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 a interacciones del usuario, como clicks, entre otros.

Aqui tengo un boton que aún no hace nada:

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

Podemos hacer que muestre un mensaje cuando el usuario le haga click en 3 pasos:
1- Declaramos una función llamada handleClick dentro del componente.
2- Implementar la lógica dentro de la función como una alert o un console.log().
3- Agregar onClick dentro del botón.

```
function App () {
   function handleClick() {
       alert('¡Me cliqueaste!');
   }
   return (
     <button onClick={handleClick}>
       No hago nada
     </button>
   );
 }
export default App
```

Por lo general las funciones manejadoras de eventos se encuentran dentro del mismo componente y suelen comenzar con la palabra handle.

También lo que podemos hacer es poner la función directamente dentro del jsx:

```
       <button onClick={function handleClick() {
           alert('¡Me cliqueaste!');
       }}>
       Click me!
     </button>
```

O con una función flecha para dejarlo aun mas corto:

```
      <button onClick={() => {
           alert('¡Me cliqueaste!');
       }}>
       Click me!
     </button>
```

LEYENDO LOS PROPS EN LOS MANEJADORES DE EVENTOS

Como los manejadores de eventos son declarados declarados dentro de un componente, tienen acceso a los props del componente, En este ejemplo tenemos un botón que, cuando recibe un clic, muestra una alerta con su propio mensaje:

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

function App () {
   return (
   <div>
     <AlertButton message="Hola!">
       Hola
     </AlertButton>
     <AlertButton message="Chau!">
       Chau
     </AlertButton>
   </div>
   );
 }
export default App
```

PROPAGACIÓN DE EVENTOS

Los manejadores de eventos tambien atrapan eventos de cualquier componente hijo que tu componente pueda llegar a tener, en este ejemplo vamos a tener un donde continente 2 botones:

```
function App () {
   return (
     <div onClick={() => {
       alert('¡Cliqueaste el Div!');
     }}>
       <button onClick={() => alert('Hola!')}>
         Hola
       </button>
       <button onClick={() => alert('Chau!')}>
         Chau
       </button>
     </div>
   );
 }
export default App
```

DETENER LA PROPAGACIÓN

Para detener la propagación lo que debemos hacer es poner un e.stopPropagation().

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

function App () {
   return (
     <div onClick={() => {
       alert('¡Cliqueaste el Div!');
     }}>
       <Button onClick={() => alert('Hola!')}>
         Hola
       </Button>
       <Button onClick={() => alert('Chau!')}>
         Chau
       </Button>
```

```
      </div>
    );
 }
export default App
```

EVITA EL COMPORTAMIENTO POR DEFECTO

Algunos eventos como por ejemplo los formularios cuando le damos click al boton este va a recargar la pagina completa y aveces eso es un comportamiento no deseado,

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

Para prevenir esto tenemos el e.preventDefault():

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