Client-Side Web Programming: JavaScript (Part 4)

Copyright © 2026 by

Robert M. Dondero, Ph.D.

Princeton University

Objectives

- We will cover:
  - React: concepts
  - React: fundamentals
  - React: useState hooks
  - React: props
  - React: useEffect hooks
  - React: realistic example

Agenda

- **React: concepts**
- React: fundamentals
- React: useState hooks
- React: props
- React: useEffect hooks
- React: realistic example

React: Concepts



Jordan
Walke

Note: **React** == **React.js** == **ReactJS**

React: Concepts

- The fundamental idea…

- jQuery (or no client-side library)
  - HTML code contains JavaScript code
- React
  - HTML code is generated by JavaScript code

React: Concepts

- Two styles of React programming:
  - **Class-based**
    - Older
    - (pro) Easier to understand?
    - (con) Semi-deprecated
  - **Functional**
    - Newer
    - (pro) More succinct
    - (con) Harder to understand?

- We'll cover functional…

React: Concepts

- Key concept: ***components***
  - Pgmmer defines components
  - Each component is defined as a function
  - Each component returns a DOM subtree
  - Components can be arranged hierarchically
  - Each component can:
    - Have **state**
    - Accept properties (**props**) from its parent component

React: Concepts

- Key concept: ***virtual DOM***
  - Corresponding to the browser DOM tree, React maintains a *virtual DOM tree*
  - For each browser DOM node, there is a *virtual* DOM node

React: Concepts

- Key concept: ***virtual DOM*** (cont.)
  - At initial rendering, and when the state of a component changes:
    - Component returns a DOM subtree
    - React updates the virtual DOM tree with the component's DOM subtree
    - React compares the updated virtual DOM tree with the previous version to determine diffs
    - Using diffs, React ***reconciles*** the virtual DOM tree and browser DOM tree
      - Updates the fewest possible browser DOM nodes

React: Concepts

- Key supporting technology: ***JSX (JavaScript XML)***
  - Allows embedding of HTML-like code (actually, XML code) in JavaScript code

React: Concepts

- Key concept: ***React bundles***
  - Typically React programmers use **webpack**, **Vite**, **Next.js**, … to create **React bundles**
  - Here, for simplicity, we will not create React bundles
    - But see next lecture

Agenda

- React: concepts
- **React: fundamentals**
- React: useState hooks
- React: props
- React: useEffect hooks
- React: realistic example

React: Fundamentals

- “Hello World” example



React: Fundamentals

- See **hellonoreact.html**
  - Things to note:
    - World's simplest web page

React: Fundamentals

- See **helloreact.html**
  - Things to note:
    - Overall structure
    - (Minimal) use of HTML
      - Almost all HTML code is generated by JavaScript code
    - Defining a component
      - Which returns a DOM subtree

React: Fundamentals

- See **helloreact.html** (cont.)
  - Things to note:
    - `React.StrictMode`
      - "Lets you find common bugs in your components early during development."
      - "Use StrictMode to enable additional development behaviors and warnings for the component tree inside."

https://react.dev/reference/react/StrictMode

React: Fundamentals

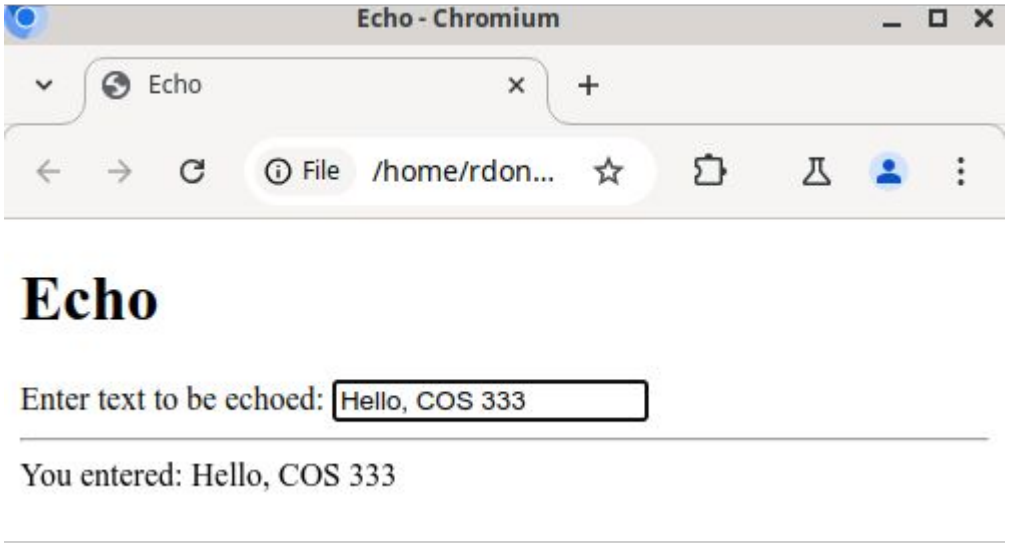
- See **helloreactjsx.html**
  - Things to note:
    - Fetch of Babel
    - Use of Babel
      - Translates JSX to JavaScript
    - Use of JSX

Agenda

- React: concepts
- React: fundamentals
- **React: useState hooks**
- React: props
- React: useEffect hooks
- React: realistic example

React: useState Hooks

- “Echo” example



React: useState Hooks

- See **echo1.html**
  - Things to note:
    - Event handling via JavaScript
- See **echo2.html**
  - Things to note:
    - Event handling mostly via HTML

React: useState Hooks

- See **echoreact1.html**
  - Things to note:
    - ***useState hook***
      - Defines a **state** variable and a function that can be called to change its value
    - When a component's **state** changes, React:
      - Uses state to create and return a DOM subtree
      - Updates virtual DOM tree with DOM subtree
      - Reconciles virtual DOM tree with browser DOM tree

React "reacts" to each change in state

Aside: Arrow Functions

- ***Arrow function def expressions***
  - Informally ***arrow functions***
  - Arrow functions vs ordinary functions:
    - Often more succinct
    - Same semantics - **mostly!!!**
  - See **Appendix 1** for more information

Aside: Arrow Functions

- See **arrow.js**

```
$ node arrow.js
25
25
25
25
25
30
30
30
30
hi
hi
hi
$
```

React: useState Hooks

- See **echoreact2.html** (cont.)
  - Things to note:
    - Uses arrow function definitions exclusively

React: useState Hooks

- See **echoreact3.html** (cont.)
  - Things to note:
    - Uses arrow function definitions as function arguments

Agenda

- React: concepts
- React: fundamentals
- React: useState hooks
- **React: props**
- React: useEffect hooks
- React: realistic example

React: Props

- See **echoreact4.html**
  - Things to note:
    - Components arranged in a tree (hierarchy)
    - Use of a **prop** to send data **downward**
      - From parent component to child component
      - From `App` to `EchoOutput`
    - Use of a **prop** to send data **upward**
      - From child component to parent component
      - From `EchoInput` to `App`
      - Prop must be a callback function

React: Props

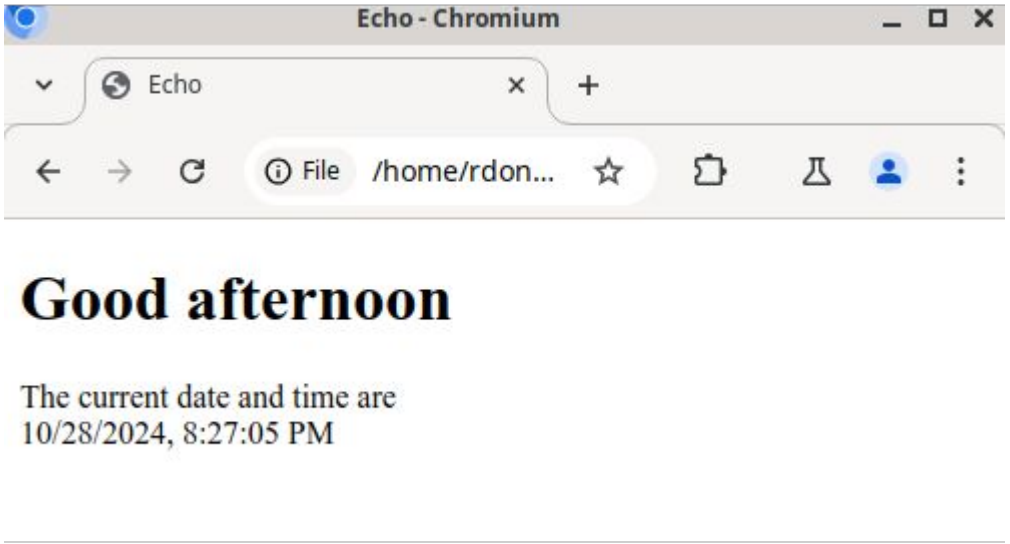
- **Question:**
  - How does a **parent** component send data to a **child** component?
- **Answer:**
  - Props
- **Question:**
  - How does a **child** component send data to its **parent** component?
- **Answer:**
  - Props that reference callback functions

Agenda

- React: concepts
- React: fundamentals
- React: useState hooks
- React: props
- **React: useEffect hooks**
- React: realistic example

React: useEffect Hooks

- “Datetime” example



React: useEffect Hooks

- See **datetime.html**
  - Things to note:
    - We've seen such code before

React: useEffect Hooks

- See **datetimereact.html**
  - Things to note:
    - ***useEffect hook***

`React.useEffect(f);`	Call `f()` at initial render and every subsequent render
`React.useEffect(f, [statevar1, statevar2, …]);`	Call `f()` at initial render and when `statevar1` or `statevar2`, or, … changes
`React.useEffect(f, []);`	Call `f()` at initial render

Agenda

- React: concepts
- React: fundamentals
- React: useState hooks
- React: props
- React: useEffect hooks
- **React: realistic example**

React: Realistic Example

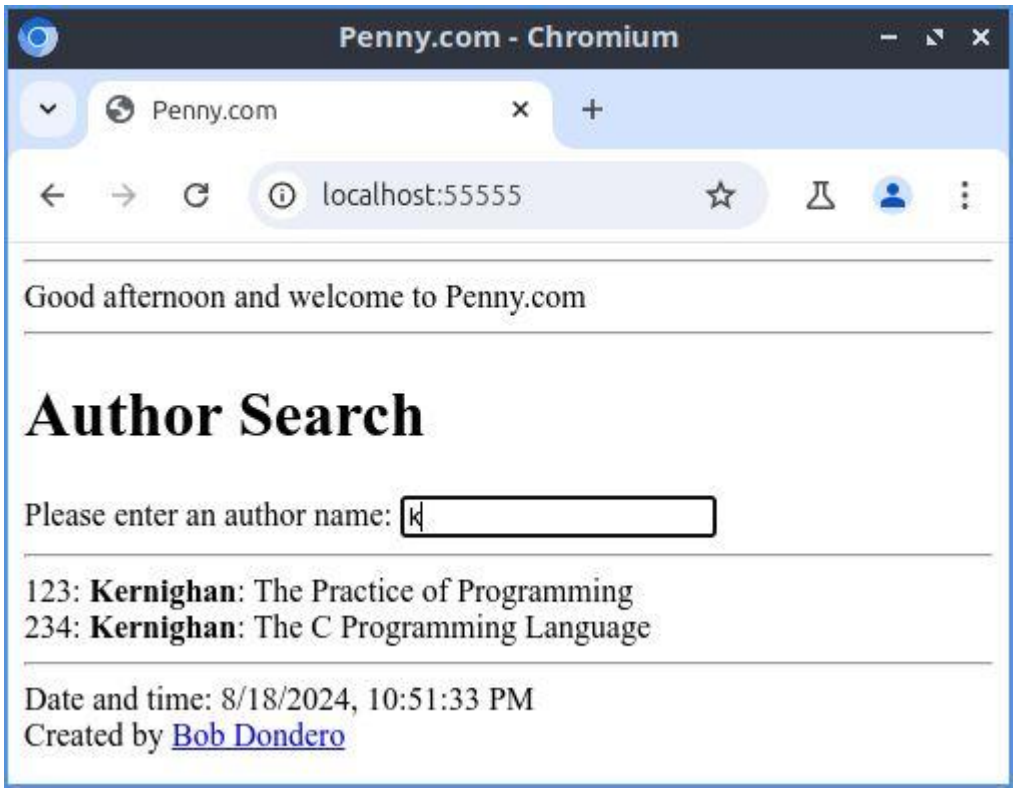
- See **PennyReact** app



Thanks, in part, to Liam Esparraguara ('24)

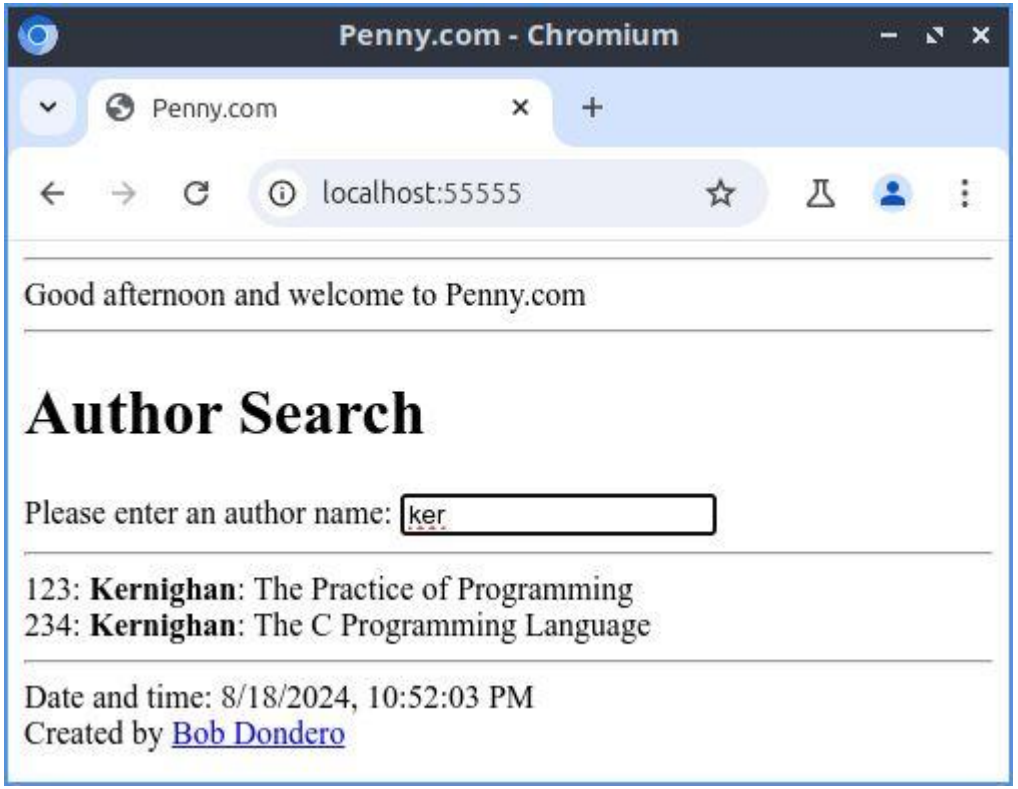
React: Realistic Example

- See **PennyReact** app (cont.)



React: Realistic Example

- See **PennyReact** app (cont.)



React: Realistic Example

- See **PennyReact** app (cont.)



React: Realistic Example

- See **PennyReact** app (cont.)
  - runserver.py
  - penny.sql, penny.sqlite
  - database.py
  - penny.py
  - **index.html**

Lecture Summary

- In this lecture we covered:
  - React: concepts
  - React: fundamentals
  - React: useState hooks
  - React: props
  - React: useEffect hooks
  - React: realistic example
- See also:
  - **Appendix 1**: Arrow functions

Appendix 1:
Arrow Functions

Arrow Functions

- Recall from JavaScript lectures…
- **Question**: How is `this` bound within a function `f()`?
- **Answer**: Depends upon how `f()` is called:

Function Call	Binding of `this`
`f()`	In `f()`, `this` is `undefined`
`o.f()`	In `f()`, `this` is bound to `o`
`new f()`	In `f()`, `this` is bound to a new empty object

Arrow Functions

- Some terms for this lecture:
  - **Ordinary function**: a non-arrow function
  - **Ordinary variable**: a non-`this` variable

Arrow Functions

- ***Arrow function def expressions***
  - Informally ***arrow functions***
  - Arrow functions vs ordinary functions:
    - More succinct
    - Same semantics - **mostly!!!**

Aside: setInterval & setTimeout

In browsers:

```
window.setInterval(f, ms);
// Call f every ms milliseconds
```

We have seen

```
window.setTimeout(f, ms);
// Call f after ms milliseconds
```

We have seen

In Node.js:

```
setInterval(f, ms);
// Call f every ms milliseconds
```

```
setTimeout(f, ms);
// Call f after ms milliseconds
```

We'll use now

Arrow Functions

- **Fact 1**: In an ordinary function…
  - The value of `this` is determined **dynamically**
    - Based upon the call
    - `o.f()`
      - In the function `this` is bound to `o`
    - `f()`
      - In the function `this` is undefined

Arrow Functions

```
$ node arrow1.js
undefined
$
```

- See **arrow1.js**
  - Notes:
    - Global code calls `main()`
    - `main()` calls `blueCar.writeColor()`
    - `blueCar.writeColor()` calls `setTimeout()`
    - `setTimeout()` calls given ordinary function
      - As `f()`, not as `o.f()`
    - In ordinary function, `this` is undefined

Arrow Functions

- **Fact 2**: In an ordinary function…
  - The value of an ordinary variable is determined **statically**
    - Based upon program block structure

Arrow Functions

```
$ node arrow2.js
blue
$
```

- See **arrow2.js**
  - Notes:
    - Global code calls `main()`
    - `main()` calls `blueCar.writeColor()`
    - `blueCar.writeColor()` calls `setTimeout()`
    - `setTimeout()` calls given ordinary function
      - As `f()`, not as `o.f()`
    - In ordinary function, `this` is undefined
      - But the ordinary function doesn't use `this`!

Arrow Functions

- **Fact 3**: In an arrow function…
  - The value of `this` (and any ordinary variable) is determined **statically**
    - Based upon program block structure

Arrow Functions

```
$ node arrow3.js
blue
$
```

- See **arrow3.js**
  - Notes:
    - Global code calls `main()`
    - `main()` calls `blueCar.writeColor()`
    - `blueCar.writeColor()` calls `setTimeout()`
    - `setTimeout()` calls given arrow function
      - As `f()`, not as `o.f()`
    - In arrow function, `this` is bound to `blueCar`

Arrow Functions

- **Question**: Why use arrow functions?
- **Answer 1**: They're often more succinct
- **Answer 2**: `this` is defined statically

- Arrow functions often are appropriate as callback functions