Events in React

Pages that react: onClick, onChange and onSubmit, the event object, why onClick={fn()} fires too early, passing arguments with arrow functions, and preventDefault.

Introduction

A static page shows; an app **responds**. Clicks, typing, mouse movement, form submits — in React these are **events**, and you handle them by passing a **function** to a prop like `onClick`. This lesson gathers the event patterns used across the whole series into one place. *(Roman Urdu: Event woh lamha hai jab user kuch karta hai — React mein har event ka jawab aik function hota hai.)*

Key points

- **The pattern is one line:** `<button onClick={handleClick}>` — you hand React the function; React calls it when the event happens. Event props are camelCase: `onClick`, `onChange`, `onSubmit`, `onMouseEnter`, `onKeyDown`.
- **Pass the function, never call it:** `onClick={handleClick}` waits for the click. `onClick={handleClick()}` calls the function **while rendering** — the famous too-early bug, demoed live in the video.
- **The event object arrives free:** React passes an event object to your handler — `e.target` is the element, `e.target.value` an input's text, `e.preventDefault()` stops the browser's default move (form reloads, link jumps).

  (Roman Urdu: Handler ko event object muft milta hai — target, value, aur default rokna, sab usi mein hai.)
- **Arguments ride in an arrow:** to pass your own argument, wrap the call: `onClick={() => selectCourse("React")}` — the arrow waits for the click, then calls with your value.
- **Typing is an event stream:** an input fires `onChange` on every keystroke — that is why controlled inputs (5.6) mirror text live.
- **Handlers are just functions:** define them in the component, name them for the event (`handleClick`, `handleChange`, `handleSubmit`) — the `handle…` naming is the community habit you will see everywhere.
- **Keyboard and mouse count too:** `onKeyDown` can watch for Enter or Escape; `onMouseEnter` / `onMouseLeave` drive hovers and tooltips without any CSS hacks.

Code examples

Example 1 — onClick, the right way and the too-early way

```
function DoorBell() {
  function ring() {
    console.log("Ding dong!");
  }

  return (
    <div>
      <button onClick={ring}>Right — waits for the click</button>
      <button onClick={() => ring()}>Also right — wrapped in an arrow</button>
    </div>
  );
}

// WRONG version (do not ship): onClick={ring()} — ring() runs during
// render, the log fires on page load, and the click does nothing.
```

What it does: Both correct buttons hand React a function to call later — one by name, one wrapped in an arrow. The wrong version (kept as a comment) *calls* the function while the component renders and passes its return value instead — the exact bug from the video.

What you will see: with the correct buttons, "Ding dong!" logs only when clicked. With the wrong one, it logs once on page load and the button is dead.

Example 2 — The event object: one handler, many buttons

```
function ColorPicker() {
  function pick(e) {
    console.log("You picked:", e.target.textContent);
  }

  return (
    <div>
      <button onClick={pick}>Red</button>
      <button onClick={pick}>Green</button>
      <button onClick={pick}>Gold</button>
    </div>
  );
}
```

What it does: One handler serves all three buttons because the event object carries the source: `e.target` is the button that was clicked, and its text tells the handler which colour was chosen. The same idea powered

"one function, many buttons" with `e.target.name` in the JavaScript series — now in React form.

What you will see: clicking Gold logs "You picked: Gold"; Red logs "Red" — one function, zero duplication.

Example 3 — Passing arguments, and onSubmit with preventDefault

```
import { useState } from "react";

function CourseShop() {
  const [picked, setPicked] = useState("none");

  function buy(course) {
    setPicked(course);
  }

  function handleSubmit(e) {
    e.preventDefault();
    console.log("Order placed for:", picked);
  }

  return (
    <form onSubmit={handleSubmit}>
      <button type="button" onClick={() => buy("React")}>Pick React</button>
      <button type="button" onClick={() => buy("CSS")}>Pick CSS</button>
      <p>Picked: {picked}</p>
      <button type="submit">Place order</button>
    </form>
  );
}
```

What it does: The pick buttons need an argument, so each wraps the call in an arrow — `() => buy("React")`. The submit handler starts with `preventDefault`, so placing the order logs the choice instead of reloading the page and losing it.

What you will see: pick a course and the line updates; press "Place order" and the console logs the order — the page stays exactly where it was.

Quick recap

- Events are handled by **passing functions** to props: `onClick`, `onChange`, `onSubmit` — camelCase, called by React when the moment comes.
- `onClick={fn()}` fires during render — pass `fn`, or wrap with an argument as `() => fn(x)`.
- The **event object** brings `e.target`, `e.target.value`, and `e.preventDefault()` for stopping reloads and jumps.

Common mistakes

1. **Calling the handler in JSX** — `onClick={buy()}` runs at render time and the button then does nothing. The fix: pass the function (`onClick={buy}`) or an arrow (`onClick={() => buy(id)}`).
2. **Forgetting preventDefault on submit** — the form reloads the page, state resets, and your handler's work vanishes. The fix: `e.preventDefault()` as the submit handler's first line, always.
3. **Inline logic avalanches** — giant anonymous functions written inside JSX become unreadable and untestable. The fix: named handlers in the component body (`handleSubmit`), with JSX holding one clean name.

Practice

1. Build a `ReactionBar`: three buttons (Like, Love, Wow). One handler using the event object counts clicks per reaction in a single state object, and each button shows its own count. Use the argument-passing arrow to say *which* reaction was clicked.
2. Build a tiny keyboard demo: an input whose `onKeyDown` watches for Enter (add the current text to a list below and clear the input) and Escape (just clear the input). Show the list of added items with their index numbers.

Practice quiz

1. What is wrong with onClick={handleClick()}?

- Nothing, it is the correct way
- It calls the function while rendering, the famous too-early bug; pass the function, onClick={handleClick}
- It is too slow
- It only works for forms

Answer: It calls the function while rendering, the famous too-early bug; pass the function, onClick={handleClick}

2. How do you pass your own argument to a React event handler?

- onClick={selectCourse("React")}, directly
- Wrap the call in an arrow: onClick={() => selectCourse("React")}, so the arrow waits for the click
- Arguments cannot be passed to handlers
- With e.target only

Answer: Wrap the call in an arrow: onClick={() => selectCourse("React")}, so the arrow waits for the click

3. What must the first line of every React submit handler be?

- setState
- e.preventDefault(), or the form reloads the page and state is lost
- console.log

- return null

Answer: e.preventDefault(), or the form reloads the page and state is lost

---

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/react-events · More free tools and notes at https://scopetreasure.com