RESPONDER A EVENTOS

React nos permite añadir manejadores de eventos a nuestro jsx, básicamente los manejadores de eventos son funciones propias que se ejecutan en respuesta de interacciones del usuario, como por ejemplo clicks o hovers con el mouse.

En este capítulos vamos a aprender:
1- Diferentes maneras de escribir un manejador de eventos
2- Como pasar la lógica de manejo de eventos desde un componente padre
3- Como los eventos se propagan y cómo detenerlos

Para aprender cómo funcionan los manejadores de eventos lo primero que vamos a hacer es crear un simple botón en nuestro componente App.

```
function App() {
 return (
   <button>No hago nada!</button>
 );
}
export default App;
```

Ahora para hacer que este botón muestre un mensaje cuando el usuario le dé un click debemos seguir 3 simples pasos:
1- Declarar una función handleClick dentro del componente App
2- Implementar la lógica dentro de la función handleClick, en este caso voy a poner un alert
3- Agregar el onClick pasando la función al botón

```
function App() {
 function handleClick() {
   alert('Hola React!')
 }
 return (
   <button onClick={handleClick}>Click me</button>
 );
}
export default App;
```

Por otro lado también podemos definir los manejadores de eventos directamente en el jsx:

```
function App() {
 return (
   <button onClick={function() {
     alert('Hola React')
   }}>Click me</button>
 );
}
export default App;
```

O también para hacerlo mas corto podemos poner una función flecha:

```
function App() {
 return (
   <button onClick={() => alert('Hola React')}>Click me</button>
 );
}
export default App;
```

Recuerda que las funciones que pasan a los manejadores de eventos deben ser pasadas, no llamadas, por ejemplo

```
<button onClick={handleClick()}>Click me</button>
```

Esta línea de aquí lo que va a hacer es que va a renderizar la alerta cada vez que carguemos la página web, y no es lo que queremos.

Luego cuando escribimos código en una misma linea la trampa se presenta de otra manera:

```
   <button onClick={alert('¡Hola React!')}>
     Click
     </button>
```

Aquí también el código se va a ejecutar cada vez que renderizar la página.

PROPS EN LOS MANEJADORES DE EVENTOS

Como los manejadores de eventos son declarados dentro de un componente estos tienen acceso a los props del componente, En este ejemplo tenemos un botón que al recibir el click muestra una alerta con un prop de mensaje:

```
function AlertButton({ message, children }) {
 return (
   <button onClick={() => alert(message)}>
     {children}
   </button>
 );
}
export default function App() {
 return (
   <div>
     <AlertButton message="Hola React!">
       Hola React
     </AlertButton>
     <AlertButton message="Hola JavaScript!">
       Hola JavaScript
     </AlertButton>
   </div>
 );
}
```

PASAR MANEJADORES DE EVENTOS COMO PROPS

En algunos casos vamos a querer que el componente padre especifique un manejador de eventos de un componente hijo, consideremos 2 botones, que dependiendo de donde estamos usando el componente Buton, es posible que queramos ejecutar una función diferente, en un caso vamos a ejecutar por consola y en otro por una alerta:

```
function Button({ onClick, children }) {
 return (
   <button onClick={onClick}>
     {children}
   </button>
 );
}

export default function App() {
 return (
   <div>
     <Button onClick={() => console.log('Hola!')}>
       Saluda por consola
     </Button>
     <Button onClick={() => alert('Chau!')}>
```

```
      Despide por alerta
    </Button>
  </div>
 );
}
```

Podemos ponerle cualquier nombre al prop del manejador de evento, cambiemoslo a onSmash.

```
function Button({ onSmash, children }) {
 return (
  <button onClick={onSmash}>
    {children}
  </button>
 );
}

export default function App() {
 return (
  <div>
    <Button onSmash={() => console.log('Hola!')}>
      Reproducir película
    </Button>
    <Button onSmash={() => alert('Chau!')}>
      Subir imagen
    </Button>
  </div>
 );
}
```

PROPAGACIÓN DE EVENTOS

Los manejadores de eventos también atrapan eventos de cualquier componente hijo pueda tener, hagamos un ejemplo donde tenemos un un manejador de eventos en un div que dentro tiene 2 botones con sus manejadores de eventos:

```
function Button({ onClick, children }) {
 return (
  <button onClick={onClick}>
    {children}
  </button>
 );
}

export default function App() {
 return (
```

```
    <div onClick={() => alert('Soy un div')}>
      <Button onClick={() => console.log('Hola!')}>
        Saluda por consola
      </Button>
      <Button onClick={() => alert('Chau!')}>
        Despide por alerta
      </Button>
    </div>
  );
}
```

DETENER LA PROPAGACIÓN

Para detener una propagación debemos llamar a s.stopPropagation() como en este componente Button hace:

```
function Button({ onClick, children }) {
  return (
    <button onClick={e => {e.stopPropagation();
    onClick()}}>
      {children}
    </button>
  );
}

export default function App() {
  return (
    <div onClick={() => alert('Soy un div')}>
      <Button onClick={() => console.log('Hola!')}>
        Saluda por consola
      </Button>
      <Button onClick={() => alert('Chau!')}>
        Despide por alerta
      </Button>
    </div>
  );
}
```

EVITAR EL COMPORTAMIENTO POR DEFECTO

Cuando tenemos un formulario por ejemplo al pulsar el botón que hay dentro de él lo que pasa es que por defecto va a recargar la pagina, y aveces ese es un comportamiento no deseado:

```
export default function App() {
 return (
   <form onSubmit={() => alert('¡Enviando!')}>
     <input />
     <button>Send</button>
   </form>
 );
}
```

Para evitar esto lo que podemos hacer es llamar a e.preventDefault():

```
export default function App() {
 return (
   <form onSubmit={e => {
     e.preventDefault();
     alert('¡Enviando!');
   }}>
     <input />
     <button>Enviar</button>
   </form>
 );
}
```