Eventos

Queremos crear aplicaciones interactivas que reaccionen a las distintas acciones que el usuario realiza sobre ellas, es decir, que reaccionen a los **eventos** que se producen.

Para la gestión de eventos tenemos distintos patrones de diseño:

- **Patrón observador:** tenemos un objeto (sujeto) y otros objetos del cual dependen (observadores). Hay una dependencia de uno a muchos entre objetos, de forma que cuando el sujeto cambia de estado lo notifica a todos los observadores normalmente a través de uno de sus métodos.
- **Patrón publicador-subscriptor:** variante del patrón observador. La diferencia es que el publicador no es el que programa las notificaciones, se realizan a través de un **event bus o broker o message broker** y el publicador y subscriptor/es no tienen consciencia el uno del otro.

En JavaScript, el sistema de eventos tiene cierta similitud. Los sujetos son los elementos de la página (botones, imágenes, etc) y los observadores, sólo son funciones que cumplen la misma función que la operación notificar.
En JavaScript y en otros lenguajes, estas funciones que se pasan a otros objetos se llaman **callbacks**. Siempre que veamos este término, tenemos que tener en cuenta que lo que se espera es una función que será ejecutada como otro punto del programa.

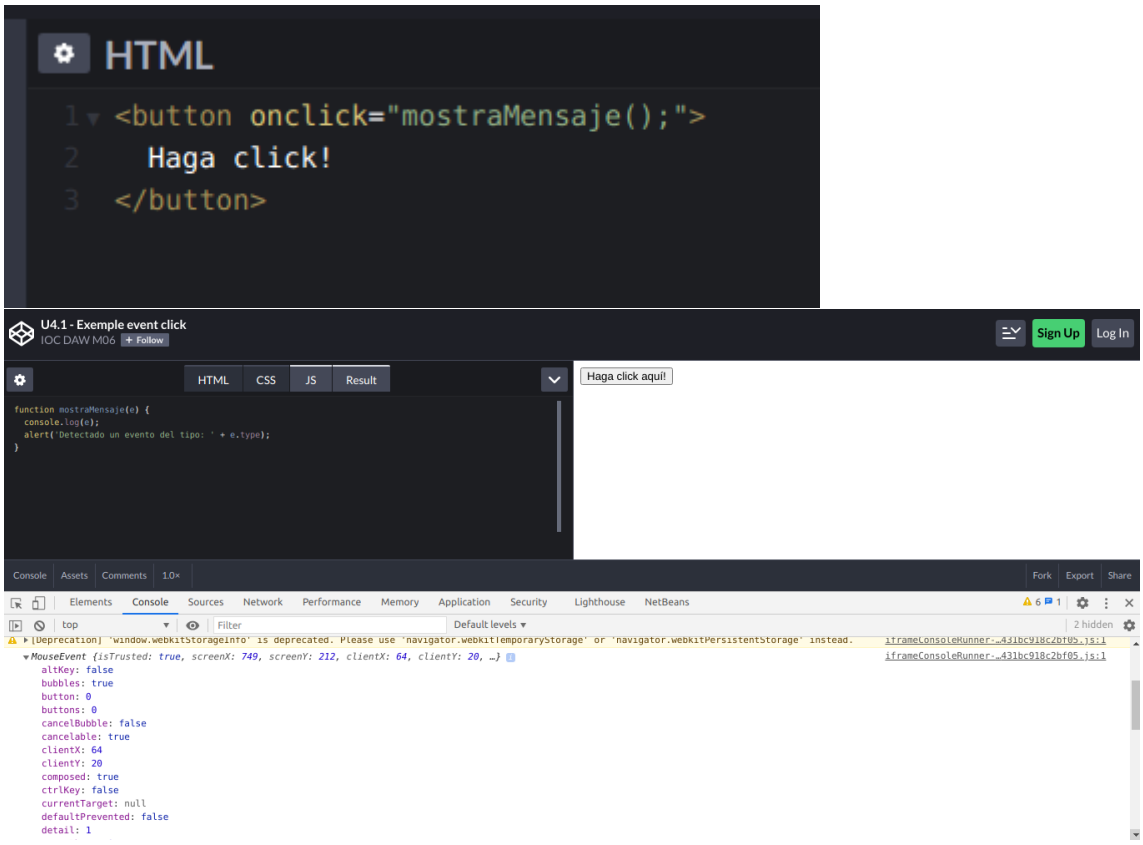
Actualmente hay una gran cantidad de eventos estándares asociados a elementos HTML.

https://developer.mozilla.org/es/docs/Web/Events

Los podríamos agrupar en:

- **DOM:** eventos asociados por acciones sobre el **Modelo de objetos del documento.**
  - **MouseEvent**
  - **KeyboardEvent**
  - **UIEvent:** ejm: cargar una página, cambiar el tamaño de una ventana.
  - …
- **HTML5**
- **HTML5 media:** audio y vídeo. Permiten conocer su estado: a la espera, buscando, finalizando.
- **WebAPI:**
  - **Web Workers:** con esta API se pueden programar tareas de forma concurrente.
  - **Web Storage**
  - **XMLHttpRequest:** API que nos permite utilizar llamadas AJAX desde el navegador.
  - **Web Sockets:** es posible mantener una conexión bidireccional con servidores, lo que permite crear aplicaciones multijugador (chat, juegos) en el navegador.

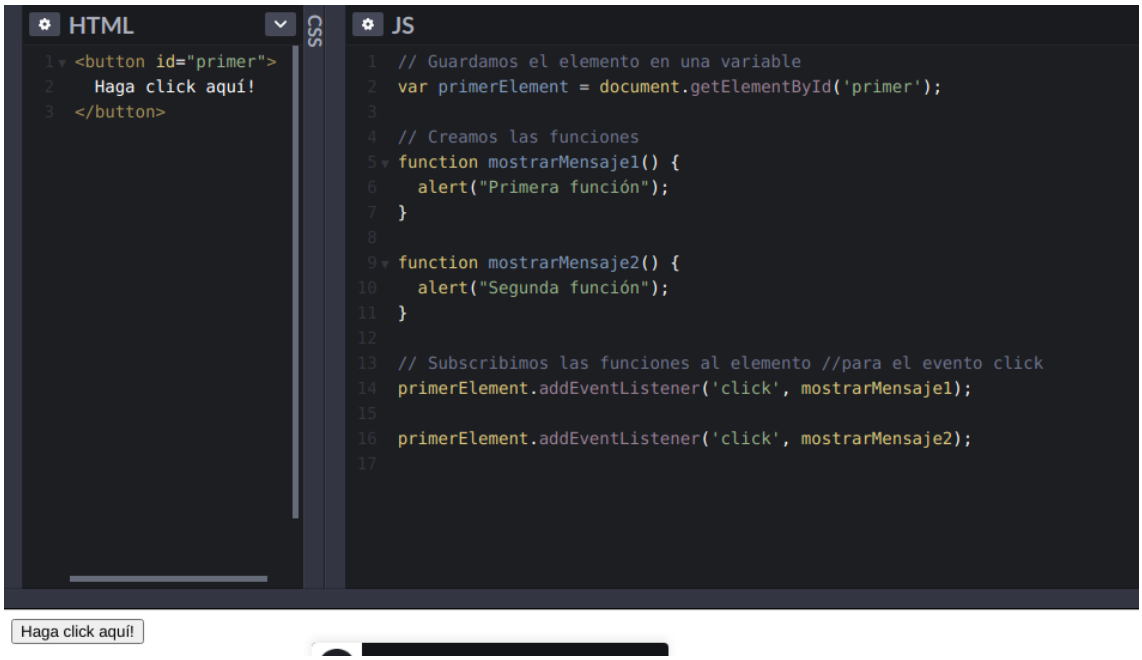
Gestión de eventos



Sin contaminar el código HTML:

Sólo se notifica la segunda, porque se sobreescribe la función que gestiona dicho evento.

Añadir múltiples observadores para un mismo sujeto:



ejemploEventos (repositorio)

Algunas cosas a tener en cuenta:

- En el aula virtual aparecerán todos los enlaces relacionados con los tipos de eventos más importantes.
- En cuanto a los eventos de teclado:

> Para poder disparar *eventos* de teclado, un elemento debe cumplir alguna de las següens condiciones:
>
> - Debe ser capaz de recibir el *foco* .
> - Debe contener un elemento hijo capaz de recibir el *foco* .
>
> Si se cumplen alguna de estas condiciones y el elemento, o alguno de sus descendientes, es el *foco* entonces disparará los *eventos* de teclado.

- Un caso interesante para los eventos de teclado es el elemento <body>. Aunque no tenga foco, si se añade un observador para los eventos de teclado, serán capturados igualmente.
- En algunos casos, nos puede interesar evitar que se dispare el evento que por defecto está asociado a una acción. Esto es posible utilizando el método **event.preventDefault().** Por ejemplo, esto nos permite detener el envío de un formulario al pulsar el botón enviar y, en su lugar, realizar una llamada asíncrona.

ejemploPreventDefault (repositorio)

- Algunos eventos interesantes con los conocimientos que tenemos actualmente:

Tabla: 1.7. Events relacionados con los formularios

tipo	operación	Descripción
select	onselect	Se dispara cuando se selecciona un texto
change	onChange	Se dispara cuando ha cambiado el valor del sujeto y éste ha perdido el foco
submit	onsubmit	Se dispara cuando se presiona un botón para enviar un formulario
reset	onreset	Se dispara cuando se presiona un botón de restablecer un formulario

Tabla: 1.8. Otros eventos destacables del DOM

tipo	operación	Descripción
foco	onfocus	Se dispara cuando un elemento recibe el *foco*
blur	onblur	Se dispara cuando un elemento pierde el *foco*
resize	onresize	Se dispara cuando se cambia el tamaño del elemento

Tabla: 1.9. Eventos de HTML5 generales

tipo	operación	Descripción
beforeunload	onbeforeunload	Se dispara antes de descargar la página y nos permite cancelar la operación

Tabla: 1.10. Eventos de HTML5 relacionados con los formularios

tipo	operación	Descripción
input	oninput	Se dispara cada vez que se produce un cambio en el elemento
invalid	oninvalid	Se dispara cuando se intenta enviar un formulario y el elemento no cumple con los requisitos que se han impuesto

Tabla: 1.11. Eventos de HTML5 media

tipo	operación	Descripción
canplay	oncanplay	Se dispara cuando es posible comenzar la reproducción del recurso de audio o vídeo
ended	onended	Se dispara cuando finaliza la reproducción del recurso
pausa	onpause	Se dispara cuando se pone en pausa la reproducción
play	onplay	Se dispara cuando se inicia la reproducción del recurso
waiting	onwaiting	Se dispara cuando se pausa la reproducción porque hay una falta temporal de datos (por ejemplo, si la reproducción es en *streaming* y todavía no se ha terminado de descargar el recurso entero)

Tabla: 1.13. Eventos del API Web Storage

tipo	operación	Descripción
storage	onstorage	Se dispara cuando hay cambios en el almacén de datos