Tema 4.2.
Modelo de objetos del documento. Event

- Gestión de Eventos
  - Manejadores como atributos de elementos HTML
  - Manejadores de eventos como funciones externas
  - Manejadores de eventos semánticos
- Método addEventListener
- Captura de eventos
- Flujo de eventos
- Supresión de eventos
- Cancelación de eventos
- Eliminación de eventos
- Generar eventos

Gestión de Eventos

- El modelo de eventos fue estandarizado por el W3C en DOM Level 2

- Un mismo tipo de evento puede estar definido para varios elementos HTML diferentes y un mismo elemento HTML puede tener asociados varios eventos distintos

- **Evento** → Acción que desencadena un usuario cuando interactúa con la página
  clik, load
- **Manejador de eventos** → Captura los eventos y llama a funciones JavaScript que los ejecuta. Acción que desencadenará la rutina

  onclick, onload

- **Controlador de eventos** → Ejecuta un segmento de código basado en eventos que ocurren dentro de la aplicación

- **Listener** → Escuchador de eventos. El objeto recibe una notificación del tipo de evento que le ocurre

Gestión de Eventos

Existen varias formas alternativas de manejar eventos en Javascript:

Forma	Ejemplo
Mediante **atributos HTML**	<button onClick="..."></button>
Mediante **propiedades Javascript**	.onclick = function() { ... }
Mediante addEventListener()	.addEventListener("click", ...)

Cada una de estas opciones se puede utilizar para gestionar eventos en Javascript de forma equivalente, pero cada una de ellas tiene sus ventajas y sus desventajas. Lo aconsejable es utilizar la última, los **listeners**, ya que son las más potentes y flexibles.

Método addEventListener

El método .addEventListener() permite añadir una escucha del **evento** indicado (*primer parámetro*), y en el caso de que ocurra, se ejecutará la función asociada indicada (*segundo parámetro*).

```
const button = document.querySelector("button");
function action() { alert("Hello!"); };
button.addEventListener("click", action);
```

En el **primer parámetro** indicamos el nombre del evento. No se precede con on los nombres de eventos y se escriben en minúsculas.
En el **segundo parámetro** indicamos la función con el código que queremos que se ejecute cuando ocurra el evento.

Es muy habitual escribir los eventos de esta otra forma:

```
const button = document.querySelector("button");
button.addEventListener("click", function() { alert("Hello!"); });
```

O con la función flecha:

```
const button = document.querySelector("button");
button.addEventListener("click", () => alert("Hello!"));
```

Método addEventListener

addEventListener() permite asociar **múltiples funciones a un mismo evento**, algo que es menos sencillo en otras las modalidades de gestionar eventos.

```
const button = document.querySelector("button");
const action = () => alert("Hola!");
const toggle = () => button.classList.toggle("red");
button.addEventListener("click", action); // Hola!
button.addEventListener("click", toggle); // Añade o elimina red CSS
```

Al pulsar el botón se efectúan ambas acciones, ya que hay dos listeners en escucha.

Método removeEventListener

Sirve para eliminar un listener que se ha añadido previamente al elemento. Para ello hay que indicar **la misma función** que añadimos con el .addEventListener().

Para el ejemplo anterior, eliminamos la función action:

```
const button = document.querySelector("button");
const action = () => alert("Hola!");
const toggle = () => button.classList.toggle("red");
button.addEventListener("click", action);
button.addEventListener("click", toggle);
button.removeEventListener("click", action); // Eliminamos el listener
```

Es posible eliminar el listener del evento porque hemos guardado en una constante la función.

Objeto Event

El objeto event se proporciona en la función asociada al evento, coincidirá con el **primer parámetro**

```
const button = document.querySelector("button");
button.addEventListener("click", (event) => { console.log(event); });
```

Si hacemos click en el botón, en la consola se nos mostrará la información de este evento. El tipo de evento asociado es PointerEvent, y tiene una serie de propiedades que no tienen porque estar presentes en otros tipos de eventos.

```
// Objeto PointerEvent {
        type: "click", // Nombre del evento
        pointerType: "mouse" // Tipo de dispositivo
        altKey: false, // ¿La tecla ALT estaba presionada?
        ctrlKey: false, // ¿La tecla CTRL estaba presionada?
        shiftKey: false, // ¿La tecla SHIFT estaba presionada?
        target: button, // Referencia al elemento que disparó el evento
        clientX: 43, // Posición en eje X donde se hizo click
        clientY: 16, // Posición en eje Y donde se hizo click
        detail: 1, // Contador de veces que se ha hecho click
        path: [], // Camino por donde ha pasado el evento
        …       // Otros...
}
```

Objeto Event

Propiedades del evento

Hay propiedades comunes que están disponibles en cualquier tipo de evento.

```
const button = document.querySelector("button");
button.addEventListener("click", (event) => {
        const { type, timeStamp, isTrusted } = event;
        console.log({ type, timeStamp, isTrusted });
        });
```

Desestructuramos tres propiedades del objeto event y las mostramos a través de una sentencia console.log():

- Type es el tipo de evento, en esta caso sería click.
- timeStamp devuelve un donde tenemos el número de milisegundos transcurridos desde que se creó el evento.
- isTrusted devuelve true si el evento en cuestión que estamos examinando es un evento real que ha surgido de una acción del usuario, o false si es un evento que ha sido emitido mediante código con un .dispatchEvent().

Objeto Event

Captura de eventos

Propiedades del objeto evento:

Propiedad	Utilidad
altKey	Devuelve si la tecla [Alt| fue pulsada durante el evento.
Button	Devuelve el botón del ratón que activó el evento: 0: botón principal. 1: botón central. Utilidad 2: botón secundario. 3y 4: cuarto y quinto botones (si los hubiera).
charCode	Contiene el valor Unicode de la tecla que se pulsó (evento keypress).
Client	Coordenada X del ratón con respecto a la ventana.
clíentY	Coordenada Y del ratón con respecto a la ventana.
ctrlKey	Devuelve si la tecla [Ctrl] fue pulsada durante el evento.
Pagex	Coordenada X del evento, relativa al documento completo.
pageY	Coordenada Y del evento, relativa al documento completo.

Objeto Event

Captura de eventos

Propiedades del objeto evento:

Propiedad	Utilidad
ScreenX	Coordenada X del evento con respecto a la pantalla.
ScreenY	Coordenada Y del evento con respecto a la pantalla.
shiftKey	Devuelve si la tecla [Mayús] fue pulsada durante el evento.
target	Referencia al elemento que lanzó el evento.
timeStamp	Devuelve el momento en el que se creó el evento.
type	Nombre del evento.

Tipos de Eventos

De Formulario

Los formularios se encuentran dentro de la etipqueta <form>.
El objeto document tiene una propiedad llamada forms, que contiene todos los formularios del documento, accediendo a ellos con forms[0], forms[1]…., no siendo necesario usar el habitual document.getElementsByTagName("form").

Propiedad	Utilidad
action	Contiene la URL que recibirá los datos del formulario
elements	Contiene todos los controles del formulario.
length	Número de controles del formulario.
method	{GETIPOST} en función del método elegido para enviarlo.
enctype	Tipo de codificación de los datos del formulario.
acceptCharset	Conjunto de caracteres del formulario.
submit()	Envía los datos del formulario a la URL de action usando method.
Reset()	Devuelve el formulario a su estado inicial.
onX	Todos los eventos asociados al formulario, siendo X el nombre del evento: onAbort, onBlur, onCancel, onClick...

Tipos de Eventos

De Formulario

Además, se puede trabajar sobre los controles de un formulario, que son elementos HTML, y como tales, tienen una serie de características comunes. Así, se puede trabajar sobre sus atributos, como con cualquier elemento, usando getAttribute("valor"). Pero es mejor usar las propiedades específicas de los controles de formulario, que asumen los cambios que se realicen en los elementos de forma dinámica.

Propiedad	Utilidad
action	Contiene la URL que recibirá los datos del formulario
elements	Contiene todos los controles del formulario. Tiene una entrada por cada control del formulario.
length	Número de controles del formulario.
method	{GET/POST} en función del método elegido para enviarlo.
enctype	Tipo de codificación de los datos del formulario.
acceptCharset	Conjunto de caracteres del formulario.
submit()	Envía los datos del formulario a la URL de action usando method.
Reset()	Devuelve el formulario a su estado inicial.
onX	Todos los eventos asociados al formulario, siendo X el nombre del evento: onAbort, onBlur, onCancel, onClick...

Tipos de Eventos

De Formulario

Propiedades de los controles de un formulario:

Propiedad	Utilidad
accept	Tipos de archivos que se permiten en un control de tipo file.
autocomplete	Si el valor del control puede ser autocompletado por el navegador.
name	Nombre del control.
type	Tipo de control.
value	Valor actual del control.
checked	true si el control está activado, false si no lo est.
defaultChecked	Valor predeterminado de la propiedad checked.
disabled	true si el control está deshabilitado, false si está habilitado.
hidden	El control es invisible para el usuario, pero no para el programador.
readonly	true si el control es de solo lectura (no modificable). false si no lo es.
required	true si proporcionarle un valor al control es obligatorio. false si no lo es.
maxLength	Anchura máxima del texto.

Tipos de Eventos

De Formulario

Propiedades de los controles de un formulario:

Propiedad	Utilidad
min	Valor mínimo para el control.
max	Valor máximo para el control.
pattern	Una expresión regular contra la que el valor del control es evaluado.
placeholder	Una pista para el usuario sobre lo que debe introducir en el control.
size	Tamaño inicial del control
step	Tamaño del cambio en el valor de un control.
selectionStart	En una selección de texto la posición del primer carácter seleccionado.
selectionEnd	En una selección de texto, la posición del último carácter seleccionado.

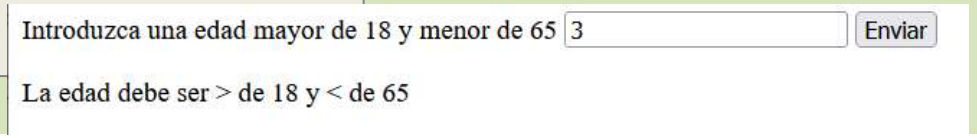
Tipos de Eventos

De Formulario

El evento más importante del formulario en el **submit**, enviando el formulario a la URL indicada en action.
Hay que hacer la validación del formulario antes del envío.
Algunos frameworks incorporan utilidades Javascript que realizan estas acciones, como Bootstrap, Materielize..etc.
Si los datos no cumplen la validación hay que anular el envío del formulario.
Ejemplo:

```
<form name="formulario" action="procesa.php" method="POST">
        Introduzca una edad mayor de 18 y menor de 65
        <input type="text" name="edad" placeholder="Edad" value=""/>
        <input type="submit" name="submit" value="Enviar"/>
</form>
<div id="errores"></div>
```

```
let formularioEdad=document.forms[0];
let errores=document.getElementById("errores");
formularioEdad.addEventListener("submit",(evento)=>{
    let edad=parseInt(formularioEdad.elements["edad"].value);
    if((edad<=18)||(edad>=65)){
        evento.preventDefault();
        errores.innerHTML="La edad debe ser > de 18 y < de 65";
    }
})
```



Tipos de Eventos

De Formulario

Reseteo del formulario:

Se puede resetear un con el evento reset():

```
formulario.reset();
```

Cambiar el valor:

El evento **change**() se produce cuando cambia el valor de un control, en el momento que éste pierde el foco.

Ganar y perder el foco:

Cuando un control recibe el foco, se genera el evento **focus()**, y cuando lo pierde el evento blur().

Tipos de Eventos

De Ratón

Representa eventos que ocurren debido a que el usuario interactúa con un dispositivo señalador (como un mouse)

Evento	Funcionamiento
click	Hacer clic sobre el botón principal del dispositivo.
dblclick	Hacer doble clic sobre el botón principal del dispositivo.
mousedown	Cuando se pulsa y justo antes de soltarlo, se lanza el evento.
mouseup	El evento se lanza cuando se suelta el botón.
mouseenter	El evento se lanza cuando el puntero se sitúa sobre el elemento que captura el evento.
mouseleave	El evento se lanza cuando el puntero deja de situarse sobre el elemento que captura el evento.
mousemove	Mientras se está dentro del elemento, el evento se lanza cada vez que se mueve el puntero.
mouseover	El evento se lanza cuando el puntero se sitúa sobre el elemento que lo captura o sobre cualquiera de sus hijos.
mouseout	El evento se lanza cuando el puntero deja de situarse sobre el elemento que lo captura o sobre cualquiera de sus hijos.
contextmenu	El evento se lanza cuando se solicita un menú contextual.

Tipos de Eventos

De teclado

Describen la interacción del usuario con el teclado; cada evento describe una única interacción entre el usuario y una tecla (o combinación de una tecla con teclas modificadoras) en el teclado

Los eventos que identifican qué tipo de actividad ocurrió en el teclado son:

- Keydown: el evento se lanza tras pulsar y antes de soltar la tecla
- Keypress: el evento se lanza tras pulsar y soltar una tecla
- Keyup: el evento se lanza tras soltar la tecla

Propiedades

key → Devuelve el valor de la tecla presionada

Los modificadores son teclas especiales que se utilizan para generar caracteres especiales o provocar acciones especiales cuando se utilizan en combinación con otras teclas

AltKey
CtrlKey
ShiftKey
MetaKey

Devuelven true si se ha pulsado una de estas teclas especiales

location-> indica qué zona del teclado se ha pulsado: 0 (teclado estándar), 1 (parte izquierda), 2 (parte derecha) y 3 (teclado numérico).

Tipos de Eventos

De teclado

Ejemplo:

```
let todobody=document.getElementsByTagName("body")[0];
todobody.addEventListener("keypress",(evento)=>{
    console.log(`Lanzando Keypress de la tecla ${evento.key}`);
});
todobody.addEventListener("keydown",(evento)=>{
    console.log(`Lanzando Keydown de la tecla ${evento.key}`);
});
todobody.addEventListener("keyup",(evento)=>{
    console.log(`Lanzando Keyup de la tecla ${evento.key}`);
});
```

En el ejemplo se han asociado los tres eventos al elemento body para que estén disponibles en todo el cuerpo del documento.

```
Lanzando Keydown de la tecla k
Lanzando Keypress de la tecla k
Lanzando Keyup de la tecla k
```

Método .preventDefault

Algunos elementos tienen un **comportamiento por defecto**.
Ejemplo, el elemento <details> muestra el texto del elemento <summary>, si se pulsa sobre el, se despliega el resto del contenido de <details>. Si se vuelve a pulsar, se oculta nuevamente. Ese es su **comportamiento por defecto**.
Pueden existir situaciones donde queremos que se anule este comportamiento y no se realice. Por ejemplo, para reimplementarlo nosotros, o cambiar su funcionalidad habitual.
Para ello, tenemos el método .preventDefault().
Este método establece el flag .defaultPrevented a true y podremos evitar el comportamiento por defecto de dicho evento y añadirle otro diferente:

```
const details = document.querySelector("details");
details.addEventListener("click", (event) => {
        event.preventDefault();
        alert("Has hecho click pero hemos eliminado el comportamiento por defecto.");
        });
```

Ahora, al pulsar sobre el elemento <details> ya no se expande ni se contrae, por lo que ahora podríamos crear nuestra propia lógica para reimplementar esta funcionalidad.

Método addEventListener (Continuación)

El método addEventListener, tiene un último parámetro, opcional, un booleano, que tiene que ver con el flujo de eventos.

```
elemento_a_escuchar.addEventListener('evento',función[,booleano])
```

elemento_a_escuchar	cualquier elemento presente en un documento (id, etiqueta, propiedades de otro nodo)
evento	suceso ocurrido sobre el elemento, con o sin interacción del usuario (OJO!! Sin el on. Ejemplo: 'click')
función	cualquier función definida que queramos que se ejecute cuando ocurra el evento.
booleano	define el orden del flujo de eventos **`true`** captura del evento de padre a hijos, **`false`** de hijo a padres

Método addEventListener (Continuación)

Flujo de eventos

Supongamos que tenemos:

```
<section>
        <p>
            Aquí se capturan eventos asociados
            a un <span id="miSpan">elemento</span> HTML
        </p>
</section>
```

Cuando hacemos clic en span no sólo lo estamos haciendo sobre él, sino sobre p y sobre section. Si sólo hay una función asignada a una escucha para el span no hay mayor problema, pero si además hay una para p y otra para el section, ¿cuál es el orden en que se deben lanzar las tres funciones?

Método addEventListener (Continuación)

Flujo de eventos

A esto da respuesta el flujo de eventos

```
let miSection=document.getElementsByTagName("section")[0];
let miP=document.getElementsByTagName("p")[0];
let miSpan=document.getElementById("miSpan");
miSection.addEventListener("click",()=>{
    console.log("<section>:Capturando el evento");
})
miP.addEventListener("click",()=>{
    console.log("<p>:Capturando el evento");
})
miSpan.addEventListener("click",()=>{
    console.log("<span>:Capturando el evento");
})
```

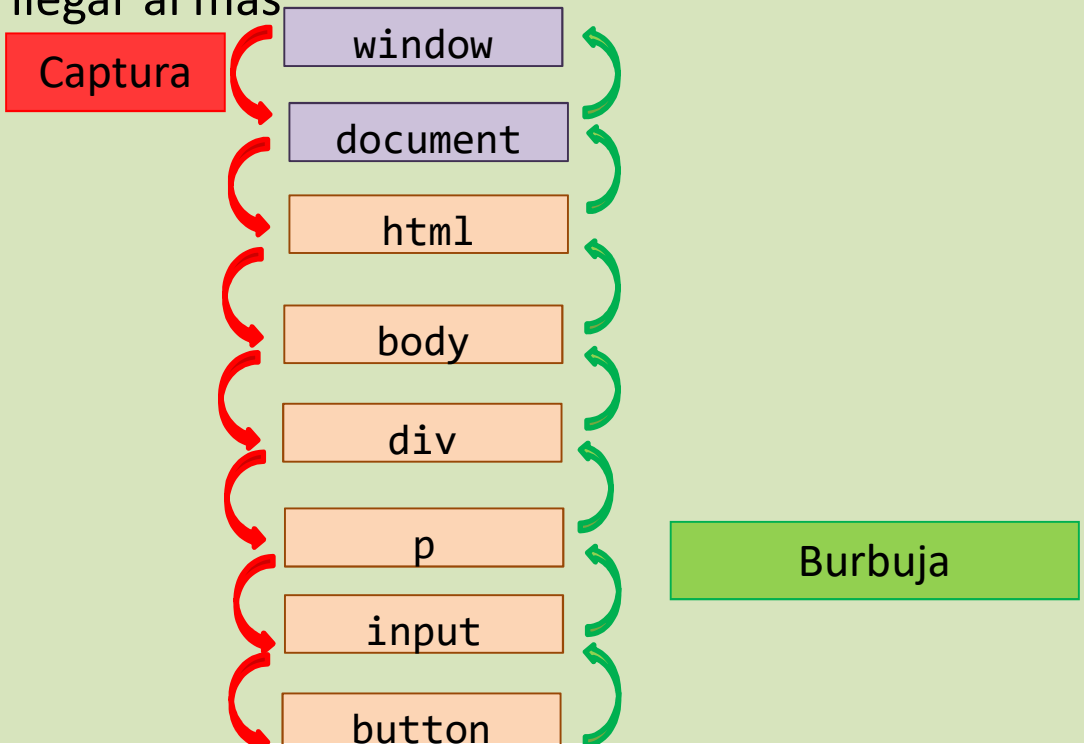
```
<span>:Capturando el evento
<p>:Capturando el evento
<section>:Capturando el evento
```

Por defecto, se propaga desde el elemento más interno al más externo. Pero hay dos formas, según el siguiente cuadro (según el tercer argumento de addeventListener):

Método addEventListener (Continuación)

Flujo de eventos

- **Burbuja**. Por defecto. Esta propagación se va dando del elemento más interno al más externo, hasta window (***false***)
- **Captura**. La propagación se realiza de elementos padre hasta llegar al más interno (***true***)



Método addEventListener (Continuación)

Flujo de eventos

Método Captura:

```
let miSection=document.getElementsByTagName("section")[0];
let miP=document.getElementsByTagName("p")[0];
let miSpan=document.getElementById("miSpan");
miSection.addEventListener("click",()=>{
    console.log("<section>:Capturando el evento");
},true);
miP.addEventListener("click",()=>{
    console.log("<p>:Capturando el evento");
},true);
miSpan.addEventListener("click",()=>{
    console.log("<span>:Capturando el evento");
},true);
```

```
<section>:Capturando el evento
<p>:Capturando el evento
<span>:Capturando el evento
```

En este caso, el flujo de evento se propaga de fuera a dentro.

Método .stopPropagation

Evita la propagación adicional del evento actual en las fases de captura y burbuja a través del árbol

`evento.stopPropagation()`

```
let miSection=document.getElementsByTagName("section")[0];
let miP=document.getElementsByTagName("p")[0];
let miSpan=document.getElementById("miSpan");
let veces=0;
miSection.addEventListener("click",()=>{
    console.log(`<section>:Capturando el evento ${veces} veces`);
});
miP.addEventListener("click",()=>{
    console.log(`<p>:Capturando el evento ${veces} veces`);
});
miSpan.addEventListener("click",(evento)=>{
    veces++;
    console.log(`<span>:Capturando el evento ${veces} veces`);
    if (veces>1)
        evento.stopPropagation();
});
```

```
<span>:Capturando el evento 1 veces
<p>:Capturando el evento 1 veces
<section>:Capturando el evento 1 veces
<span>:Capturando el evento 2 veces
<span>:Capturando el evento 3 veces
<span>:Capturando el evento 4 veces
```

La propagación se produce sólo la primera vez que se clica sobre el elemento. La segunda vez se ejecuta stopPropagation

Generar eventos con Event

No sólo podemos asignar controladores a eventos que se producen, sino que también podemos generar eventos desde javascript.

`Constructor de eventos`

Podemos crear objetos Event así:

```
let event = new Event(type[, options]);
```

`dispatchEvent`

Después de que se crea un objeto de evento, debemos "ejecutarlo" en un elemento usando dispatchEvent. Luego, los controladores reaccionan como si fuera un evento normal del navegador.

Lanza un evento en el sistema de eventos. El evento está sujeto al mismo comportamiento y capacidades que si fuera un evento de lanzamiento directo

```
objeto.dispatchEvent(evento)
```

Generar eventos con Event

Ejemplo:

```
<a id="miEnlace" href="google.es">Enlace</a>
<span id="miSpan">Ejecutar el evento del enlace</span>
```

```
let miSpan=document.getElementById("miSpan");
let miEnlace=document.getElementById("miEnlace");
miEnlace.addEventListener("click",(evento)=>{
    evento.preventDefault();
    console.log("Gestor del evento del enlace");
});
miSpan.addEventListener("click",()=>{
    let miEvento=new Event("click");
    console.log("Lanzando evento");
    miEnlace.dispatchEvent(miEvento);
});
```

```
Lanzando evento
Gestor del evento del enlace
Lanzando evento
Gestor del evento del enlace
```

Objeto Event

Target y CurrentTarget

Propiedad `target`

Es una referencia al objeto en el cual se lanza el evento. Dónde comenzó el evento. Devuelve un objeto, por lo que se utilizará con sus propiedades.

```
elemento = objeto.target
```

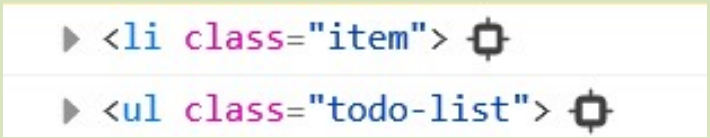
currentTarget

Controlador de eventos (manejador de eventos) es llamado durante la fase de burbujeo o captura del evento. En qué elemento se adjuntó el evento o el elemento cuyo `eventListener` desencadenó el evento

```
elemento = objeto.currentTarget
```

Ejemplo:

```
<ul class="todo-list">
        <li class="item">Elemento</li>
</ul>
```




```
let list = document.querySelector(".todo-list");
list.addEventListener("click", e => {
  console.log(e.target);
  console.log(e.currentTarget);
});
```

Eventos de carga del documento

DOMContentLoaded y Load

El ciclo de vida de una página HTML tiene tres eventos importantes:

•DOMContentLoaded – el navegador HTML está completamente cargado y el árbol DOM está construido, pero es posible que los recursos externos como <img> y hojas de estilo aún no se hayan cargado. Es un evento del objeto document.

```
document.addEventListener("DOMContentLoaded", funcion);
```

•load – no solo se cargó el HTML, sino también todos los recursos externos: imágenes, estilos, etc. Es un método del objeto window.

Tipos de Eventos

De arrastrar y soltar

Son elementos que se lanzan al utilizar elementos definidos como arrastrables, es decir, tienen a true el atributo draggable.
Al tratarse de una acción con un origen y un destino, se diferencian dos conjuntos de eventos:

Evento	Funcionamiento
Elemento que se arrastra	
drag	Cada vez que se mueve una vez que se ha iniciado el arrastre.
dragstart	Al comenzar a arrastrar el elemento.
dragstop	Al finalizar el arrastre del elemento.
Elemento donde puede soltarse	
dragenter	Cuando el elemento que se arrastra entra en el elemento destino.
dragleave	Cuando el elemento que se arrastra sale del elemento destino.
dragover	Cada vez que continúa el arrastre sobre el elemento de destino.
drop	Cuando el elemento que se arrastra se suelta sobre el elemento destino

Tipos de Eventos

De arrastrar y soltar

Para evitar acciones de defecto del navegador sobre elementos destinados a ser arrastrados y soltados deberemos usar preventDefault().
Todos los eventos de arrastre tienen una propiedad denominada **dataTransfer** que se usa para almacenar la información sobre lo que se está arrastrando. La información sobre lo que se está arrastrando se compone de dos elementos: el tipo o formato de lo que se arrastra, y su valor. Por ejemplo el tipo puede ser texto y su valor "iesleonardodavinci.com". O el tipo podría ser una url y su valor "http://iesleonardodavinci.com".
Métodos de dataTransfer:

- **setData(tipo,dato):** usado para declarar los datos a ser enviados y su tipo. El método puede recibir tipos de datos regulares (como *text/plain*, *text/html* o *text/uri-list),* tipos de datos especiales (como URL o Text) o incluso tipos de datos personalizados. Un método setData() debe llamarse por cada tipo de datos que queremos enviar en la misma operación.
- **getData(tipo): r**etorna los datos enviados por el origen, pero solo del tipo especificado.
- **clearData():** Este método remueve los datos del tipo especificado.

Tipos de Eventos

De arrastrar y soltar

Ejemplo:

```
<ul id="origen">
        <li draggable="true" id="origen1">Primer elemento</li>
        <li draggable="true" id="origen2">Segundo elemento</li>
        <li draggable="true" id="origen3">Tercer elemento</li>
</ul>
<div id="destino">NUEVA LISTA</div>
```

```
let elementosOrigen=document.querySelectorAll("#origen>li");
let elementoDestino=document.getElementById("destino");
elementosOrigen.forEach(function(elemento){
    elemento.addEventListener("dragstart",(evento)=>iniciadoArrastre(evento));
});
elementoDestino.addEventListener("dragover",(evento)=>permitirSoltar(evento));
elementoDestino.addEventListener("drop",(evento)=>soltar(evento));
function iniciadoArrastre(evento){
    evento.dataTransfer.setData("idElementoOrigen",evento.target.id);
}
function permitirSoltar(evento){
    evento.preventDefault();
}
function soltar(evento){
    evento.preventDefault();
    let elementoArrastrandose=evento.dataTransfer.getData("idElementoOrigen");
    elementoDestino.appendChild(document.getElementById(elementoArrastrandose));
}
```

Tipos de Eventos

De arrastrar y soltar

Resultado:

- Primer elemento
- Segundo elemento
- Tercer elemento

NUEVA LISTA

- Segundo elemento
- Tercer elemento

NUEVA LISTA
- Primer elemento

NUEVA LISTA
- Primer elemento
- Segundo elemento
- Tercer elemento