

# Client-Side Web Programming: JavaScript (Part 4)

Copyright © 2024 by  
Robert M. Dondero, Ph.D.  
Princeton University

# Objectives

- We will cover:
  - React concepts
  - React examples
  - jQuery vs. React

# Agenda

- **React concepts**
- React simple examples
- React examples
- jQuery vs. React

# 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**
    - (pro) Easier to understand?
    - (con) Semi-deprecated
  - **Functional**
    - (pro) More succinct
    - (con) Harder to understand?
- We'll cover functional...

# React Concepts

- Key concept: ***components***
  - Programmer defines components
  - Each component is defined as a function
  - Components can be arranged hierarchically
  - Each component:
    - Can have **state**
    - Can accept properties (**props**) from parent component
    - Returns a (partial) DOM tree

# 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 (typically) returns a (partial) DOM tree
    - React updates the virtual DOM tree with the component's (partial) DOM tree
    - 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 smallest possible browser DOM subtree

# React Concepts

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

# React Concepts

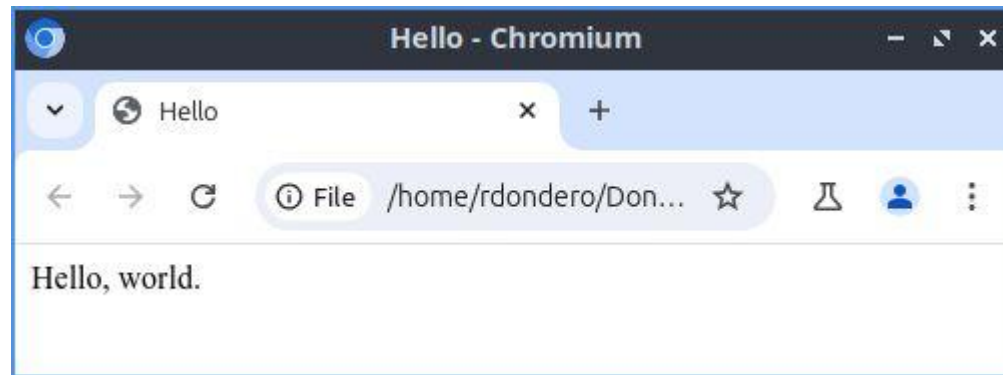
- Key concept: *React bundles*
  - Typically React programmers use **WebPack**, **Next.js**, **Vite**, ... to create a **React bundles**
  - Here, for simplicity, we will not create React bundles
    - But see next lecture

# Agenda

- React concepts
- **React simple examples**
- React examples
- jQuery vs. React

# React Simple Examples

- See [reacthello.html](#)



# React Simple Examples

- See **reacthello.html** (cont.)
  - Things to note:
    - Overall structure
    - (Minimal) use of HTML
    - Use of JSX
    - Defining a component
      - Which returns a partial DOM tree

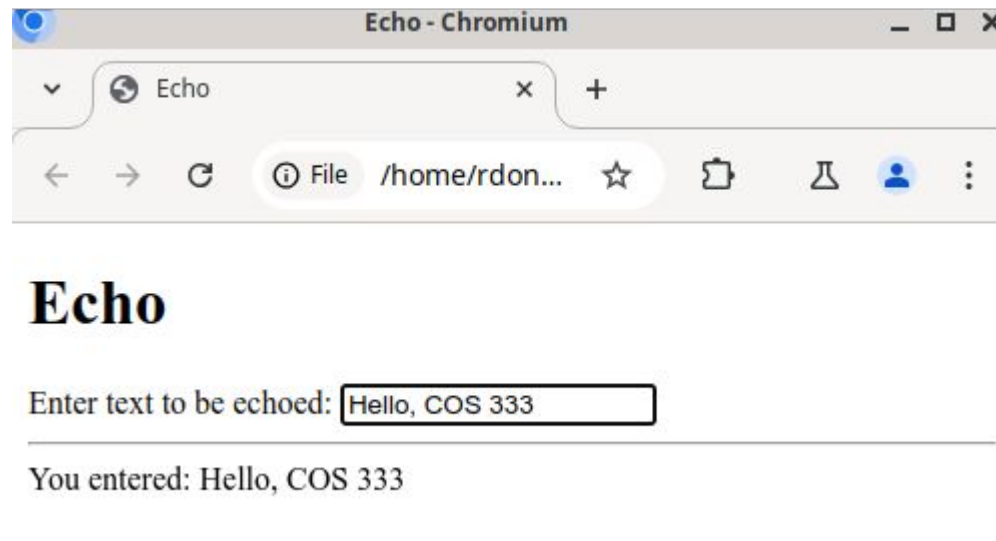
# React Simple Examples

- See **reacthello.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 Simple Examples

- See [reactecho1.html](#)





# React Simple Examples

- See [reactecho1.html](#) (cont.)
  - 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 (partial) DOM tree
      - Update virtual DOM tree with partial DOM tree
      - Compares new virtual DOM tree with old
      - Reconciles virtual DOM tree with browser DOM tree

# 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

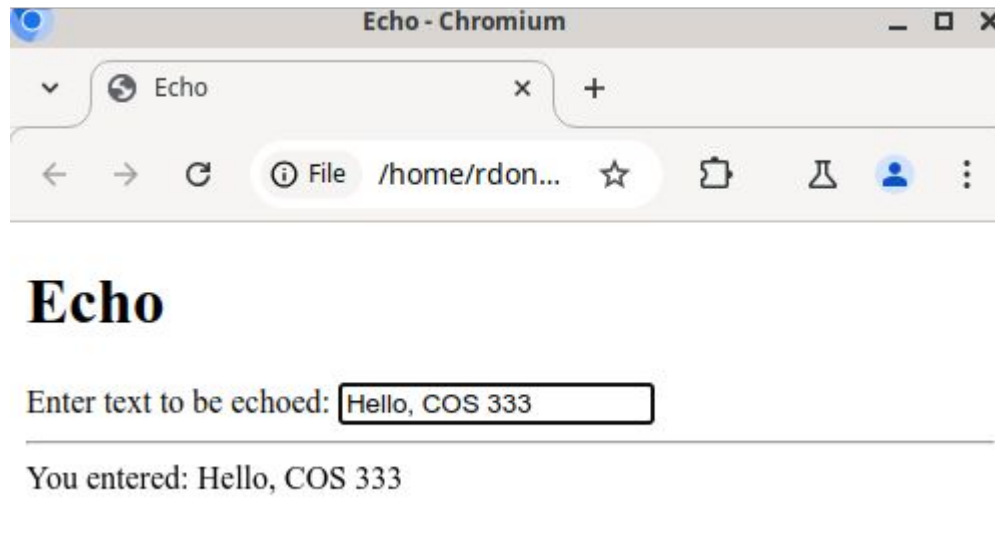
# Aside: Arrow Functions

- See [arrow.js](#)

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

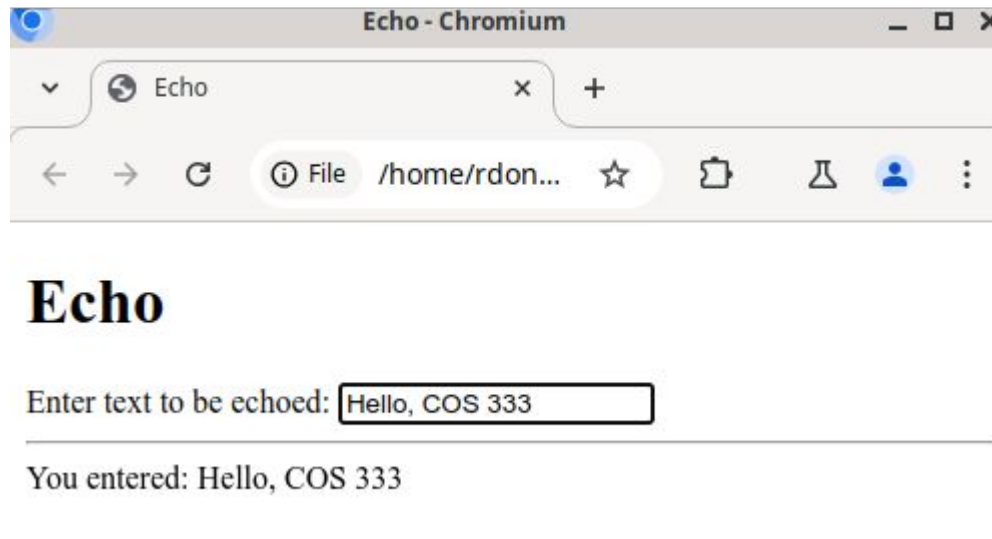
# React Simple Examples

- See [reactecho2.html](#) (cont.)
  - Things to note:
    - Uses arrow functions exclusively



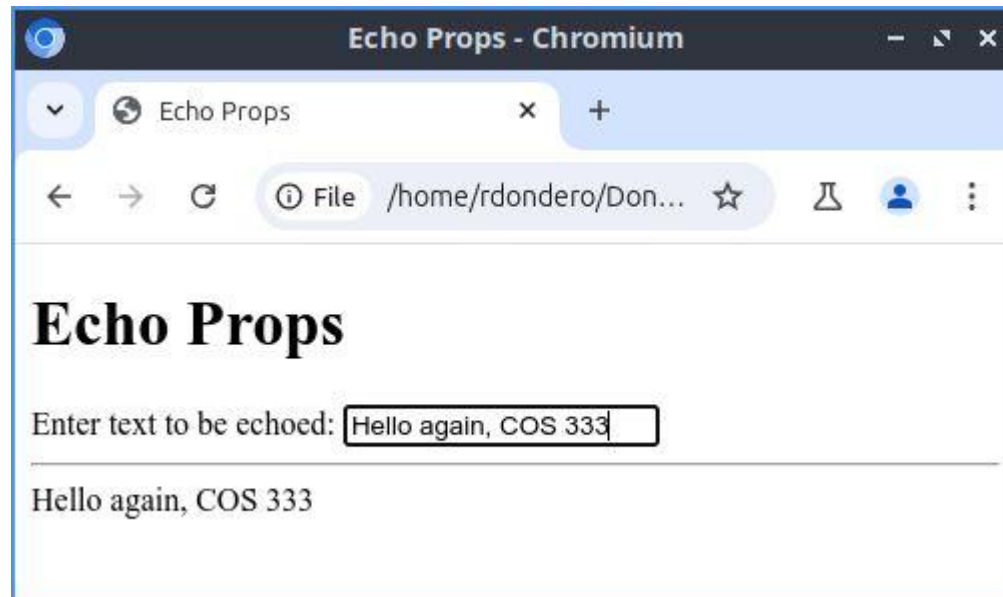
# React Simple Examples

- See [reactecho3.html](#) (cont.)
  - Things to note:
    - Uses arrow functions as callbacks



# React Simple Examples

- See [reactecho4.html](#)



# React Simple Examples

- See [reactecho4.html](#) (cont.)
  - Things to note:
    - Components arranged in a tree (hierarchy)
    - Use of a **prop** to send data **downward**
      - From parent component to child component
      - From `Echo` to `EchoOutput`
    - Use of a **prop** to send data **upward**
      - From child component to parent component
      - From `EchoInput` to `Echo`
      - Prop must be a callback function

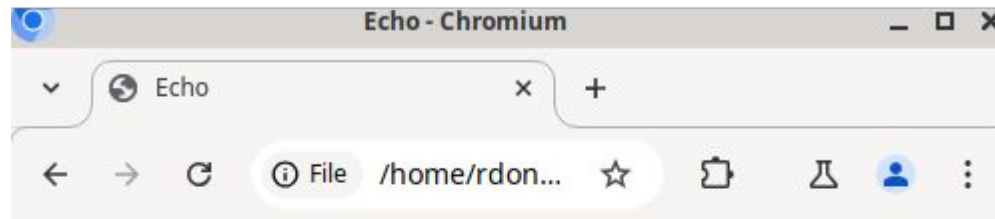
# React Simple Examples

- **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



# React Simple Examples

- See [reactdatetime.html](#)



**Good afternoon**

The current date and time are  
10/28/2024, 8:27:05 PM

---

# React Simple Examples

- See **reactdatetime.html** (cont.)
  - Things to note:
    - ***useEffect hook***
      - `React.useEffect (f) ;`
        - » Call `f ()` at initial render and every subsequent render
      - `React.useEffect (f, []) ;`
        - » Call `f ()` at initial render
      - `React.useEffect (f, [statevar]) ;`
        - » Call `f ()` at initial render and when `statevar` changes

# Agenda

- React concepts
- React simple examples
- **React examples**
- jQuery vs. React

# React Examples

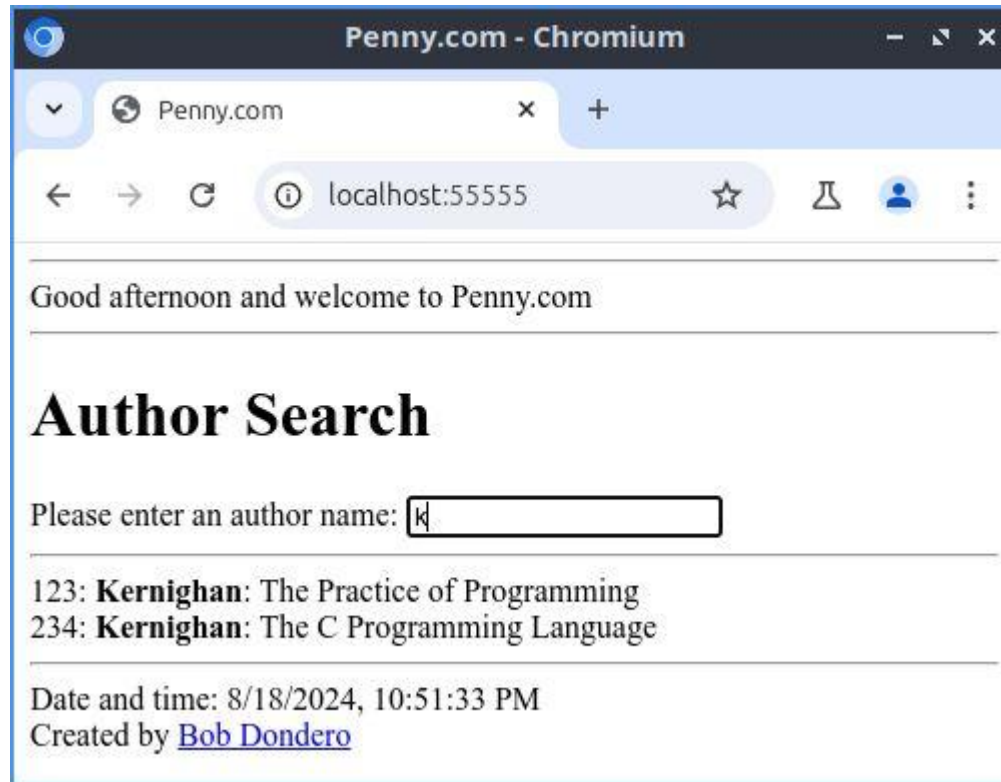
- See **PennyReact1** app



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

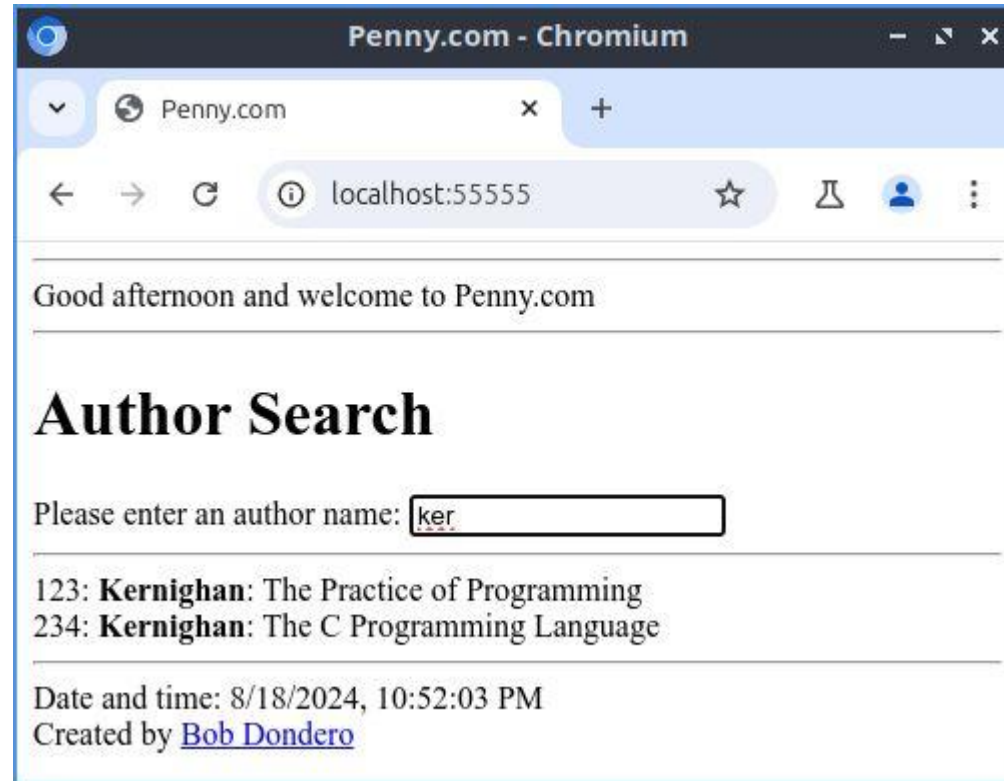
# React Examples

- See **PennyReact1** app (cont.)



# React Examples

- See **PennyReact1** app (cont.)



# React Examples

- See **PennyReact1** app (cont.)



# React Examples

- See **PennyReact1** app (cont.)
  - runserver.py
  - penny.sql, penny.sqlite
  - database.py
  - penny.py
  - **index.html**



# React Examples

- **Problem** (not really, but let's pretend)
  - The PennySearch function is too long
  - The PennySearch component is too complex
- **Solution**
  - Factor the PennySearch function into subordinate functions
  - Factor the PennySearch component into child components

# React Examples

- See **PennyReact2** app (cont.)
  - runserver.py
  - penny.sql, penny.sqlite
  - database.py
  - penny.py
  - **index.html**

# Agenda

- React concepts
- React simple examples
- React examples
- **jQuery vs. React**

# jQuery vs. React

- Repeating the fundamental idea...
- **jQuery**
  - HTML code contains JavaScript code
- **React**
  - HTML code is generated by JavaScript code

# jQuery vs. React

- **jQuery**

- HTML code contains JavaScript code
- Modularity by **technologies**

- **React**

- HTML code is generated by JavaScript code
- Modularity by **components**

# jQuery vs. React

- Commentary:
  - Is jQuery code more maintainable?
  - Is React code more reusable?

# Summary

- We have covered:
  - React concepts
  - React examples
  - jQuery vs. React
- 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 <code>this</code>
<code>f ()</code>	In <code>f ()</code> , <code>this</code> is undefined
<code>o.f ()</code>	In <code>f ()</code> , <code>this</code> is bound to <code>o</code>
<code>new f ()</code>	In <code>f ()</code> , <code>this</code> 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

- 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

```
$ node arrow1.js
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

- 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`!

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



# 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

- 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`

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

# 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