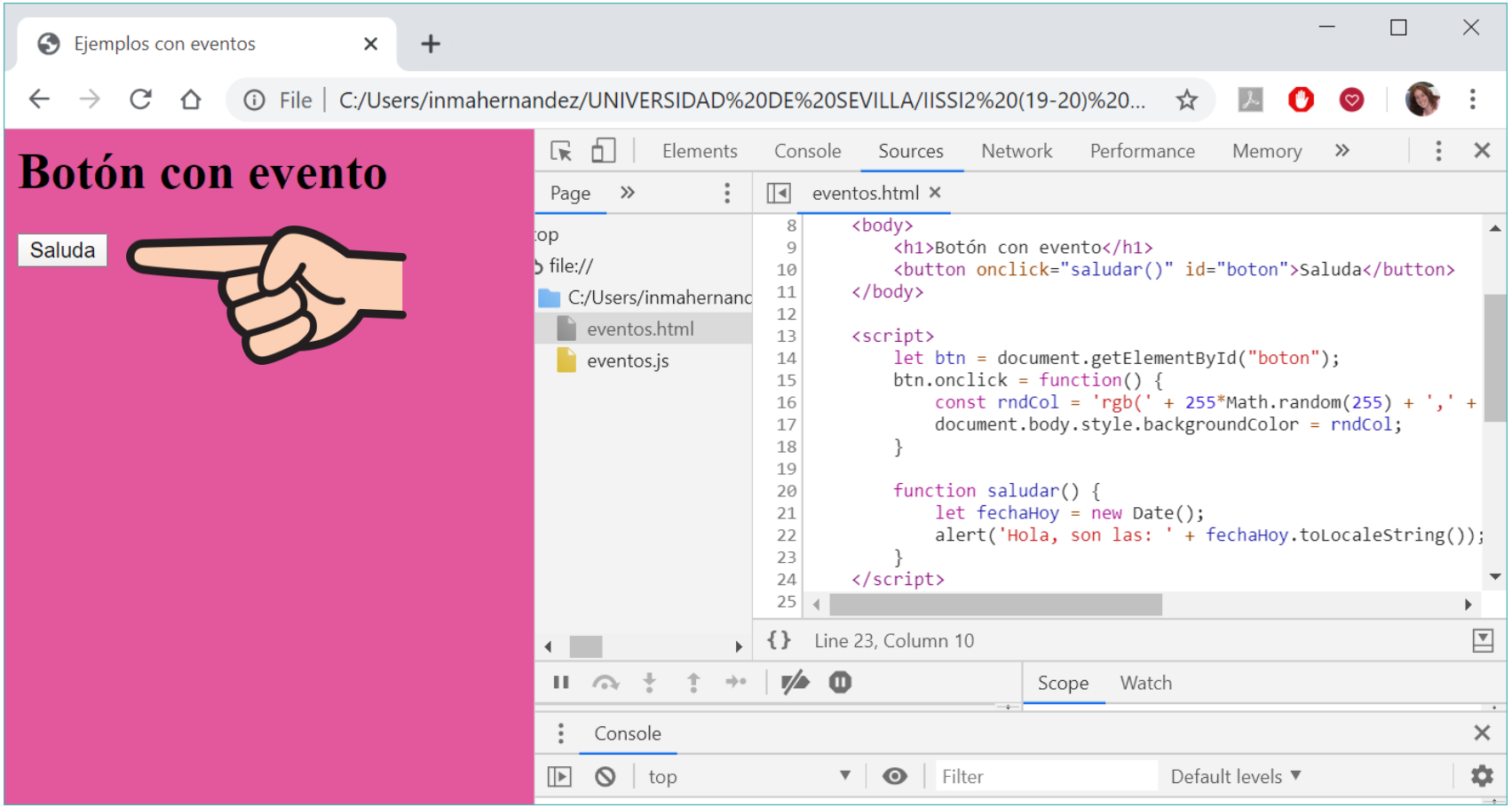


Varios handlers

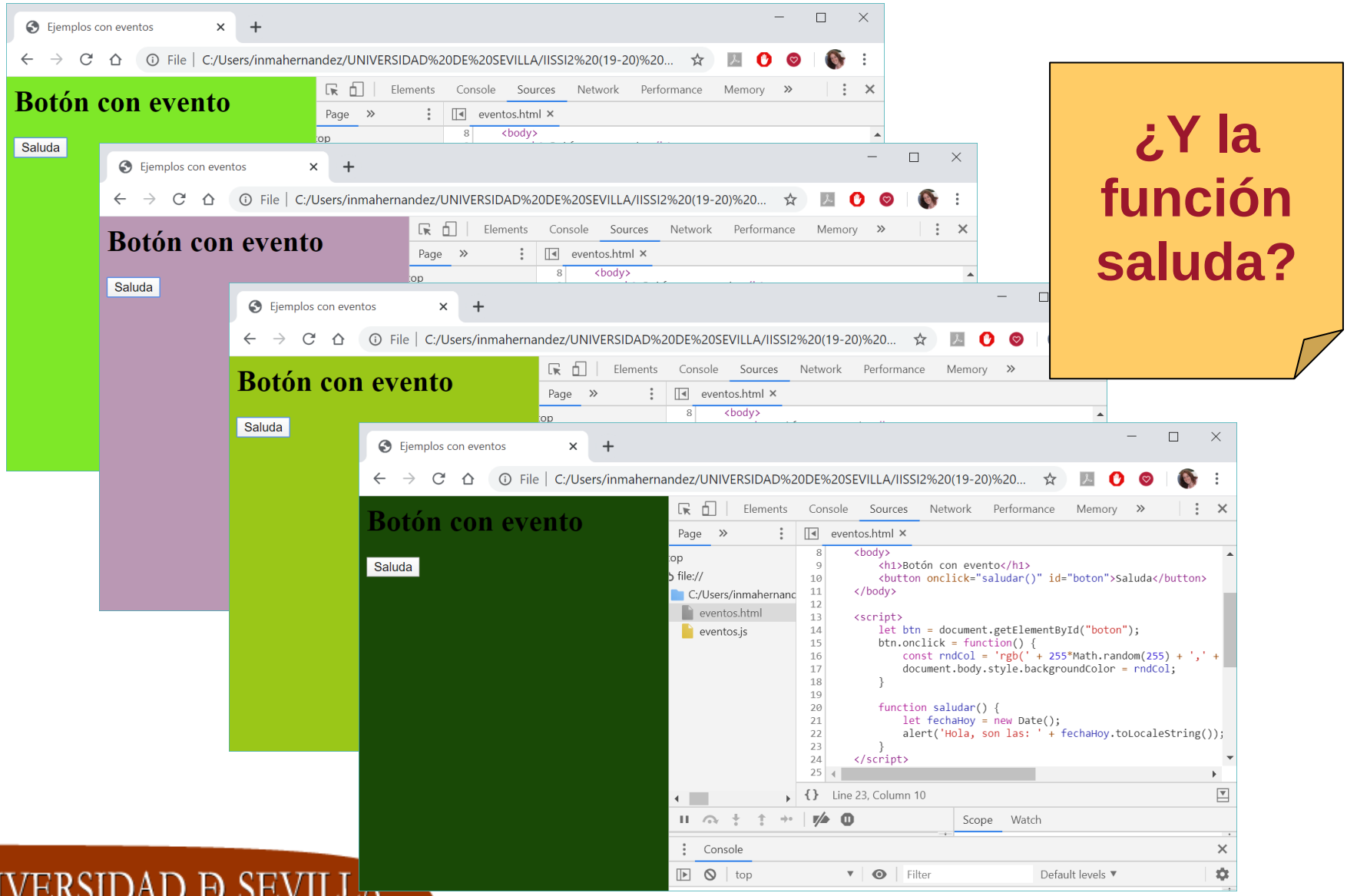



```
let btn = document.getElementById("boton");
btn.onclick = function() {
   const rndCol = 'rgb(' + 255*Math.random(255) + ',' + 2
55*Math.random(255) + ',' + 255*Math.random(255) + ')';
   document.body.style.backgroundColor = rndCol;
}

function saludar() {
   let fechaHoy = new Date();
   alert('Hola, son las: ' + fechaHoy.toLocaleString());
}
```

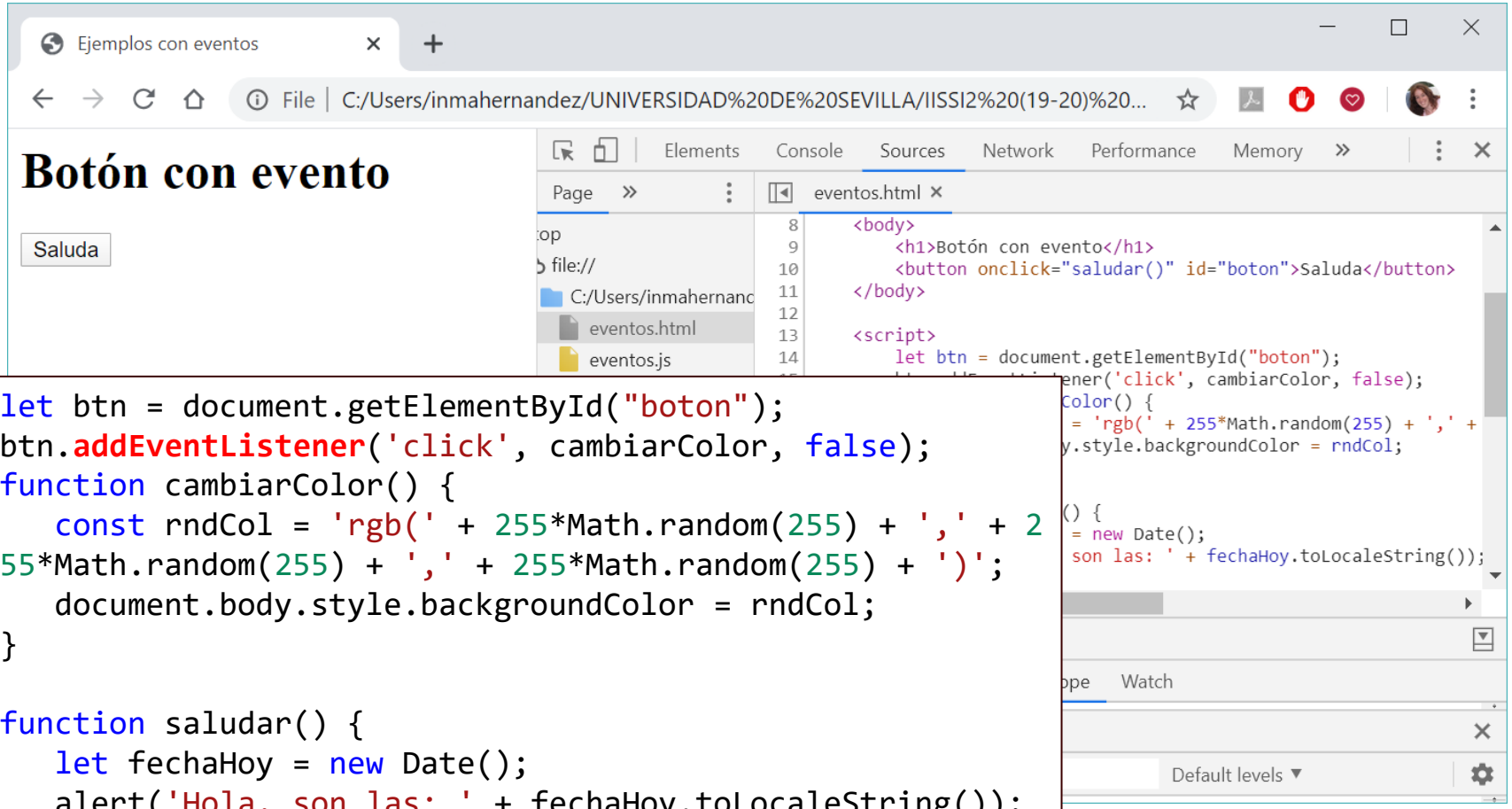
Varios handlers



Más clicks…

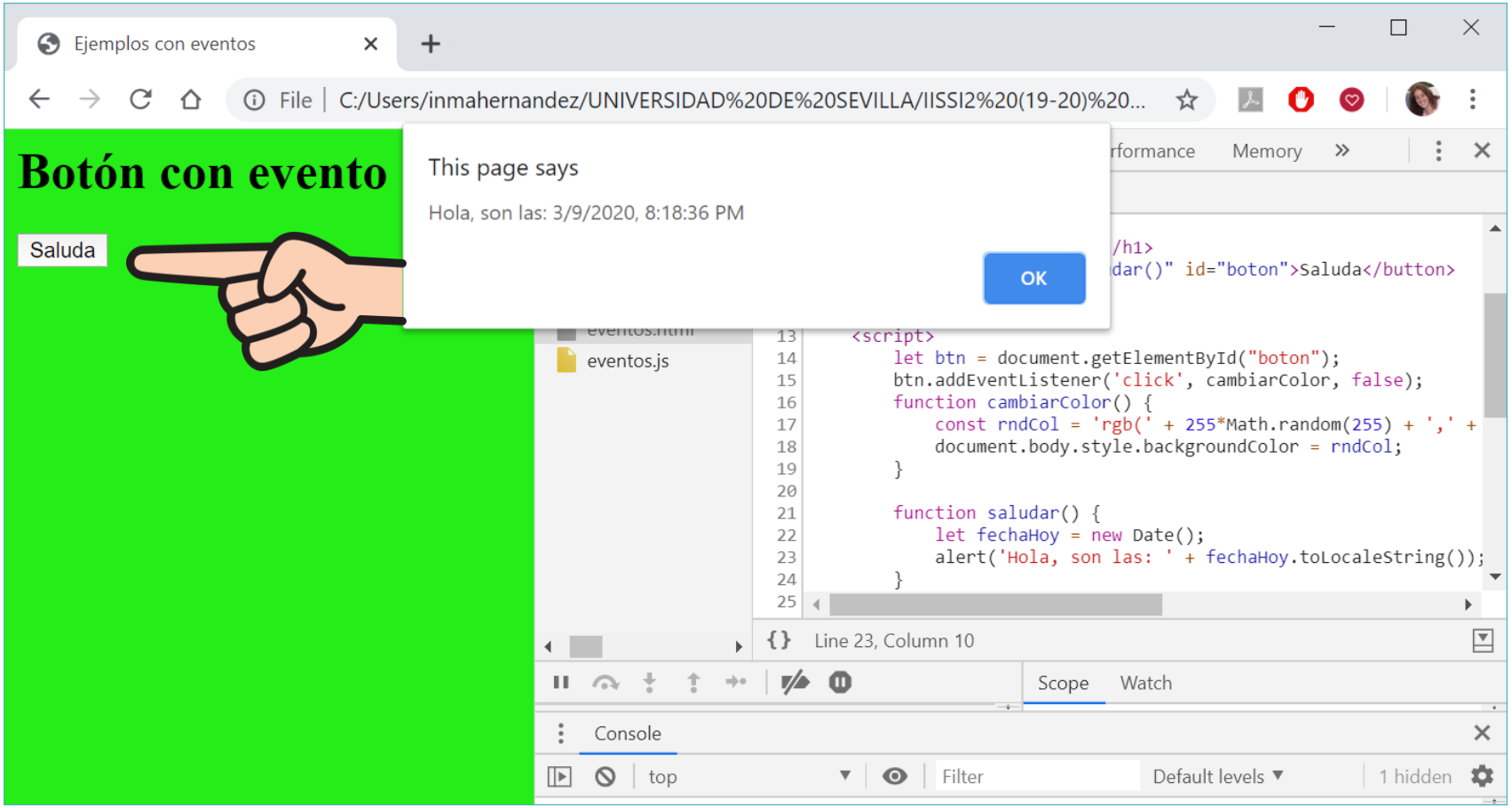

Ejemplos con eventos
Botón con evento
Saluda
eventos.html
eventos.js
¿Y la función saluda?

Event listeners

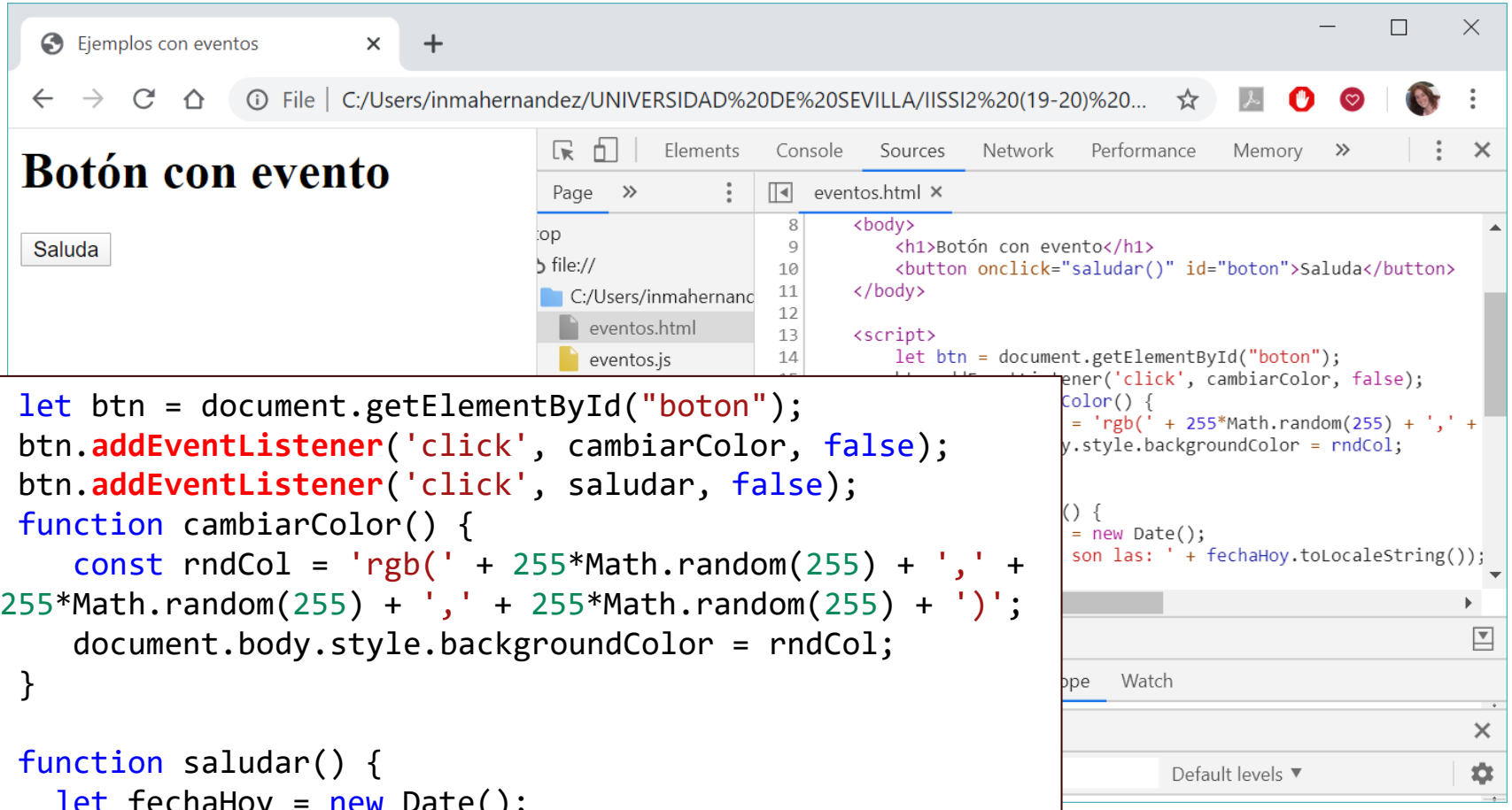



```
let btn = document.getElementById("boton");
btn.addEventListener('click', cambiarColor, false);
function cambiarColor() {
   const rndCol = 'rgb(' + 255*Math.random(255) + ',' + 2
55*Math.random(255) + ',' + 255*Math.random(255) + ')';
   document.body.style.backgroundColor = rndCol;
}

function saludar() {
   let fechaHoy = new Date();
   alert('Hola, son las: ' + fechaHoy.toLocaleString());
}
```

Event listeners

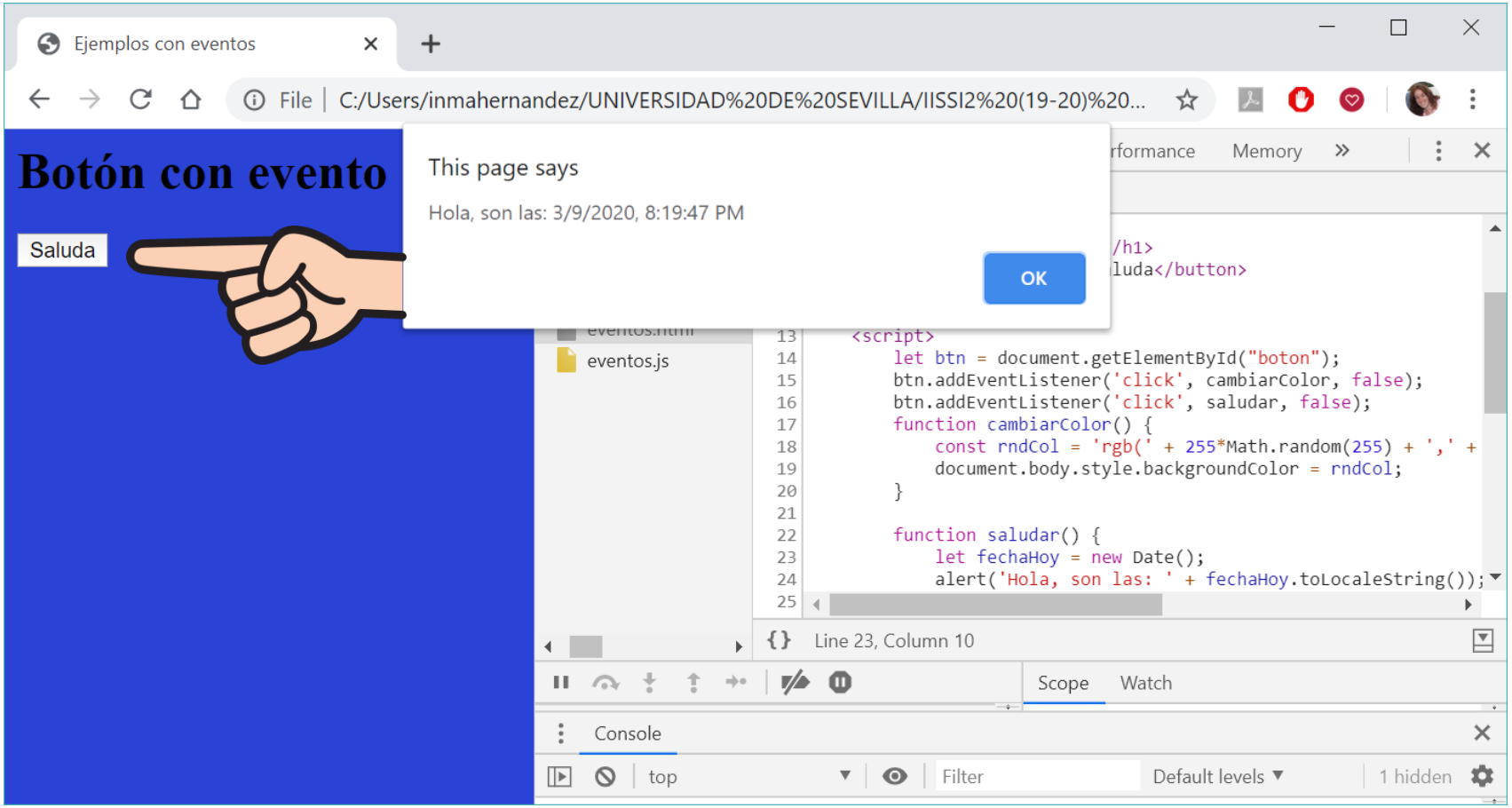


Varios Event Listeners

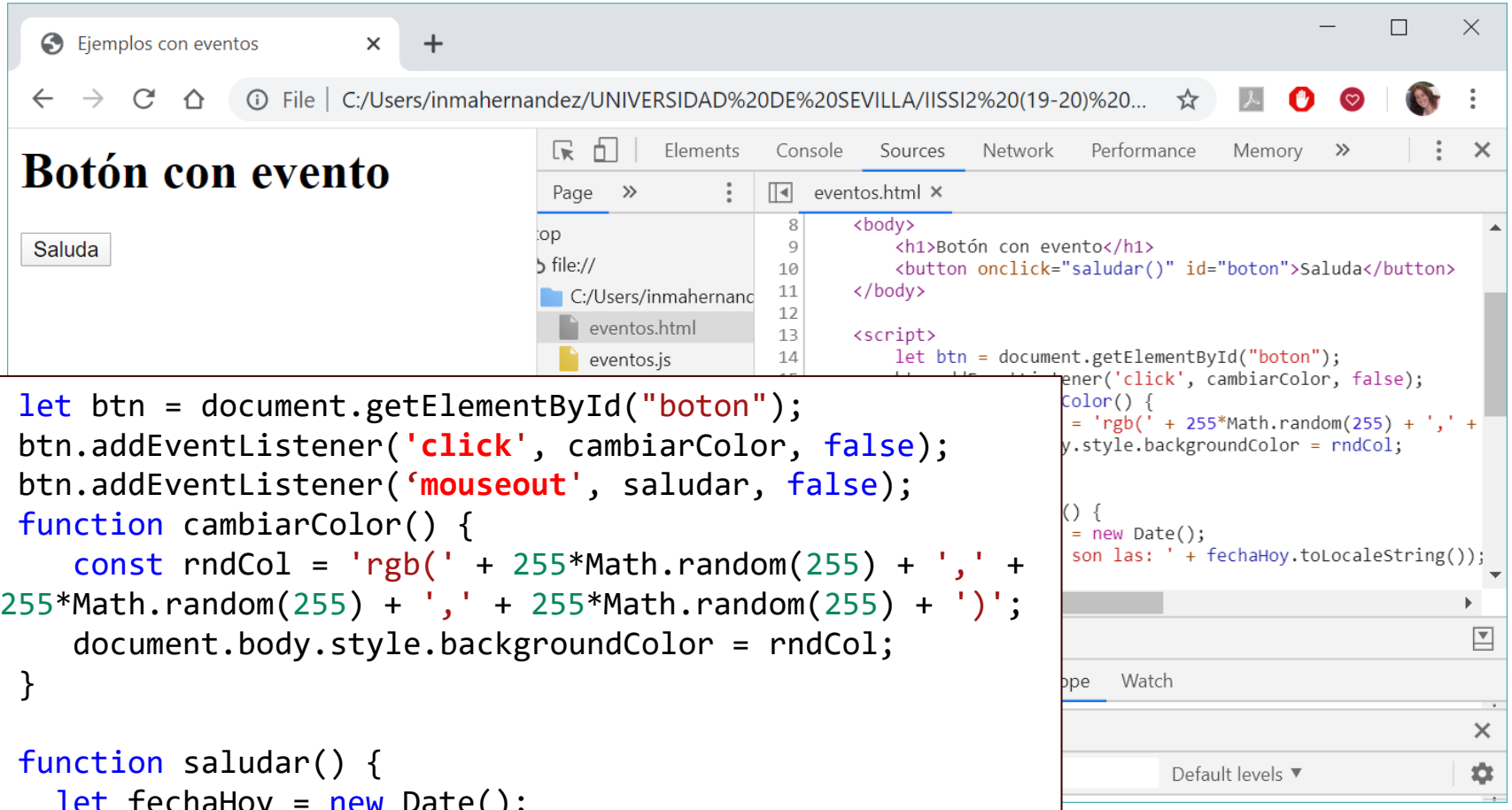



```
let btn = document.getElementById("boton");
btn.addEventListener('click', cambiarColor, false);
btn.addEventListener('click', saludar, false);
function cambiarColor() {
    const rndCol = 'rgb(' + 255*Math.random(255) + ',' +
255*Math.random(255) + ',' + 255*Math.random(255) + ')';
    document.body.style.backgroundColor = rndCol;
}

function saludar() {
  let fechaHoy = new Date();
  alert('Hola, son las: ' + fechaHoy.toLocaleString());
}
```

Varios Event Listeners

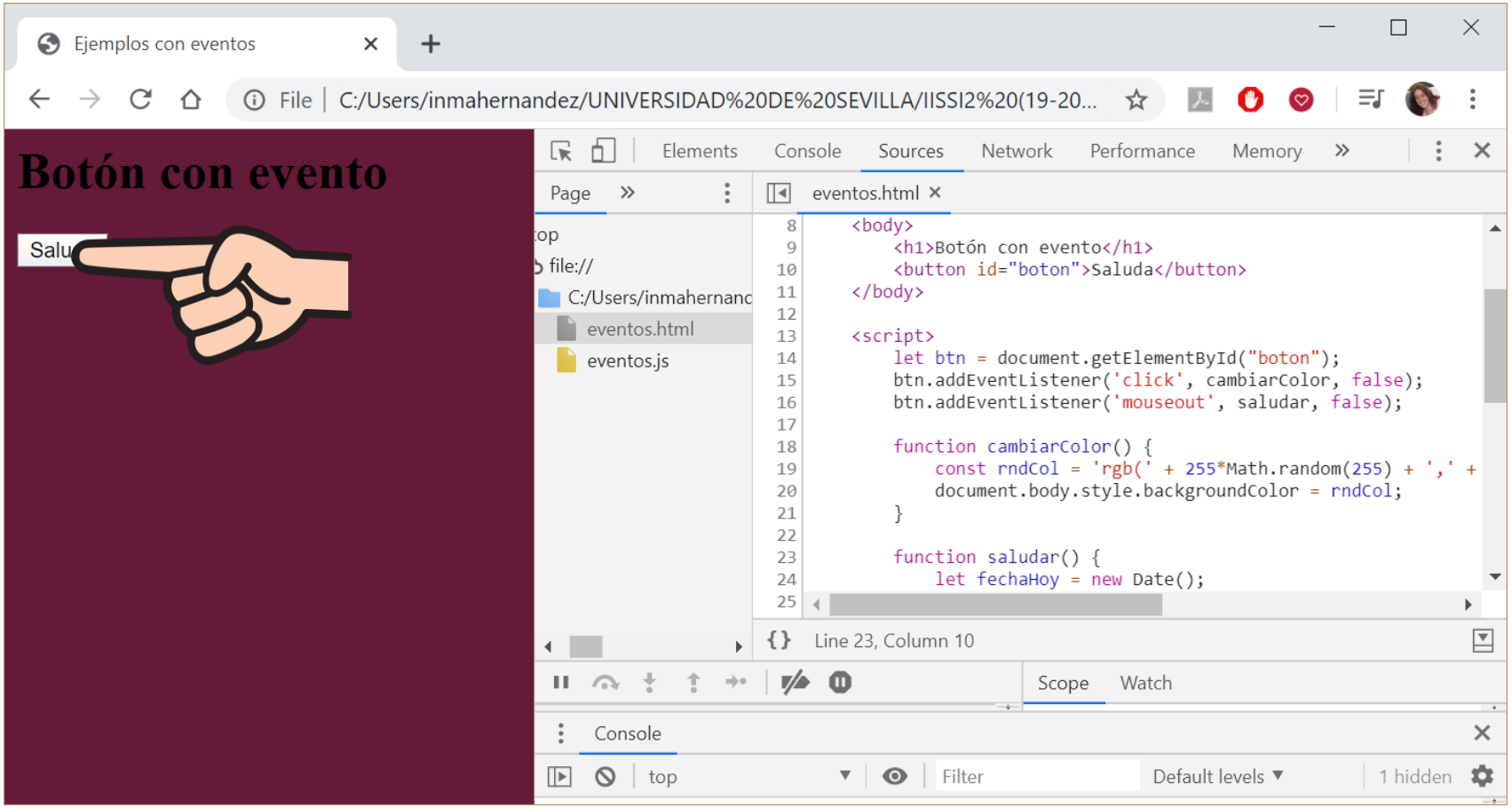


Event Listeners para distintos tipos de evento

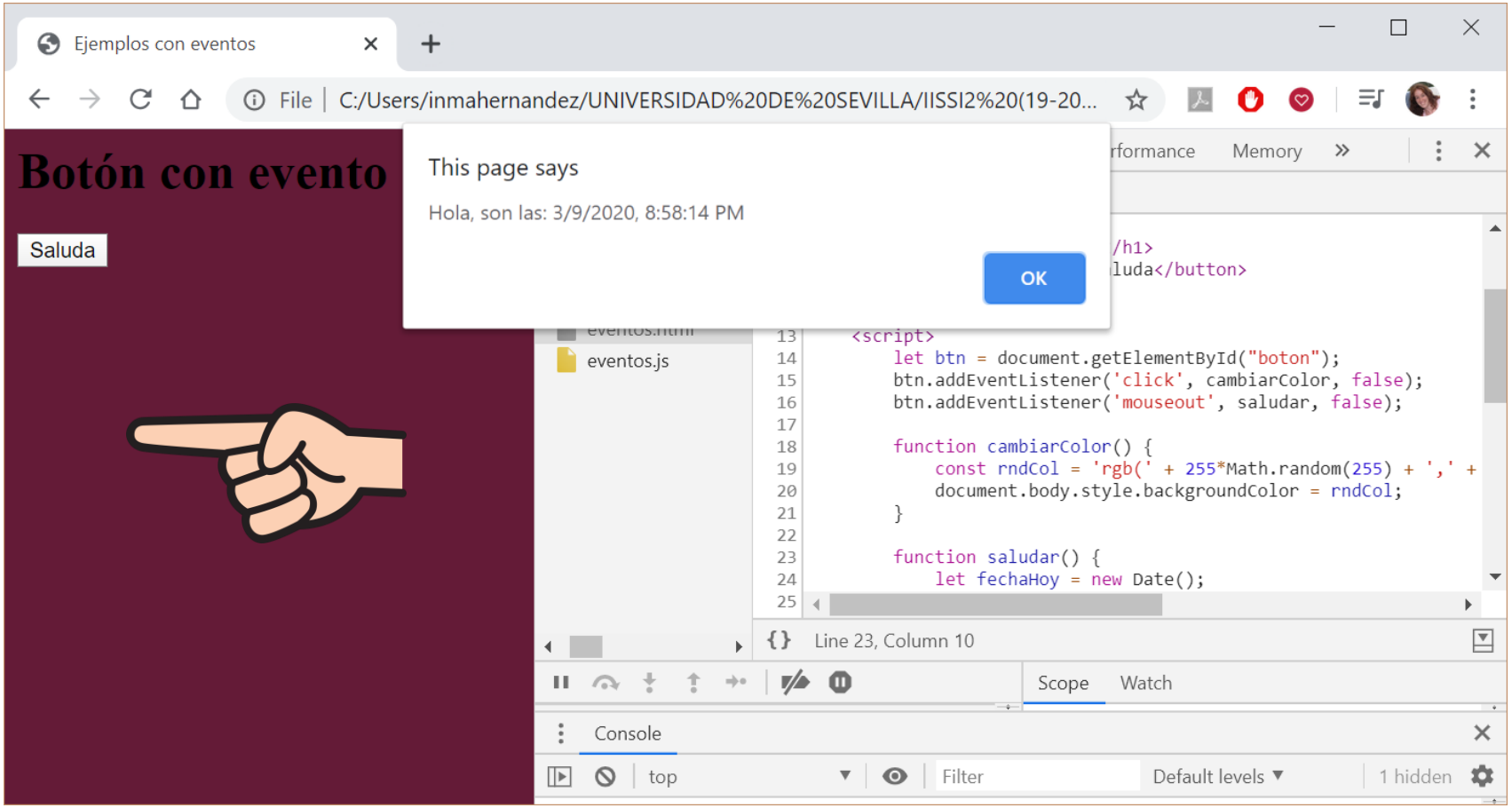



```
let btn = document.getElementById("boton");
btn.addEventListener('click', cambiarColor, false);
btn.addEventListener('mouseout', saludar, false);
function cambiarColor() {
    const rndCol = 'rgb(' + 255*Math.random(255) + ',' +
255*Math.random(255) + ',' + 255*Math.random(255) + ')';
    document.body.style.backgroundColor = rndCol;
}

function saludar() {
  let fechaHoy = new Date();
  alert('Hola, son las: ' + fechaHoy.toLocaleString());
}
```

Event Listeners para distintos tipos de evento



Event Listeners para distintos tipos de evento



Otros métodos

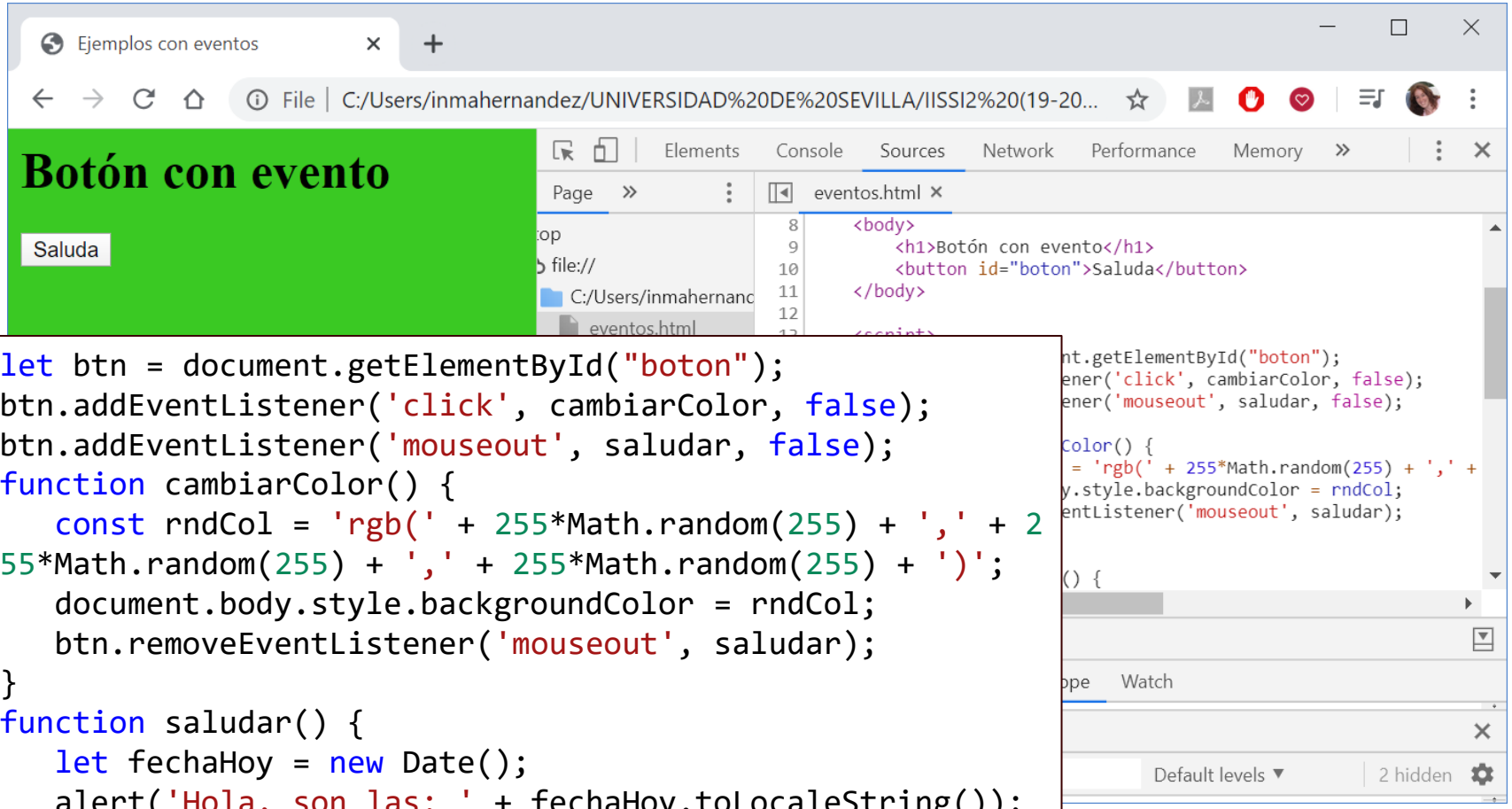
- removeEventListener(): elimina un manejador del elemento

- Forma de uso:

```
elemento.removeEventListener("evento", función)
```

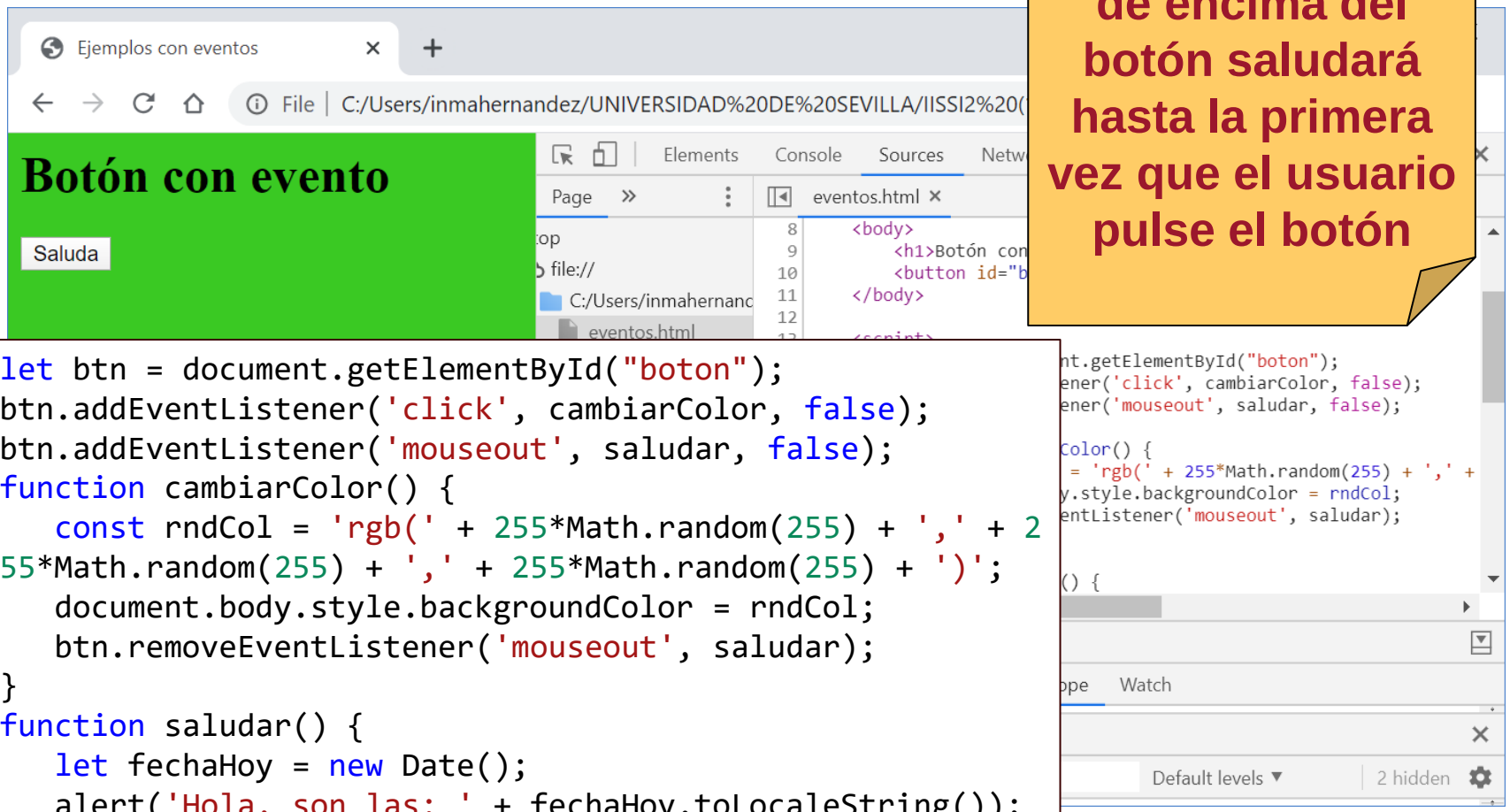
- Efecto: La función deja de estar asociada al manejador del evento (la próxima vez que se produzca el evento ya no se ejecutará).

Eliminar manejadores - Ejemplo




```
let btn = document.getElementById("boton");
btn.addEventListener('click', cambiarColor, false);
btn.addEventListener('mouseout', saludar, false);
function cambiarColor() {
   const rndCol = 'rgb(' + 255*Math.random(255) + ',' + 2
55*Math.random(255) + ',' + 255*Math.random(255) + ')';
   document.body.style.backgroundColor = rndCol;
   btn.removeEventListener('mouseout', saludar);
}
function saludar() {
   let fechaHoy = new Date();
   alert('Hola, son las: ' + fechaHoy.toLocaleString());
}
```

Eliminar manejadores - Ejemplo



Al quitar el ratón de encima del botón saludará hasta la primera vez que el usuario pulse el botón

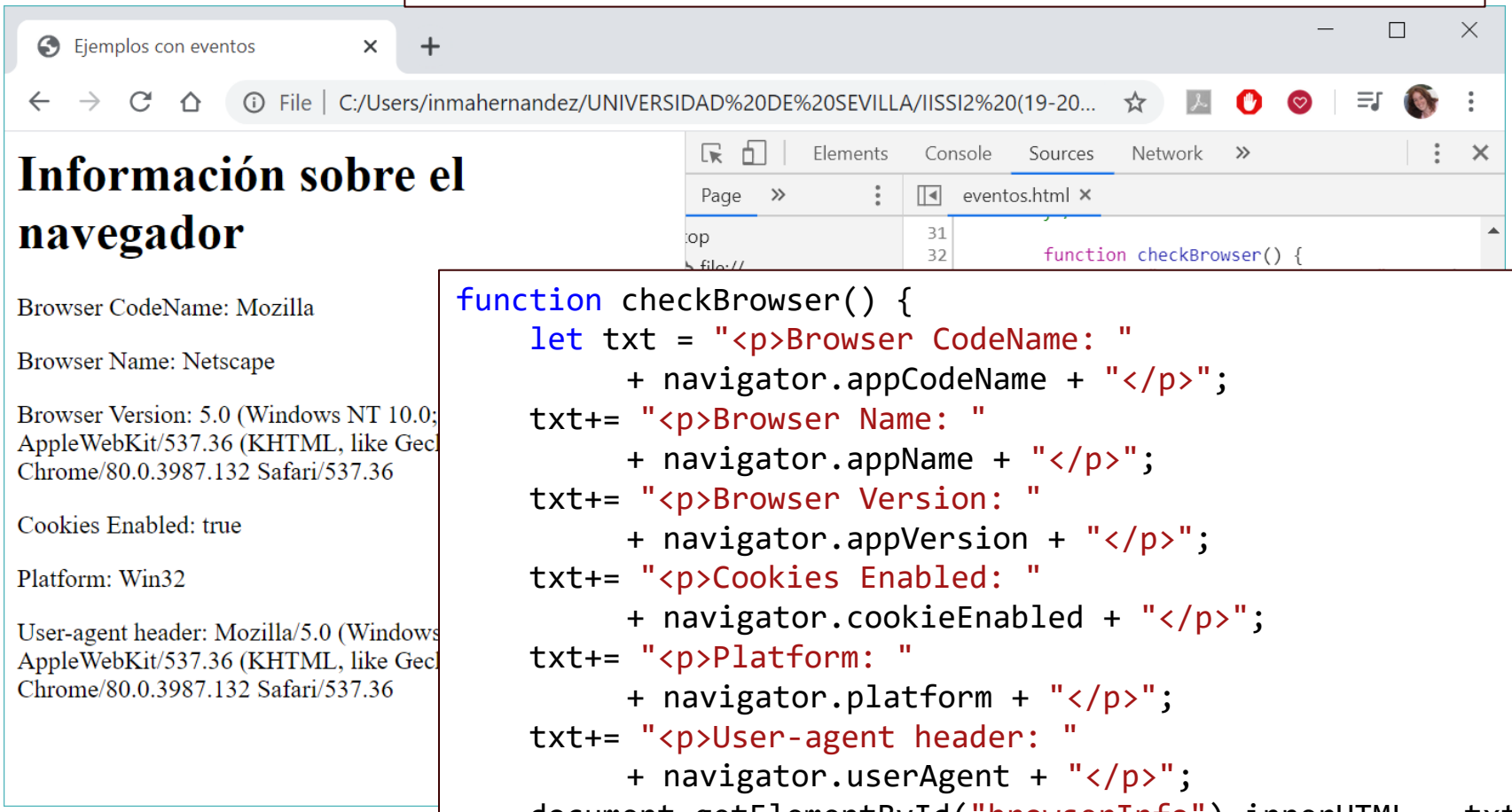
```
let btn = document.getElementById("boton");
btn.addEventListener('click', cambiarColor, false);
btn.addEventListener('mouseout', saludar, false);
function cambiarColor() {
   const rndCol = 'rgb(' + 255*Math.random(255) + ',' + 2
55*Math.random(255) + ',' + 255*Math.random(255) + ')';
   document.body.style.backgroundColor = rndCol;
   btn.removeEventListener('mouseout', saludar);
}
function saludar() {
   let fechaHoy = new Date();
   alert('Hola, son las: ' + fechaHoy.toLocaleString());
}
```

Eventos load/unload

- Eventos manejados muy frecuentemente en asociación con el elemento body
- Se disparan cuando el usuario carga completamente la página (incluyendo estilos, scripts, etc) o sale de ella, respectivamente.
  - Por ejemplo, se puede asociar un manejador al evento load para comprobar la versión del navegador del visitante, y de esta forma cargar una versión de la página optimizada para esa versión
- También se pueden usar para gestionar las cookies

Evento onload - Ejemplo

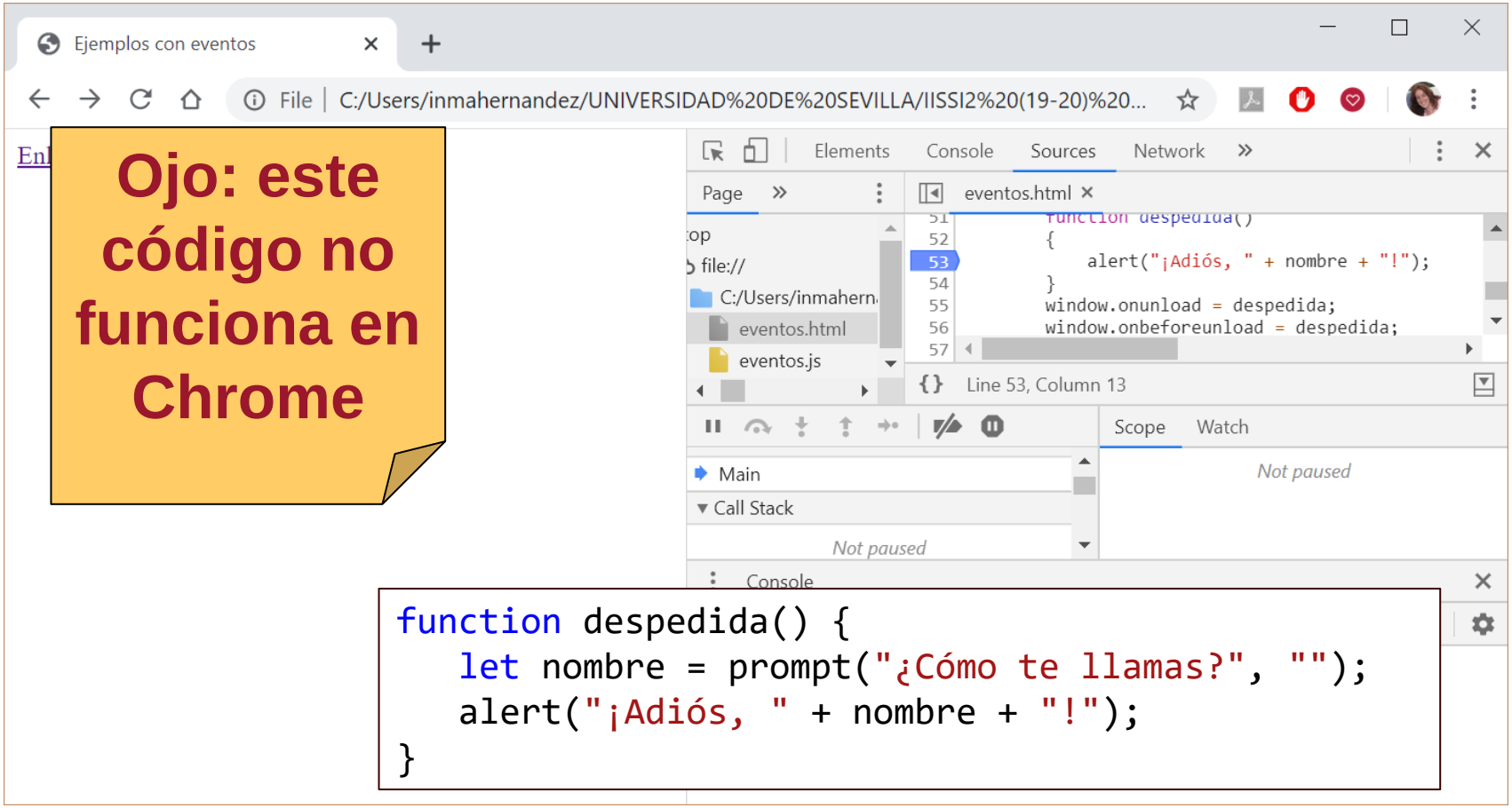
```
<body onLoad="checkBrowser()">
 <div id="browserInfo"></div>
```




```
function checkBrowser() {
    let txt = "<p>Browser CodeName: "
        + navigator.appCodeName + "</p>";
    txt+= "<p>Browser Name: "
        + navigator.appName + "</p>";
    txt+= "<p>Browser Version: "
        + navigator.appVersion + "</p>";
    txt+= "<p>Cookies Enabled: "
        + navigator.cookieEnabled + "</p>";
    txt+= "<p>Platform: "
        + navigator.platform + "</p>";
    txt+= "<p>User-agent header: "
        + navigator.userAgent + "</p>";
    document.getElementById("browserInfo").innerHTML = txt;
}
```

Evento unload - Ejemplo

```
<body onUnload="despedida()">
```



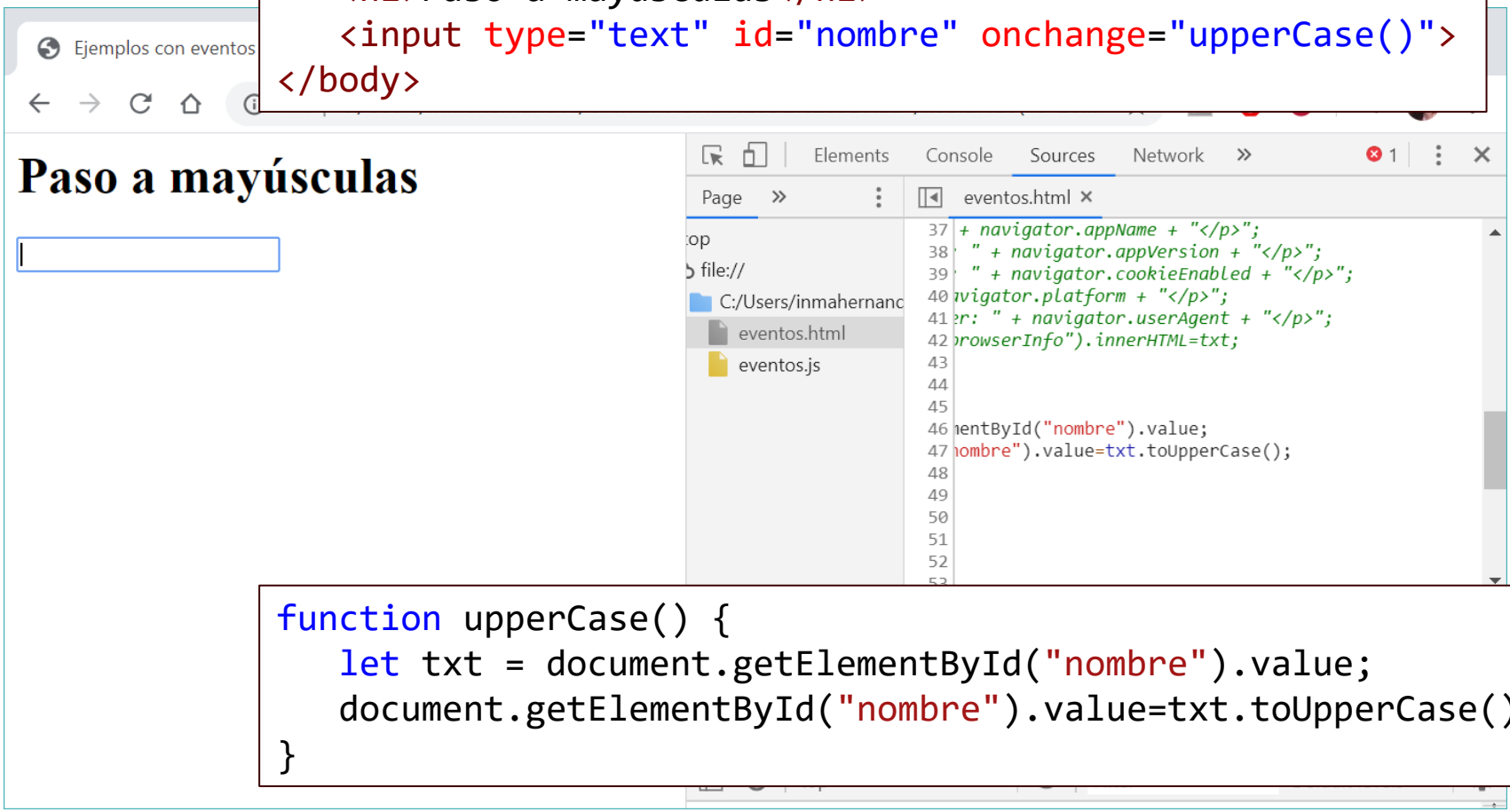

```
function despedida() {
    let nombre = prompt("¿Cómo te llamas?", "");
    alert("¡Adiós, " + nombre + "!");
}
```

Evento change

- Evento que se suele manejar frecuentemente asociado a los campos de los formularios (sobre todo para la validación de los datos introducidos)

Evento change - Ejemplo

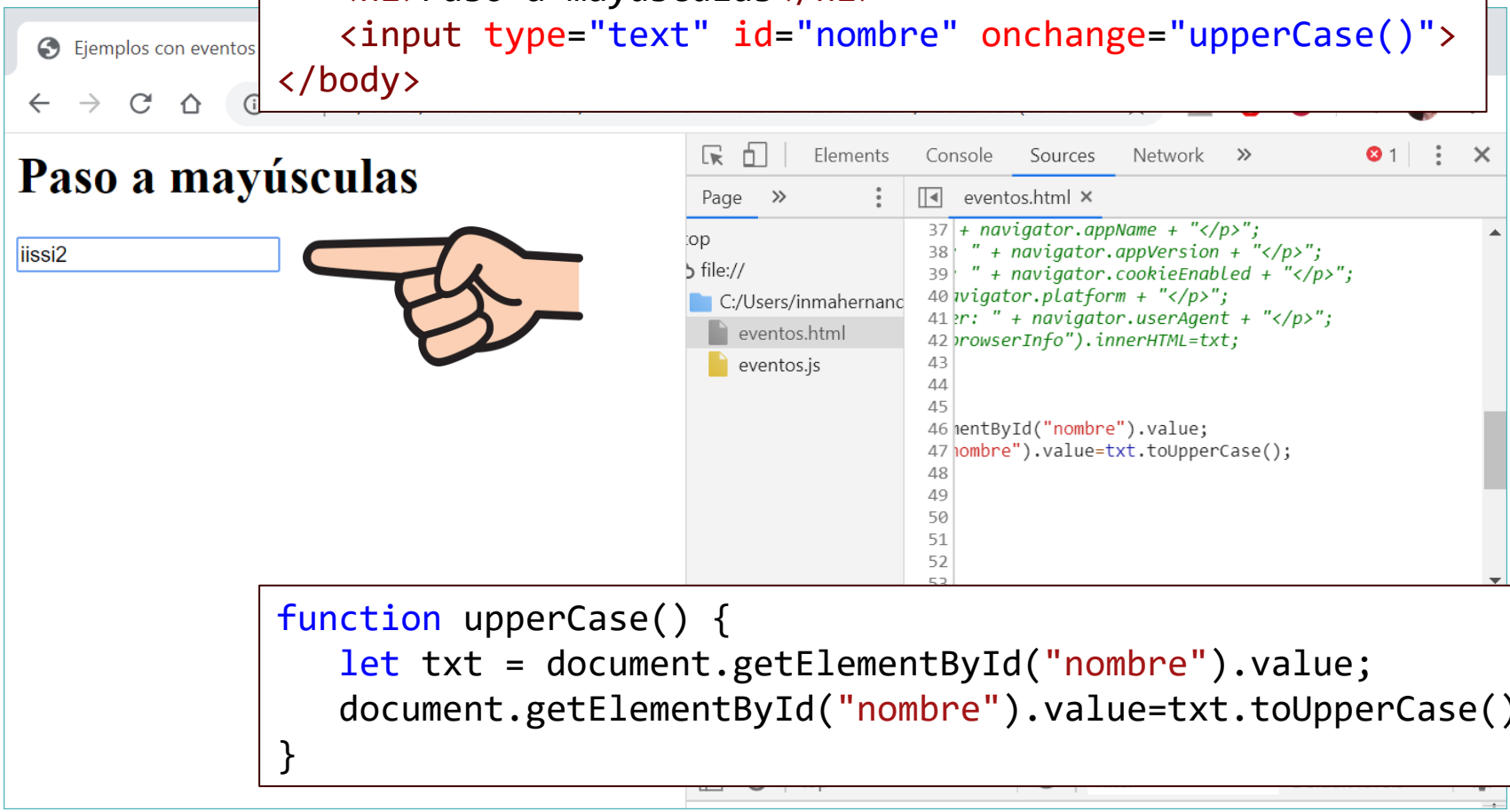
```
<body>
    <h1>Paso a mayúsculas</h1>
    <input type="text" id="nombre" onchange="upperCase()">
</body>
```




```
function upperCase() {
    let txt = document.getElementById("nombre").value;
    document.getElementById("nombre").value=txt.toUpperCase();
}
```

Evento change - Ejemplo

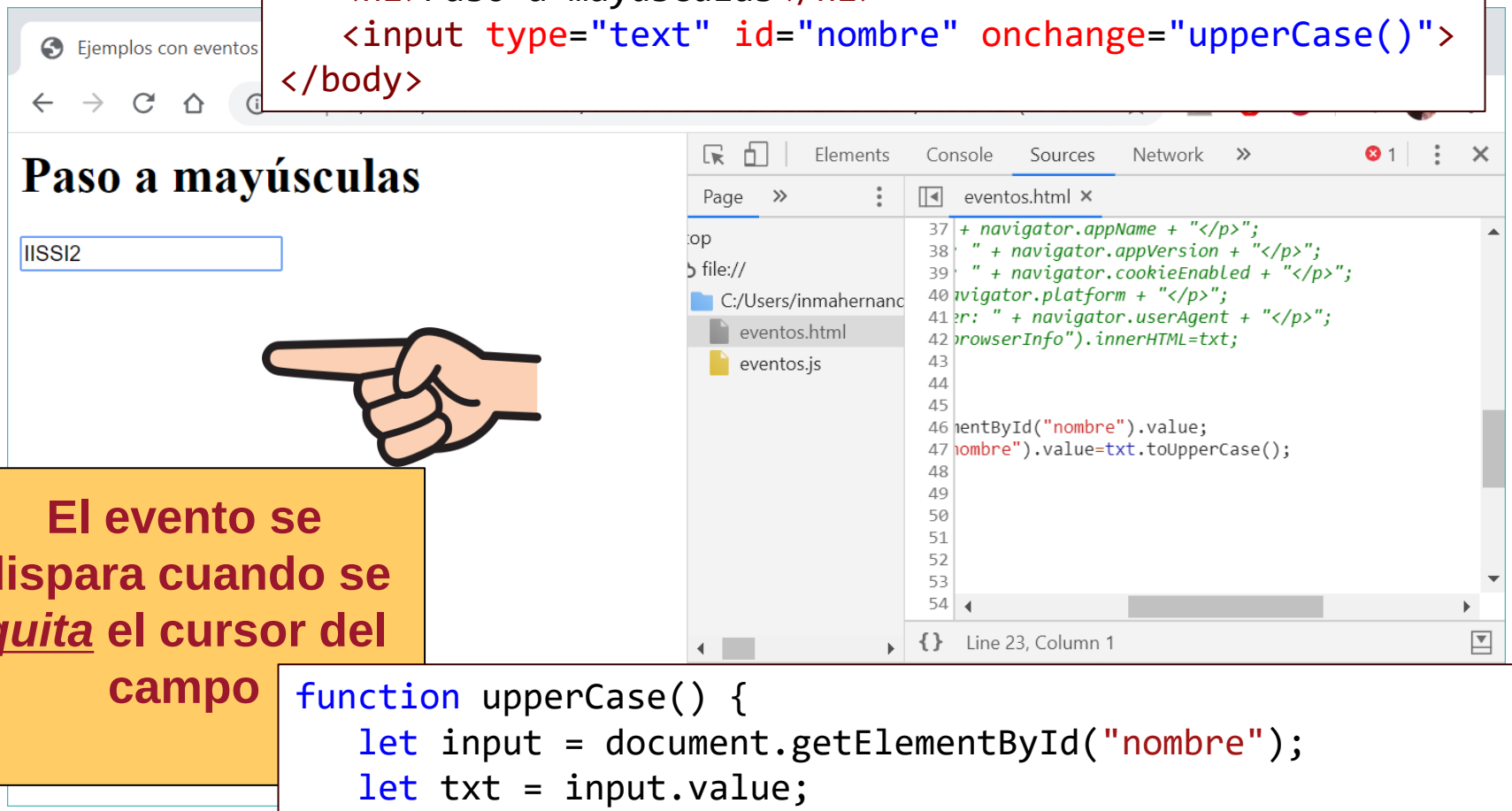
```
<body>
    <h1>Paso a mayúsculas</h1>
    <input type="text" id="nombre" onchange="upperCase()">
</body>
```




```
function upperCase() {
    let txt = document.getElementById("nombre").value;
    document.getElementById("nombre").value=txt.toUpperCase();
}
```

Evento change - Ejemplo

```
<body>
    <h1>Paso a mayúsculas</h1>
    <input type="text" id="nombre" onchange="upperCase()">
</body>
```



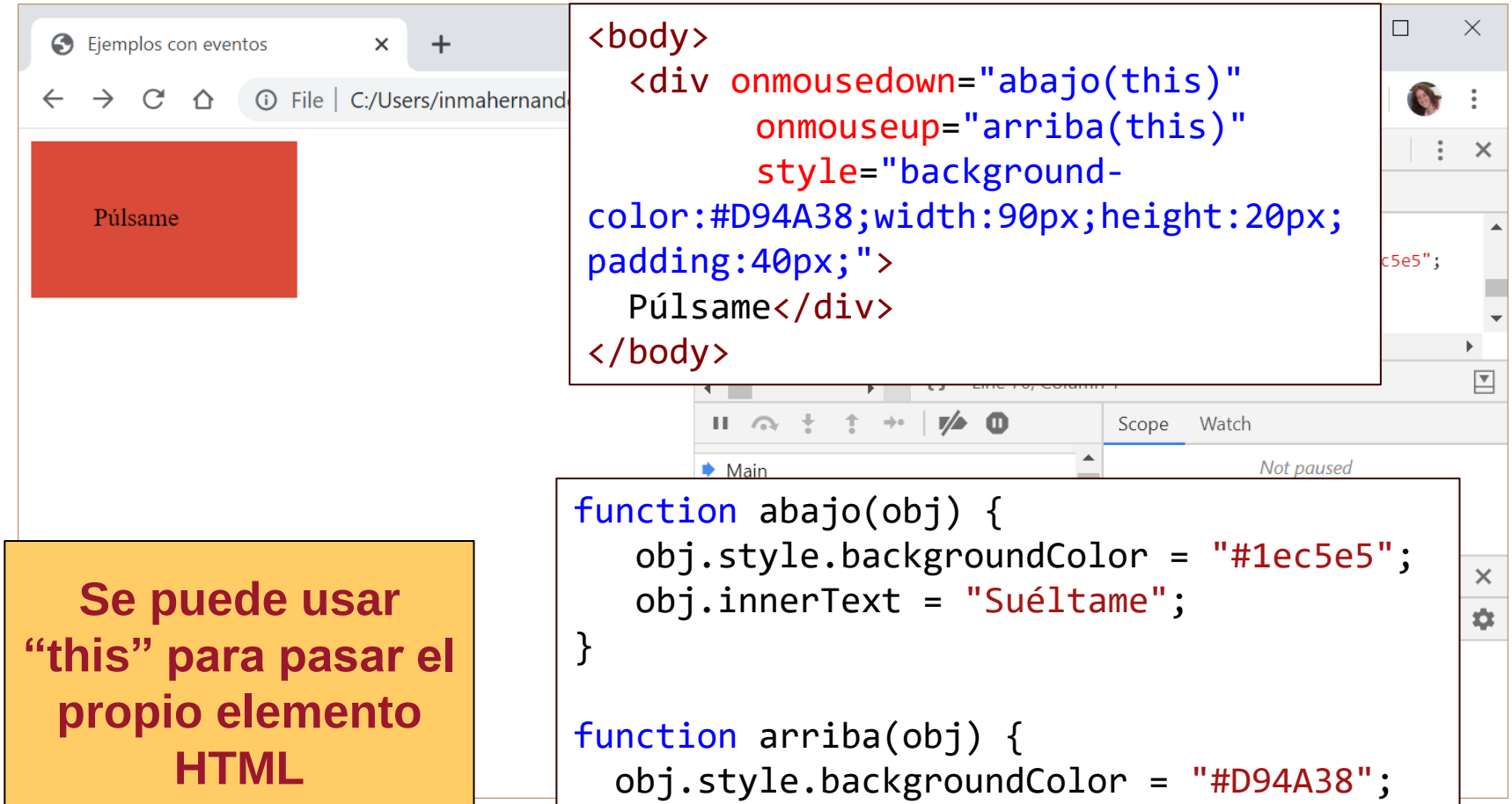
El evento se dispara cuando se *quita* el cursor del campo

```
function upperCase() {
    let input = document.getElementById("nombre");
    let txt = input.value;
    input.value = txt.toUpperCase();
}
```

Eventos de ratón

- Se disparan cuando el usuario interactúa con el ratón con un elemento
  - mouseover: cuando el ratón está pasando por encima del elemento
  - mouseout: cuando el ratón deja de pasar por encima del elemento
  - mousedown: Cuando se pulsa el botón del ratón sobre el elemento
  - mouseup: Cuando se libera el botón del ratón tras pulsar sobre un elemento
  - click: cuando se completa la pulsación (mousedown + mouseup)

Eventos de ratón - Ejemplo

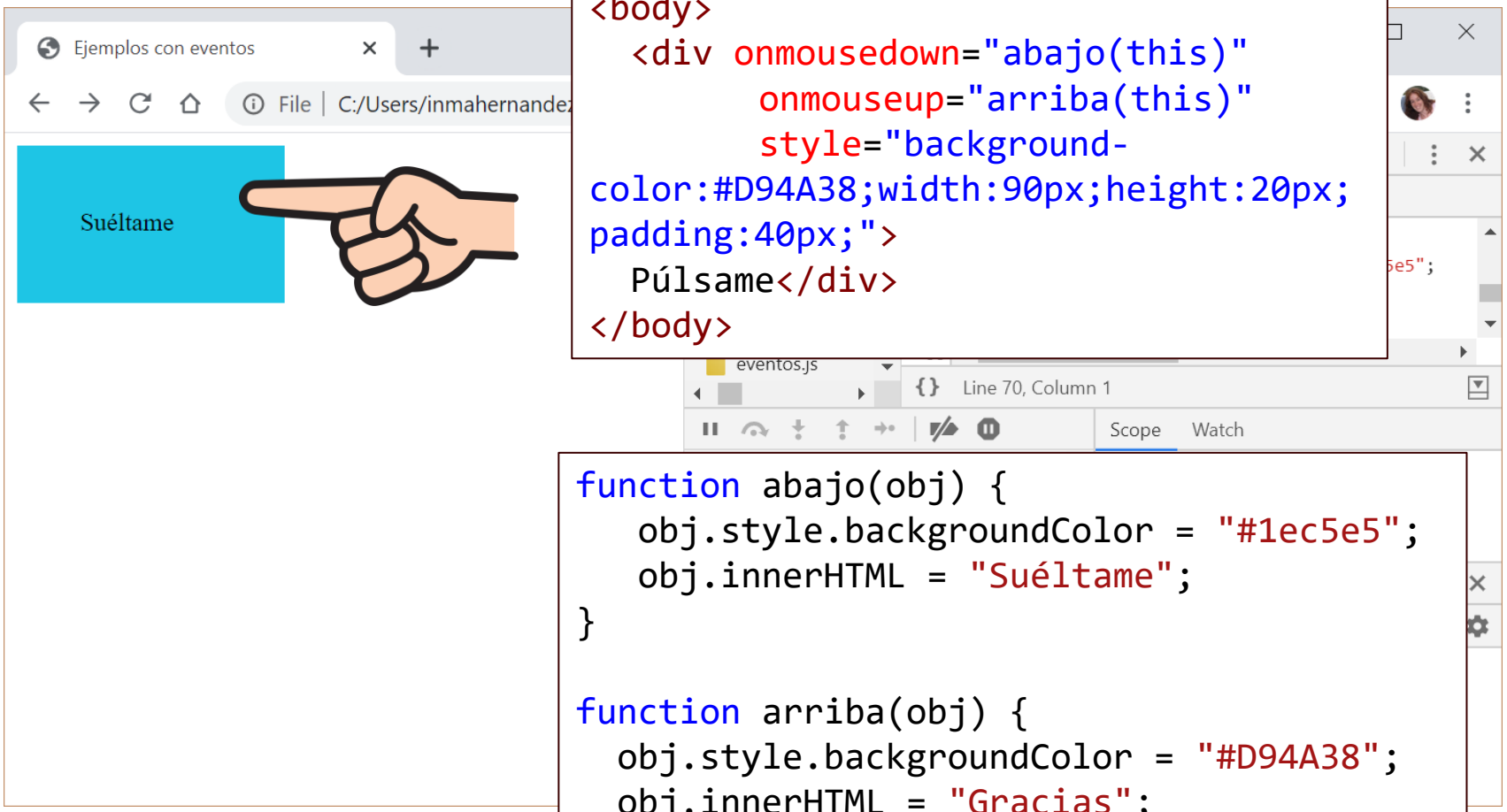



```
<body>
  <div onmousedown="abajo(this)"
       onmouseup="arriba(this)"
       style="background-
color:#D94A38;width:90px;height:20px;
padding:40px;">
  Púlsame</div>
</body>
```

```
function abajo(obj) {
   obj.style.backgroundColor = "#1ec5e5";
   obj.innerText = "Suéltame";
}

function arriba(obj) {
  obj.style.backgroundColor = "#D94A38";
  obj.innerText = "Gracias";
}
```

Se puede usar "this" para pasar el propio elemento HTML

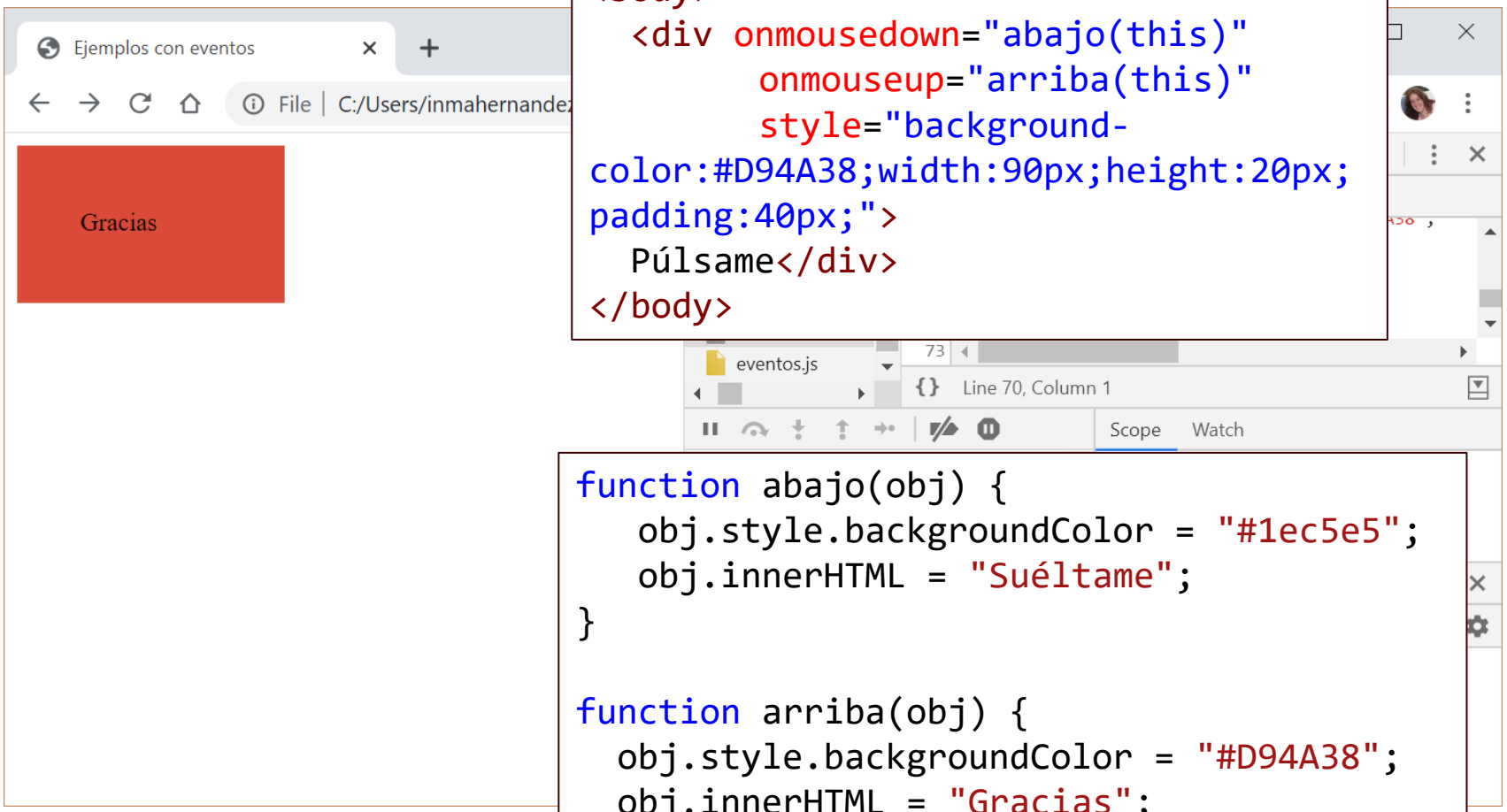
Eventos de ratón - Ejemplo




```
<body>
  <div onmousedown="abajo(this)"
       onmouseup="arriba(this)"
       style="background-
color:#D94A38;width:90px;height:20px;
padding:40px;">
  Púlsame</div>
</body>
```

```
function abajo(obj) {
   obj.style.backgroundColor = "#1ec5e5";
   obj.innerHTML = "Suéltame";
}

function arriba(obj) {
  obj.style.backgroundColor = "#D94A38";
  obj.innerHTML = "Gracias";
}
```

Eventos de ratón - Ejemplo



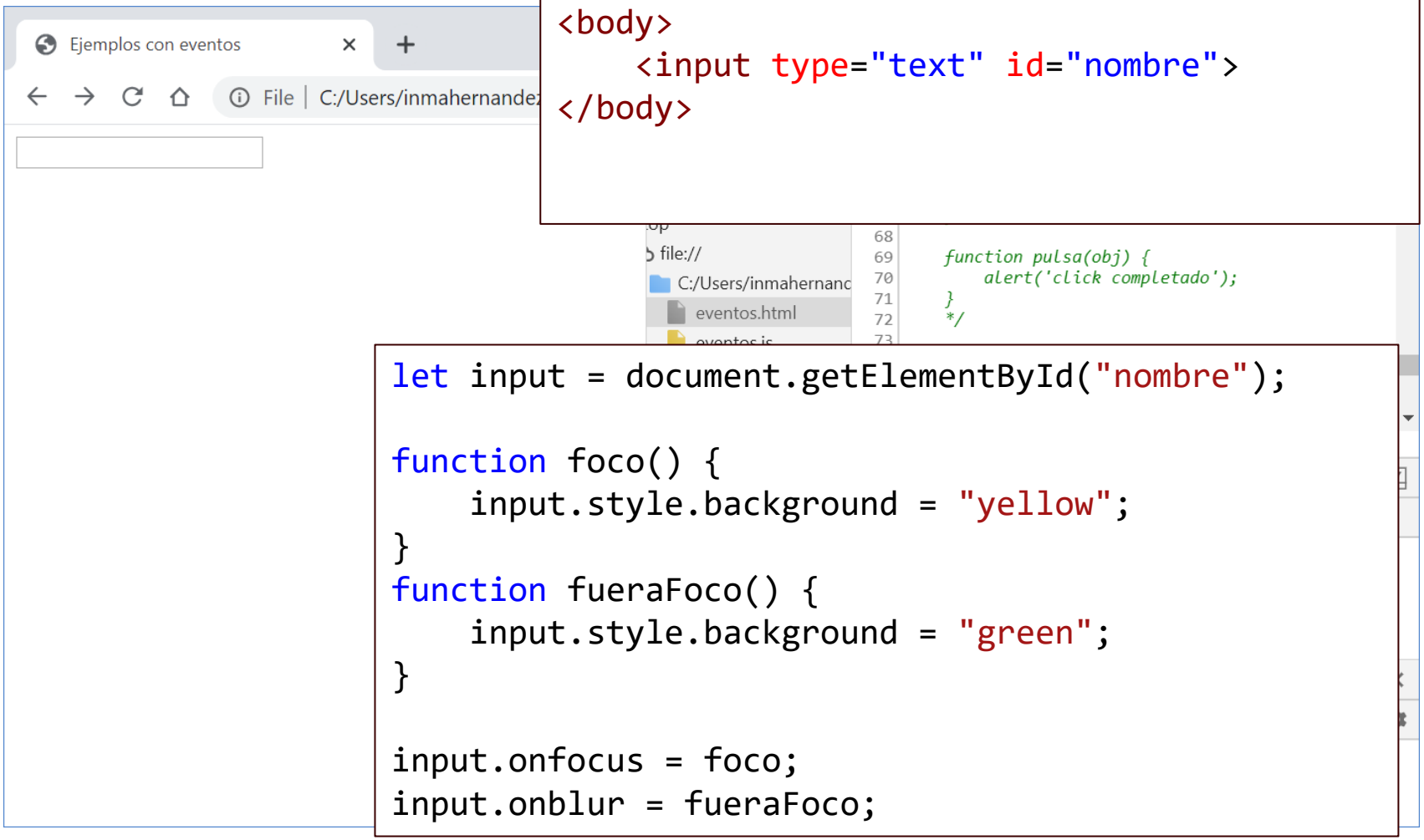

```
<body>
  <div onmousedown="abajo(this)"
       onmouseup="arriba(this)"
       style="background-
color:#D94A38;width:90px;height:20px;
padding:40px;">
  Púlsame</div>
</body>
```

```
function abajo(obj) {
   obj.style.backgroundColor = "#1ec5e5";
   obj.innerHTML = "Suéltame";
}

function arriba(obj) {
  obj.style.backgroundColor = "#D94A38";
  obj.innerHTML = "Gracias";
}
```

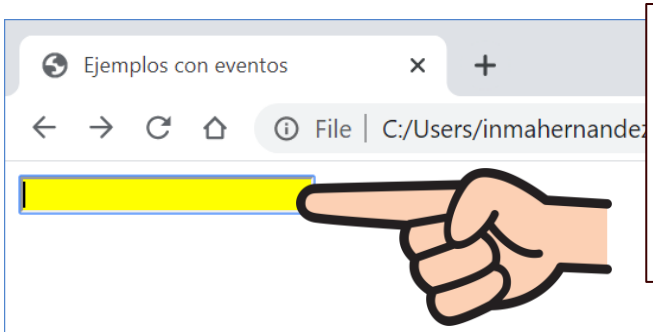
Evento focus/blur

- Se producen cuando el elemento adquiere/pierde el foco
- Se suelen emplear para aplicar algún tipo de procesado sobre los campos de los formularios conforme el usuario los va completando

Eventos focus/blur - Ejemplos



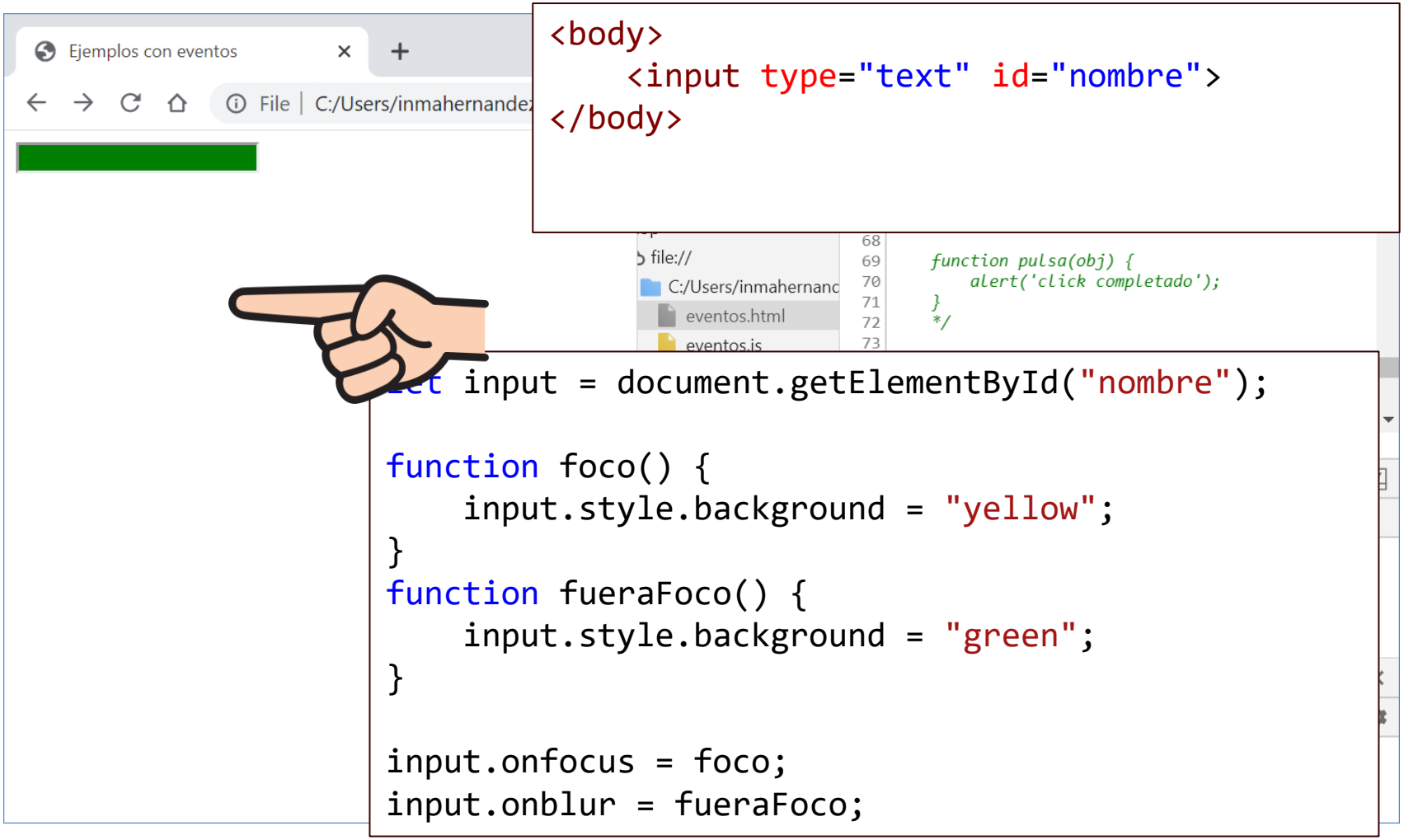
Eventos focus/blur - Ejemplos




```
<body>
    <input type="text" id="nombre">
</body>
```

```
let input = document.getElementById("nombre");

function foco() {
    input.style.background = "yellow";
}
function fueraFoco() {
    input.style.background = "green";
}

input.onfocus = foco;
input.onblur = fueraFoco;
```

Eventos focus/blur - Ejemplos



Ejemplo: lista de la compra

```
<body>
    Nuevo elemento: <input type="text" id="input-list">
    <br>
    <button id="btn-clear">Vaciar la lista</button>
    <h1>Lista de la compra</h1>
    <ul id="list"></ul>
</body>
```

Ejemplo: lista de la compra

```
<body>
    Nuevo elemento: <input type="text" id="input-list">
    <br>
    <button id="btn-clear">Vaciar la lista</button>
    <h1>Lista de la compra</h1>
    <ul id="list"></ul>
</body>
```


```
let input = document.getElementById("input-list");
let list = document.getElementById("list");
let clearBtn = document.getElementById("btn-clear");
```

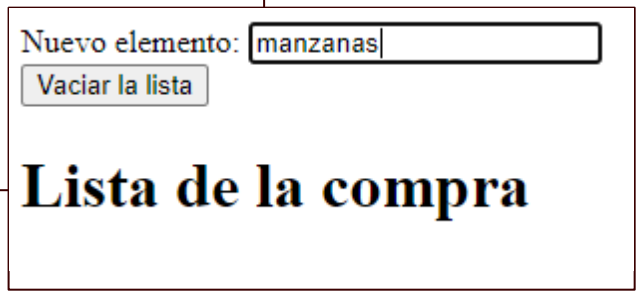
Ejemplo: lista de la compra

```
<body>
    Nuevo elemento: <input type="text" id="input-list">
    <br>
    <button id="btn-clear">Vaciar la lista</button>
    <h1>Lista de la compra</h1>
    <ul id="list"></ul>
</body>
```


```
let input = document.getElementById("input-list");
let list = document.getElementById("list");
let clearBtn = document.getElementById("btn-clear");

function addElement() {
    let element = input.value;
    input.value = "";

    let newItem = document.createElement("li");
    newItem.innerText = element;
    list.appendChild(newItem);
}

input.addEventListener("change", addElement);
```

Ejemplo: lista de la compra

```
<body>
    Nuevo elemento: <input type="text" id="input-list">
    <br>
    <button id="btn-clear">Vaciar la lista</button>
    <h1>Lista de la compra</h1>
    <ul id="list"></ul>
</body>
```




```
let input = document.getElementById("input-list");
let list = document.getElementById("list");
let clearBtn = document.getElementById("btn-clear");

function addElement() {
    let element = input.value;
    input.value = "";

    let newItem = document.createElement("li");
    newItem.innerText = element;
    list.appendChild(newItem);
}

input.addEventListener("change", addElement);
```

Ejemplo: lista de la compra

```
<body>
    Nuevo elemento: <input type="text" id="input-list">
    <br>
    <button id="btn-clear">Vaciar la lista</button>
    <h1>Lista de la compra</h1>
    <ul id="list"></ul>
</body>
```


```
let input = document.getElementById("input-list");
let list = document.getElementById("list");
let clearBtn = document.getElementById("btn-clear");

function addElement() {
    let element = input.value;
    input.value = "";

    let newItem = document.createElement("li");
    newItem.innerText = element;
    list.appendChild(newItem);
}

input.addEventListener("change", addElement);
```

Ejemplo: lista de la compra

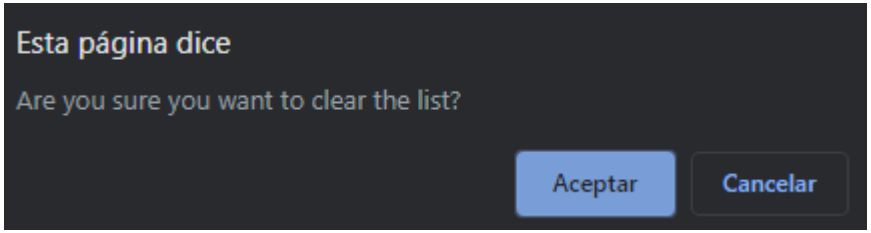
```
<body>
    Nuevo elemento: <input type="text" id="input-list">
    <br>
    <button id="btn-clear">Vaciar la lista</button>
    <h1>Lista de la compra</h1>
    <ul id="list"></ul>
</body>
```


```
let input = document.getElementById("input-list");
let list = document.getElementById("list");
let clearBtn = document.getElementById("btn-clear");

function clearList() {
    let msg = "Are you sure you want to
               clear the list?";
    if(confirm(msg)) {
        list.innerHTML = "";
    }
}

clearBtn.addEventListener("click", clearList);
```

Ejemplo: lista de la compra

```
<body>
    Nuevo elemento: <input type="text" id="input-list">
    <br>
    <button id="btn-clear">Vaciar la lista</button>
    <h1>Lista de la compra</h1>
    <ul id="list"></ul>
</body>
```


```
let input = document.getElementById("input-list");
let list = document.getElementById("list");
let clearBtn = document.getElementById("btn-clear");

function clearList() {
    let msg = "Are you sure you want to
               clear the list?";
    if(confirm(msg)) {
        list.innerHTML = "";
    }
}

clearBtn.addEventListener("click", clearList);
```



Otros eventos interesantes

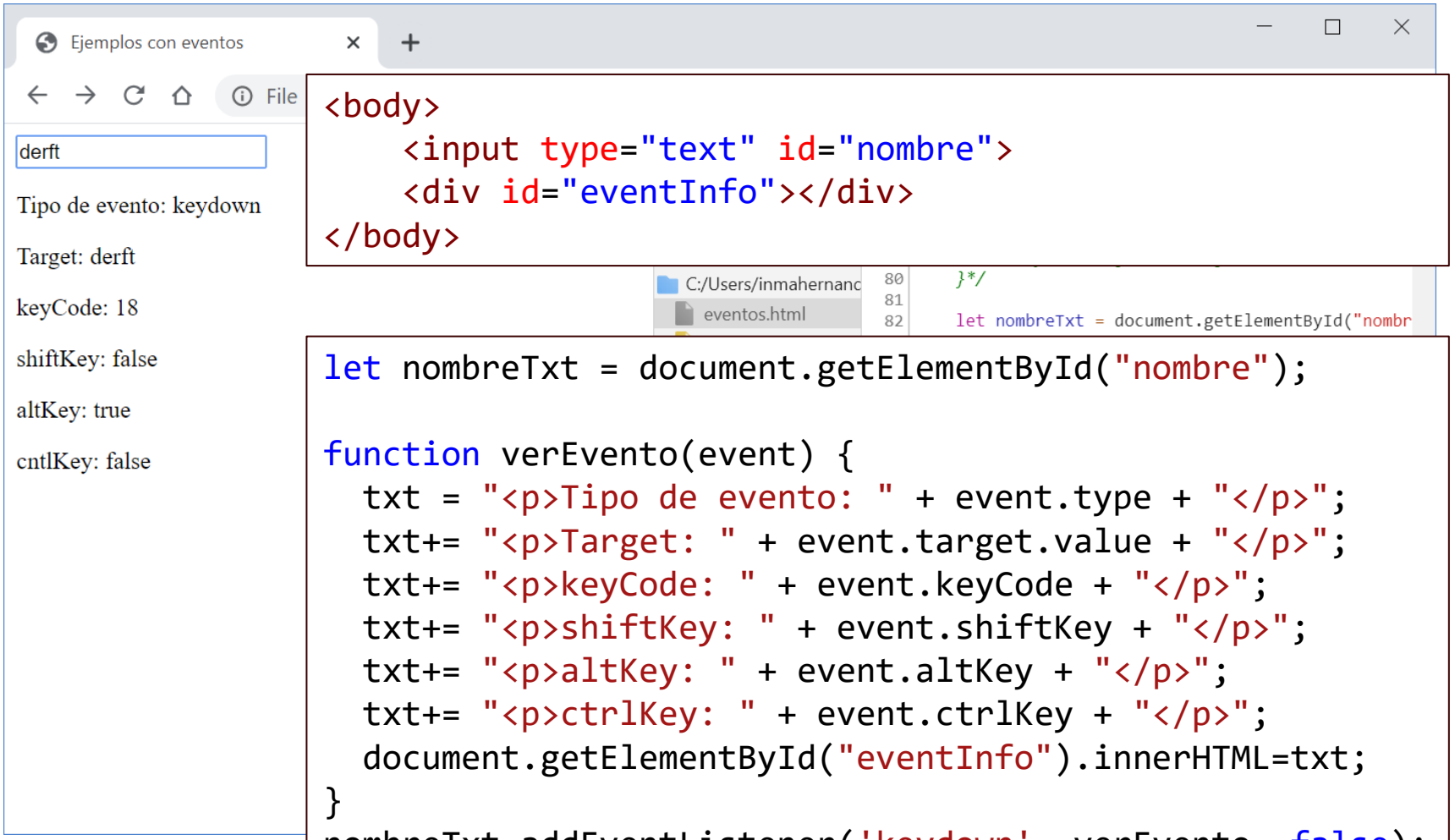
- submit (objeto Form)
- scroll (objeto Document)
- keydown/keypress/keyup

Contenidos

El objeto Event

- Cuando el listener notifica que se ha producido un evento y se ejecuta el código de una función para manejar dicho evento, se pasa a dicha función como parámetro un objeto que representa el evento en sí
- Este objeto tiene ciertas propiedades que pueden ser consultadas desde la función
- Todo objeto evento hereda de la clase Event, pero las propiedades concretas dependen del subtipo de evento de que se trate
  - Todos tienen la propiedad "target", que devuelve una referencia al elemento HTML que provoca el evento

Objeto Event - Ejemplo




```
<body>
    <input type="text" id="nombre">
    <div id="eventInfo"></div>
</body>
```

C:/Users/inmahernanc
eventos.html

```
80      }*/
81
82      let nombreTxt = document.getElementById("nombr
```

```
let nombreTxt = document.getElementById("nombre");

function verEvento(event) {
  txt = "<p>Tipo de evento: " + event.type + "</p>";
  txt+= "<p>Target: " + event.target.value + "</p>";
  txt+= "<p>keyCode: " + event.keyCode + "</p>";
  txt+= "<p>shiftKey: " + event.shiftKey + "</p>";
  txt+= "<p>altKey: " + event.altKey + "</p>";
  txt+= "<p>ctrlKey: " + event.ctrlKey + "</p>";
  document.getElementById("eventInfo").innerHTML=txt;
}
nombreTxt.addEventListener('keydown', verEvento, false);
```

Propiedades de los eventos de teclado

Property/Method	Description
altKey	Returns whether the "ALT" key was pressed when the key event was triggered
charCode	Returns the Unicode character code of the key that triggered the event
code	Returns the code of the key that triggered the event
ctrlKey	Returns whether the "CTRL" key was pressed when the key event was triggered
getModifierState()	Returns true if the specified key is activated
isComposing	Returns whether the state of the event is composing or not
key	Returns the key value of the key represented by the event
keyCode	Returns the Unicode character code of the key that triggered the onkeypress event, or the Unicode key code of the key that triggered the onkeydown or onkeyup event
location	Returns the location of a key on the keyboard or device
metaKey	Returns whether the "meta" key was pressed when the key event was triggered
repeat	Returns whether a key is being hold down repeatedly, or not
shiftKey	Returns whether the "SHIFT" key was pressed when the key event was triggered
which	Returns the Unicode character code of the key that triggered the onkeypress event, or the Unicode key code of the key that triggered the onkeydown or onkeyup event

https://www.w3schools.com/jsref/obj_keyboardevent.asp

Propiedades de los eventos de ratón

Property/Method	Description
altKey	Returns whether the "ALT" key was pressed when the mouse event was triggered
button	Returns which mouse button was pressed when the mouse event was triggered
buttons	Returns which mouse buttons were pressed when the mouse event was triggered
clientX	Returns the horizontal coordinate of the mouse pointer, relative to the current window, when the mouse event was triggered
clientY	Returns the vertical coordinate of the mouse pointer, relative to the current window, when the mouse event was triggered
ctrlKey	Returns whether the "CTRL" key was pressed when the mouse event was triggered
getModifierState()	Returns true if the specified key is activated
metaKey	Returns whether the "META" key was pressed when an event was triggered
movementX	Returns the horizontal coordinate of the mouse pointer relative to the position of the last mousemove event
movementY	Returns the vertical coordinate of the mouse pointer relative to the position of the last mousemove event
offsetX	Returns the horizontal coordinate of the mouse pointer relative to the position of the edge of the target element
offsetY	Returns the vertical coordinate of the mouse pointer relative to the position of the edge of the target element
pageX	Returns the horizontal coordinate of the mouse pointer, relative to the document, when the mouse event was triggered
pageY	Returns the vertical coordinate of the mouse pointer, relative to the document, when the mouse event was triggered
region	
relatedTarget	Returns the element related to the element that triggered the mouse event
screenX	Returns the horizontal coordinate of the mouse pointer, relative to the screen, when an event was triggered
screenY	Returns the vertical coordinate of the mouse pointer, relative to the screen, when an event was triggered
shiftKey	Returns whether the "SHIFT" key was pressed when an event was triggered
which	Returns which mouse button was pressed when the mouse event was triggered

https://www.w3schools.com/jsref/obj_mouseevent.asp

Propiedades de los eventos de campos de formulario

Property/Method	Description
data	Returns the inserted characters
dataTransfer	Returns an object containing information about the inserted/deleted data
getTargetRanges()	Returns an array containing target ranges that will be affected by the insertion/deletion
inputType	Returns the type of the change (i.e "inserting" or "deleting")
isComposing	Returns whether the state of the event is composing or not

https://www.w3schools.com/jsref/obj_inputevent.asp

Propiedades de los eventos táctiles

Property/Method	Description
altKey	Returns whether the "ALT" key was pressed when the touch event was triggered
changedTouches	Returns a list of all the touch objects whose state changed between the previous touch and this touch
ctrlKey	Returns whether the "CTRL" key was pressed when the touch event was triggered
metaKey	Returns whether the "meta" key was pressed when the touch event was triggered
shiftKey	Returns whether the "SHIFT" key was pressed when the touch event was triggered
targetTouches	Returns a list of all the touch objects that are in contact with the surface and where the touchstart event occured on the same target element as the current target element
touches	Returns a list of all the touch objects that are currently in contact with the surface

https://www.w3schools.com/jsref/obj_touchevent.asp

Contenidos

1. Introducción
2. Manejo de eventos en JavaScript
3. Ejemplos
4. El objeto Event
5. **Validación de formularios**

¿Por qué validar formularios en cliente?

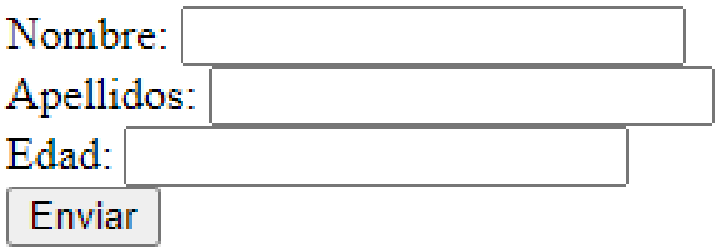
- Permite mostrar al usuario mensajes de error intuitivos antes de enviar el formulario al backend
- Se puede cancelar el envío del formulario para ahorrar al backend procesar formularios inválidos
- Podemos decidir enviar el formulario de cualquier otra manera en lugar de la que ofrece por defecto el navegador (por ejemplo, mediante una petición asíncrona AJAX)
- En cualquier caso, siempre es necesario validar también en backend

Evento "submit"

- Los elementos <form> emiten un evento "submit" cuando el usuario envía el formulario:

```
<form id="user-form">
    Nombre: <input id="user-firstName" name="firstName" type="text">
    Apellidos: <input id="user-lastName" name="lastName" type="text">
    Edad: <input id="user-age" name="age" type="number" min="18">
    <input type="submit">
</form>
```

```
let form = document.getElementById("user-form");
form.onsubmit = function() {
    alert("Formulario enviado");
}
```

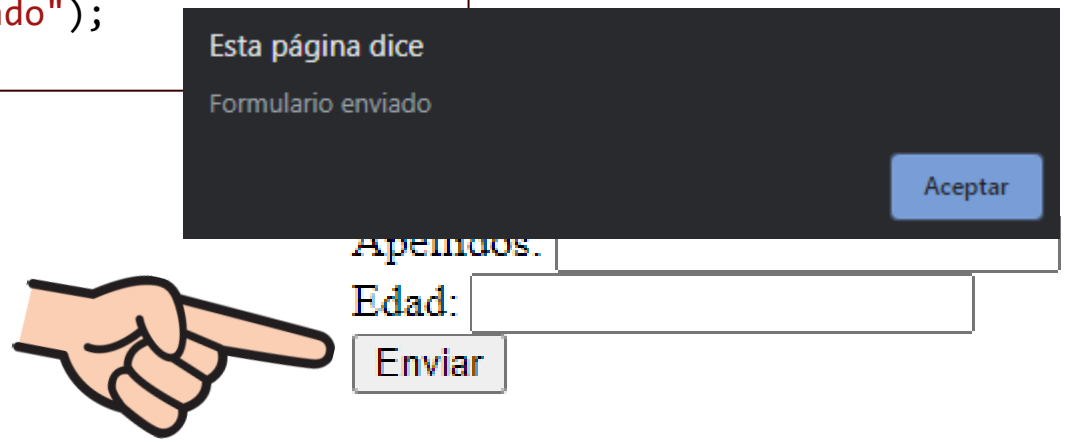


Evento "submit"

- Los elementos <form> emiten un evento "submit" cuando el usuario envía el formulario:

```
<form id="user-form">
    Nombre: <input id="user-firstName" name="firstName" type="text">
    Apellidos: <input id="user-lastName" name="lastName" type="text">
    Edad: <input id="user-age" name="age" type="number" min="18">
    <input type="submit">
</form>
```

```
let form = document.getElementById("user-form");
form.onsubmit = function() {
    alert("Formulario enviado");
}
```



Evento "submit"

- Si el manejador del evento devuelve **false**, el envío del formulario se cancela

```
<form id="user-form">
    Nombre: <input id="user-firstName" name="firstName" type="text">
    Apellidos: <input id="user-lastName" name="lastName" type="text">
    Edad: <input id="user-age" name="age" type="number" min="18">
    <input type="submit">
</form>
```

```
function validateForm() {
    let valid = true;
    let name = document.getElementById("user-firstName").value;

    if(name.length < 3) {
        alert("The name must be at least 3 characters long!");
        valid = false;
    }

    return valid;
 }

form.onsubmit = validateForm;
```

Podemos acceder al valor introducido en cada campo con el atributo *value*

Objetos FormData

- Los formularios se pueden envolver en objetos FormData para facilitar su consulta:

```
<form id="user-form">
    Nombre: <input id="user-firstName" name="firstName" type="text">
    Apellidos: <input id="user-lastName" name="lastName" type="text">
    Edad: <input id="user-age" name="age" type="number" min="18">
    <input type="submit">
</form>
```

```
let form = document.getElementById("user-form");

function validateForm() {
    let formData = new FormData(form);

    if(formData.get("firstName").length < 3) {
        alert("The name must be at least 3 characters long!");
        res = false;
    }

    return res;
}

form.onsubmit = validateForm;
```

Creamos un nuevo FormData a partir de la referencia al formulario

Los campos se referencian por su *name*, no por su *id*

Métodos FormData

- **`formData.append("name", "value")`**: Añade un nuevo par clave/valor a los datos del formulario
- **`formData.set("name", "value")`**: Cambia el valor asociado a una clave, y la añade si ésta no existe
- **`formData.delete("name")`**: Elimina una clave (y su valor) de los datos del formulario
- **`formData.get("name")`**: Obtiene el valor asociado a una clave
- **`formData.has("name")`**: Devuelve un boolean indicando si el formulario tiene asociada una clave indicada
- **`formData.keys()`**: Devuelve un iterador sobre todas las claves (atributos *name*) asociadas al formulario
- **`formData.values()`**: Devuelve un iterador sobre todos los valores del formulario
- **`formData.entries()`**: Devuelve un iterador de todos los pares clave/valor

Ventajas y desventajas FormData

- Ventajas de usar FormData sobre acceder manualmente a los campos:
  - Permite añadir, eliminar y modificar más fácilmente los valores que se enviarán al servidor
  - Sólo es necesaria una búsqueda en el DOM para obtener el formulario, al contrario que tener que realizar una por cada <input>
  - Se puede enviar directamente el objeto FormData a través de AJAX (lo veremos más adelante)
  - Ofrece una interfaz de programación estándar para acceder y manipular los datos de un formulario: permite el uso de módulos validadores
- Desventajas:
  - Los valores del objeto FormData son copias de los que hay en los <input>, si se desea modificar el formulario que ve el usuario hay que acceder a ellos y cambiarlos

Módulos validadores

- Si se envuelve el formulario en un FormData se le puede pasar a otro módulo JS para que lo valide:
  - Simplifica el código asociado a la gestión de eventos de la página
  - Permite la combinación mediante composición de validadores

Módulos validadores

- Si se envuelve el formulario en un FormData se le puede pasar a otro módulo JS para que lo valide:

validators/user.js

```
const userValidator = {
    validateUser: function(formData) {
        let errors = [];

        if(formData.get("firstName").length < 3) {
            errors.append("The first name is too short");
        }

        if(formData.get("lastName").length < 3) {
            errors.append("The last name is too short");
        }

        if(formData.get("age") < 18) {
            errors.append("You need to be 18+ years old");
        }

        return errors;
    }
};
```

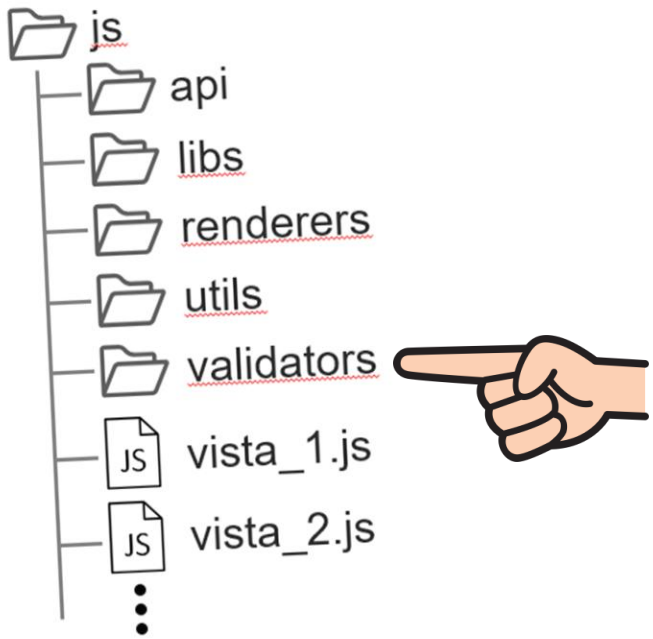
Manejador de submit del form:

```
import { userValidator } from 'validators/user.js';
let form = document.getElementById("user-form");
let formData = new FormData(form);

let errors = userValidator.validateUser(formData);
if(errors) {
    showErrors(errors);
} else {
    sendForm(formData);
}
```

Estructura del proyecto

Organización del código

- Nosotros seguiremos la siguiente estructura



Ejercicios

- Implementar una aplicación web que vaya informando de los sitios en los que hace click el ratón
- Modificar la aplicación anterior para añadir un contador de clicks
- Modificar la aplicación anterior para contar solamente los clicks que se produzcan en la parte inferior de la pantalla (de la mitad hacia abajo)

Referencias

- https://www.w3schools.com/jsref/dom_obj_event.asp
- https://www.w3schools.com/jsref/obj_events.asp
- https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events
- https://www.htmlgoodies.com/beyond/javascript/events-and-javascript-part-3-the-event-object.html

Uso de JavaScript (II): Eventos y formularios

Departamento de Lenguajes y Sistemas Informáticos

Universidad de Sevilla

Daniel Ayala, Carlos Arévalo, José Calderón, Margarita Cruz, Inma Hernández, David Ruiz