

# 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

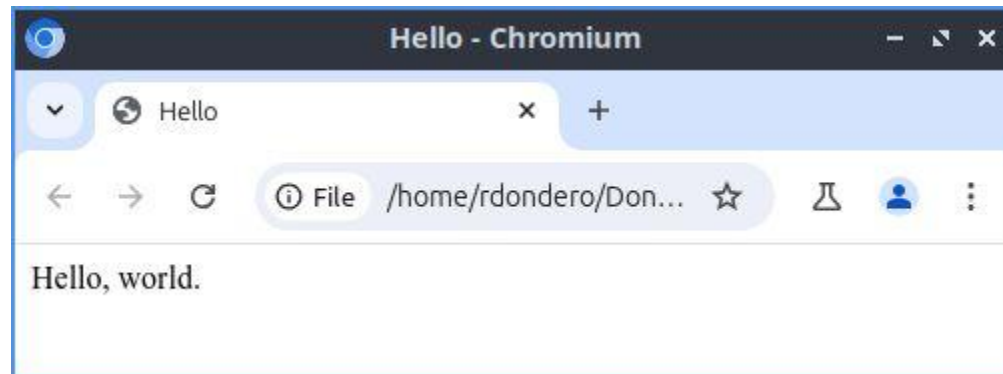
- Key concept: ***React bundles***
  - Typically React programmers use **WebPack**, **Next.js**, **Vite**, ... to create a **React bundles**
  - Here, for simplicity, we will not create React bundles
    - But see next lecture

# Agenda

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

# React Simple Examples

- See **reacthello.html**



## reacthello.html (Page 1 of 1)

```

1: <!DOCTYPE html>
2: <html>
3:   <head>
4:     <title>Hello</title>
5:   </head>
6:
7:   <body>
8:
9:     <div id="root"></div>
10:
11:     <script src=
12:       "https://cdn.jsdelivr.net/npm/react@18.3.1/umd/react.production
.min.js">
13:     </script>
14:
15:     <script src=
16:       "https://cdn.jsdelivr.net/npm/react-dom@18.3.1/umd/react-dom.pr
oduction.min.js">
17:     </script>
18:
19:     <script src=
20:       "https://cdn.jsdelivr.net/npm/babel-standalone@6.26.0/babel.min
.js">
21:     </script>
22:
23:     <script type="text/babel">
24:
25:       'use strict';
26:
27:       //-----
28:
29:       function App() {
30:         return (
31:           <div>
32:             Hello, world.
33:           </div>
34:         );
35:       }
36:
37:       //-----
38:
39:       let domRoot = document.getElementById('root');
40:       let reactRoot = ReactDOM.createRoot(domRoot);
41:       reactRoot.render(
42:         <React.StrictMode>
43:           <App />
44:         </React.StrictMode>
45:       );
46:
47:     </script>
48:   </body>
49: </html>
50:

```

## blank (Page 1 of 1)

1: This page is intentionally blank.

# 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

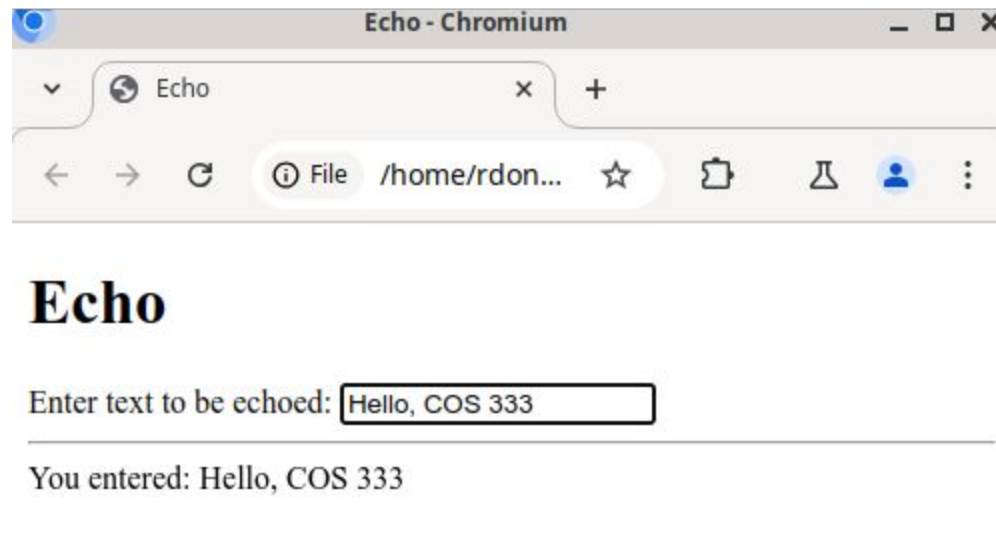
- See **reacthello.html** (cont.)
  - Things to note:
    - `React.StrictMode`
      - “Lets you find common bugs in your components early during development.”
      - “Use StrictMode to enable additional development behaviors and warnings for the component tree inside.”

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



# React Simple Examples

- See **reactecho1.html**



## reactecho1.html (Page 1 of 1)

```

1: <!DOCTYPE html>
2: <html>
3:   <head>
4:     <title>Echo</title>
5:   </head>
6:   <body>
7:
8:     <div id="root"></div>
9:
10:    <script src=
11:      "https://cdn.jsdelivr.net/npm/react@18.3.1/umd/react.production
.min.js">
12:    </script>
13:
14:    <script src=
15:      "https://cdn.jsdelivr.net/npm/react-dom@18.3.1/umd/react-dom.pr
oduction.min.js">
16:    </script>
17:
18:    <script src=
19:      "https://cdn.jsdelivr.net/npm/babel-standalone@6.26.0/babel.min
.js">
20:    </script>
21:
22:    <script type="text/babel">
23:
24:      'use strict';
25:
26:      //-----
27:
28:      function App() {
29:        const [enteredText, setEnteredText] = React.useState('');
30:
31:        return (
32:          <div>
33:            <h1>Echo</h1>
34:            Enter text to be echoed:&nbsp;
35:            <input
36:              type='text'
37:              onInput={ function (event) {
38:                setEnteredText(event.target.value);
39:              }}
40:              autoFocus
41:            />
42:            <hr />
43:            You entered: {enteredText}
44:          </div>
45:        );
46:      }
47:
48:      //-----
49:
50:      let domRoot = document.getElementById('root');
51:      let reactRoot = ReactDOM.createRoot(domRoot);
52:      reactRoot.render(
53:        <React.StrictMode>
54:        <App />
55:        </React.StrictMode>
56:      );
57:
58:    </script>
59:  </body>
60: </html>
61:

```

## arrow.js (Page 1 of 1)

```

1: //-----
2: // arrow.js
3: // Author: Bob Dondero
4: //-----
5:
6: 'use strict';
7:
8: function square1(i) {return i * i;}
9: let square2 = function(i) {return i * i;};
10: let square3 = (i) => {return i * i;};
11: let square4 = (i) => i * i;
12: let square5 = i => i * i;
13:
14: function prod1(i, j) {return i * j;}
15: let prod2 = function(i, j) {return i * j;};
16: let prod3 = (i, j) => {return i * j;};
17: let prod4 = (i, j) => i * j;
18:
19: function greet1() {process.stdout.write('hi\n');}
20: let greet2 = function() {process.stdout.write('hi\n');};
21: let greet3 = () => {process.stdout.write('hi\n');};
22: let greet4 = () => process.stdout.write('hi\n');
23:
24: function main() {
25:   process.stdout.write(String(square1(5)) + '\n');
26:   process.stdout.write(String(square2(5)) + '\n');
27:   process.stdout.write(String(square3(5)) + '\n');
28:   process.stdout.write(String(square4(5)) + '\n');
29:   process.stdout.write(String(square5(5)) + '\n');
30:
31:   process.stdout.write(String(prod1(5, 6)) + '\n');
32:   process.stdout.write(String(prod2(5, 6)) + '\n');
33:   process.stdout.write(String(prod3(5, 6)) + '\n');
34:   process.stdout.write(String(prod4(5, 6)) + '\n');
35:
36:   greet1();
37:   greet2();
38:   greet3();
39:   greet4();
40: }
41:
42: if (require.main === module)
43:   main();

```

# 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
$
```

## reactecho1.html (Page 1 of 1)

```

1: <!DOCTYPE html>
2: <html>
3:   <head>
4:     <title>Echo</title>
5:   </head>
6:   <body>
7:
8:
9:     <div id="root"></div>
10:
11:   <script src=
12:     "https://cdn.jsdelivr.net/npm/react@18.3.1/umd/react.production
.min.js">
13:   </script>
14:
15:   <script src=
16:     "https://cdn.jsdelivr.net/npm/react-dom@18.3.1/umd/react-dom.pr
oduction.min.js">
17:   </script>
18:
19:   <script src=
20:     "https://cdn.jsdelivr.net/npm/babel-standalone@6.26.0/babel.min
.js">
21:   </script>
22:
23:   <script type="text/babel">
24:
25:     'use strict';
26:
27:     //-----
28:
29:     function App() {
30:       const [enteredText, setEnteredText] = React.useState('');
31:
32:       return (
33:         <div>
34:           <h1>Echo</h1>
35:           Enter text to be echoed:&nbsp;
36:           <input
37:             type='text'
38:             onInput={ function (event) {
39:               setEnteredText(event.target.value);
40:             }}
41:             autoFocus
42:           />
43:           <hr />
44:           You entered: {enteredText}
45:         </div>
46:       );
47:     }
48:
49:     //-----
50:
51:     let domRoot = document.getElementById('root');
52:     let reactRoot = ReactDOM.createRoot(domRoot);
53:     reactRoot.render(
54:       <React.StrictMode>
55:         <App />
56:       </React.StrictMode>
57:     );
58:
59:   </script>
60: </body>
61: </html>

```

## arrow.js (Page 1 of 1)

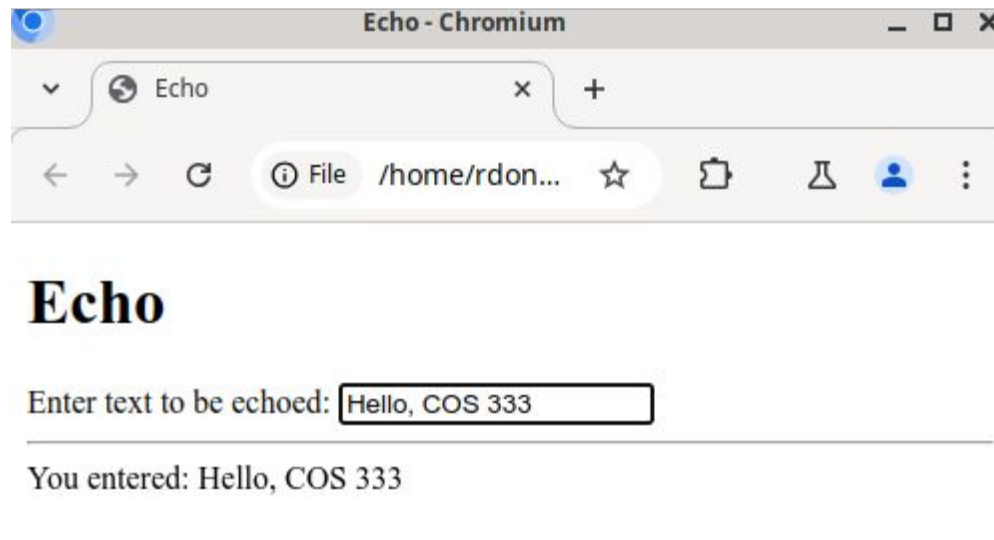
```

1: //-----
2: // arrow.js
3: // Author: Bob Dondero
4: //-----
5:
6: 'use strict';
7:
8: function square1(i) {return i * i;}
9: let square2 = function(i) {return i * i;};
10: let square3 = (i) => {return i * i;};
11: let square4 = (i) => i * i;
12: let square5 = i => i * i;
13:
14: function prod1(i, j) {return i * j;}
15: let prod2 = function(i, j) {return i * j;};
16: let prod3 = (i, j) => {return i * j;};
17: let prod4 = (i, j) => i * j;
18:
19: function greet1() {process.stdout.write('hi\n');}
20: let greet2 = function() {process.stdout.write('hi\n');};
21: let greet3 = () => {process.stdout.write('hi\n');};
22: let greet4 = () => process.stdout.write('hi\n');
23:
24: function main() {
25:   process.stdout.write(String(square1(5)) + '\n');
26:   process.stdout.write(String(square2(5)) + '\n');
27:   process.stdout.write(String(square3(5)) + '\n');
28:   process.stdout.write(String(square4(5)) + '\n');
29:   process.stdout.write(String(square5(5)) + '\n');
30:
31:   process.stdout.write(String(prod1(5, 6)) + '\n');
32:   process.stdout.write(String(prod2(5, 6)) + '\n');
33:   process.stdout.write(String(prod3(5, 6)) + '\n');
34:   process.stdout.write(String(prod4(5, 6)) + '\n');
35:
36:   greet1();
37:   greet2();
38:   greet3();
39:   greet4();
40: }
41:
42: if (require.main === module)
43:   main();

```

# React Simple Examples

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



## reactecho2.html (Page 1 of 1)

```

1: <!DOCTYPE html>
2: <html>
3:   <head>
4:     <title>Echo</title>
5:   </head>
6:   <body>
7:     <div id="root"></div>
8:
9:     <script src=
10:       "https://cdn.jsdelivr.net/npm/react@18.3.1/umd/react.production
.min.js">
11:     </script>
12:
13:     <script src=
14:       "https://cdn.jsdelivr.net/npm/react-dom@18.3.1/umd/react-dom.pr
oduction.min.js">
15:     </script>
16:
17:     <script src=
18:       "https://cdn.jsdelivr.net/npm/babel-standalone@6.26.0/babel.min
.js">
19:     </script>
20:
21:     <script type="text/babel">
22:
23:       'use strict';
24:
25:       //-----
26:
27:       const App = () => {
28:         const [enteredText, setEnteredText] = React.useState('');
29:
30:         return (
31:           <div>
32:             <h1>Echo</h1>
33:             Enter text to be echoed:&nbsp;
34:             <input
35:               type='text'
36:               onInput={ (event) => {
37:                 setEnteredText(event.target.value);
38:               }}
39:               autoFocus
40:             />
41:             <hr />
42:             You entered: {enteredText}
43:           </div>
44:         );
45:       };
46:
47:       //-----
48:
49:       let domRoot = document.getElementById('root');
50:       let reactRoot = ReactDOM.createRoot(domRoot);
51:       reactRoot.render(
52:         <React.StrictMode>
53:           <App />
54:         </React.StrictMode>
55:       );
56:
57:     </script>
58:   </body>
59: </html>

```

## reactecho3.html (Page 1 of 1)

```

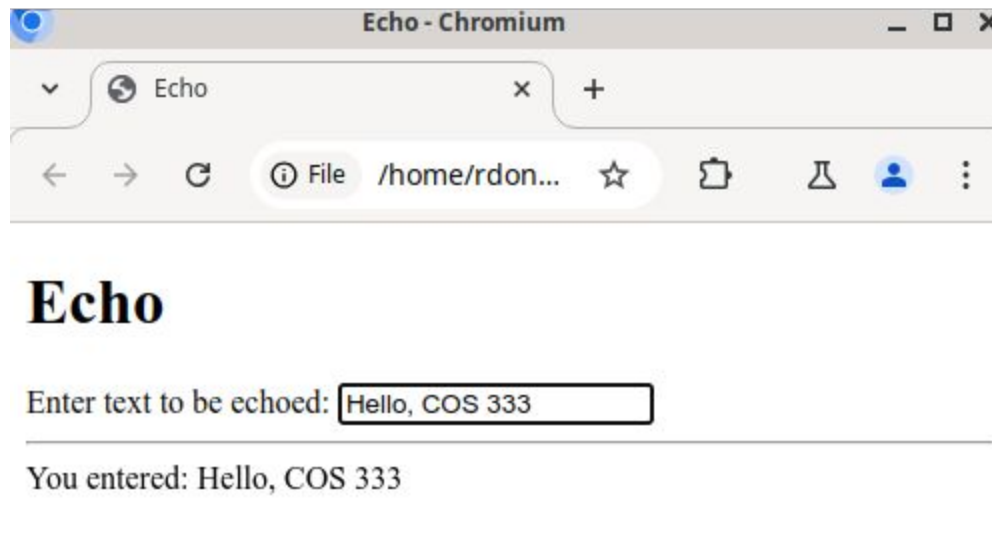
1: <!DOCTYPE html>
2: <html>
3:   <head>
4:     <title>Echo</title>
5:   </head>
6:   <body>
7:     <div id="root"></div>
8:
9:     <script src=
10:       "https://cdn.jsdelivr.net/npm/react@18.3.1/umd/react.production
.min.js">
11:     </script>
12:
13:     <script src=
14:       "https://cdn.jsdelivr.net/npm/react-dom@18.3.1/umd/react-dom.pr
oduction.min.js">
15:     </script>
16:
17:     <script src=
18:       "https://cdn.jsdelivr.net/npm/babel-standalone@6.26.0/babel.min
.js">
19:     </script>
20:
21:     <script type="text/babel">
22:
23:       'use strict';
24:
25:       //-----
26:
27:       function App() {
28:         const [enteredText, setEnteredText] = React.useState('');
29:
30:         return (
31:           <div>
32:             <h1>Echo</h1>
33:             Enter text to be echoed:&nbsp;
34:             <input
35:               type='text'
36:               onInput={ (event) => {
37:                 setEnteredText(event.target.value);
38:               }}
39:               autoFocus
40:             />
41:             <hr />
42:             You entered: {enteredText}
43:           </div>
44:         );
45:       };
46:
47:       //-----
48:
49:       let domRoot = document.getElementById('root');
50:       let reactRoot = ReactDOM.createRoot(domRoot);
51:       reactRoot.render(
52:         <React.StrictMode>
53:           <App />
54:         </React.StrictMode>
55:       );
56:
57:     </script>
58:   </body>
59: </html>

```



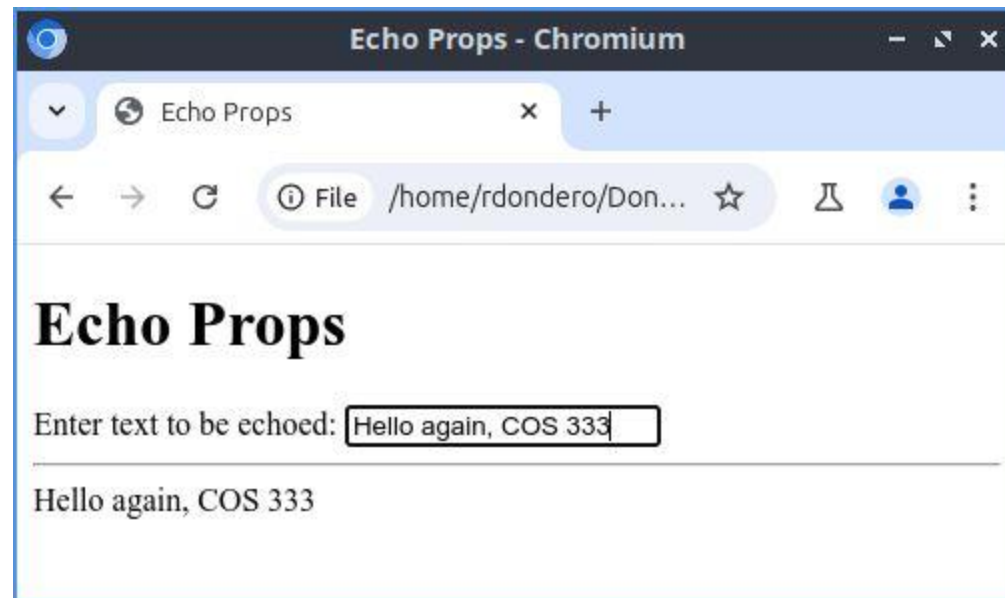
# React Simple Examples

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



# React Simple Examples

- See [reactecho4.html](#)



## reactecho4.html (Page 1 of 2)

```

1: <!DOCTYPE html>
2: <html>
3:   <head>
4:     <title>Echo Props</title>
5:   </head>
6:   <body>
7:     <div id="root"></div>
8:
9:     <script src=
10:       "https://cdn.jsdelivr.net/npm/react@18.3.1/umd/react.production
.min.js">
11:     </script>
12:
13:     <script src=
14:       "https://cdn.jsdelivr.net/npm/react-dom@18.3.1/umd/react-dom.pr
oduction.min.js">
15:     </script>
16:
17:     <script src=
18:       "https://cdn.jsdelivr.net/npm/babel-standalone@6.26.0/babel.min
.js">
19:     </script>
20:
21:     <script type="text/babel">
22:
23:       'use strict';
24:
25:       //-----
26:
27:       function EchoInput(props) {
28:         return (
29:           <div>
30:             <h1>Echo Props</h1>
31:             Enter text to be echoed:&nbsp;  
32:             <input
33:               type='text'
34:               onInput={ (event) => {
35:                 props.callback(event.target.value);
36:               }}
37:               autoFocus
38:             />
39:           </div>
40:         );
41:       };
42:     };
43:   }
44:
45:   //-----
46:
47:   function EchoOutput(props) {
48:     return (
49:       <div>
50:         {props.txt}
51:       </div>
52:     );
53:   }
54:
55:   //-----
56:
57:   function Echo() {
58:     const [enteredText, setEnteredText] = React.useState('');
59:
60:     return (
61:       <div>
62:         <EchoInput callback={setEnteredText} />

```

## reactecho4.html (Page 2 of 2)

```

63:         <hr />
64:         <EchoOutput txt={enteredText} />
65:       </div>
66:     );
67:   }
68:
69:   //-----
70:
71:   function App() {
72:     return (
73:       <Echo />
74:     );
75:   }
76:
77:   //-----
78:
79:   let domRoot = document.getElementById('root');
80:   let reactRoot = ReactDOM.createRoot(domRoot);
81:   reactRoot.render(
82:     <React.StrictMode>
83:       <App />
84:     </React.StrictMode>
85:   );
86:
87: </script>
88: </body>
89: </html>
90:

```

# React Simple Examples

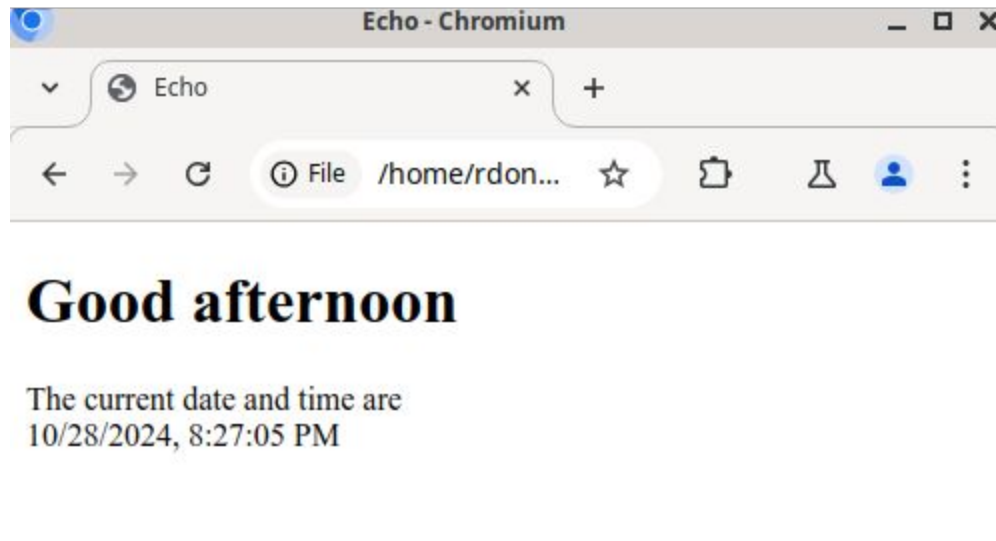
- See [reactecho4.html](#) (cont.)
  - Things to note:
    - Components arranged in a tree (hierarchy)
    - Use of a **prop** to send data **downward**
      - From parent component to child component
      - From `Echo` to `EchoOutput`
    - Use of a **prop** to send data **upward**
      - From child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



## reactdatetime.html (Page 1 of 2)

```

1: <!DOCTYPE html>
2: <html>
3:   <head>
4:     <title>Echo</title>
5:   </head>
6:
7:   <body>
8:
9:     <div id="root"></div>
10:
11:     <script src=
12:       "https://cdn.jsdelivr.net/npm/react@18.3.1/umd/react.production
.min.js">
13:     </script>
14:
15:     <script src=
16:       "https://cdn.jsdelivr.net/npm/react-dom@18.3.1/umd/react-dom.pr
oduction.min.js">
17:     </script>
18:
19:     <script src=
20:       "https://cdn.jsdelivr.net/npm/babel-standalone@6.26.0/babel.min
.js">
21:     </script>
22:
23:     <script type="text/babel">
24:
25:       'use strict';
26:
27:       //-----
28:
29:       function App() {
30:         const [datetime, setDatetime] = React.useState(new Date());
31:
32:         function updateDatetime() {
33:           window.setInterval(
34:             () => {setDatetime(new Date());},
35:             1000
36:           );
37:         }
38:
39:         React.useEffect(updateDatetime, []);
40:
41:         let hours = datetime.getHours();
42:         let ampm = (hours < 12) ? 'morning' : 'afternoon';
43:         return (
44:           <div>
45:             <h1>Good {ampm}</h1>
46:             The current date and time are<br/>
47:             {datetime.toLocaleString()}
48:           </div>
49:         );
50:       }
51:
52:       //-----
53:
54:       let domRoot = document.getElementById('root');
55:       let reactRoot = ReactDOM.createRoot(domRoot);
56:       reactRoot.render(
57:         <React.StrictMode>
58:           <App />
59:         </React.StrictMode>
60:       );
61:
62:     </script>

```

## reactdatetime.html (Page 2 of 2)

```

63:   </body>
64: </html>
65:

```

# 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

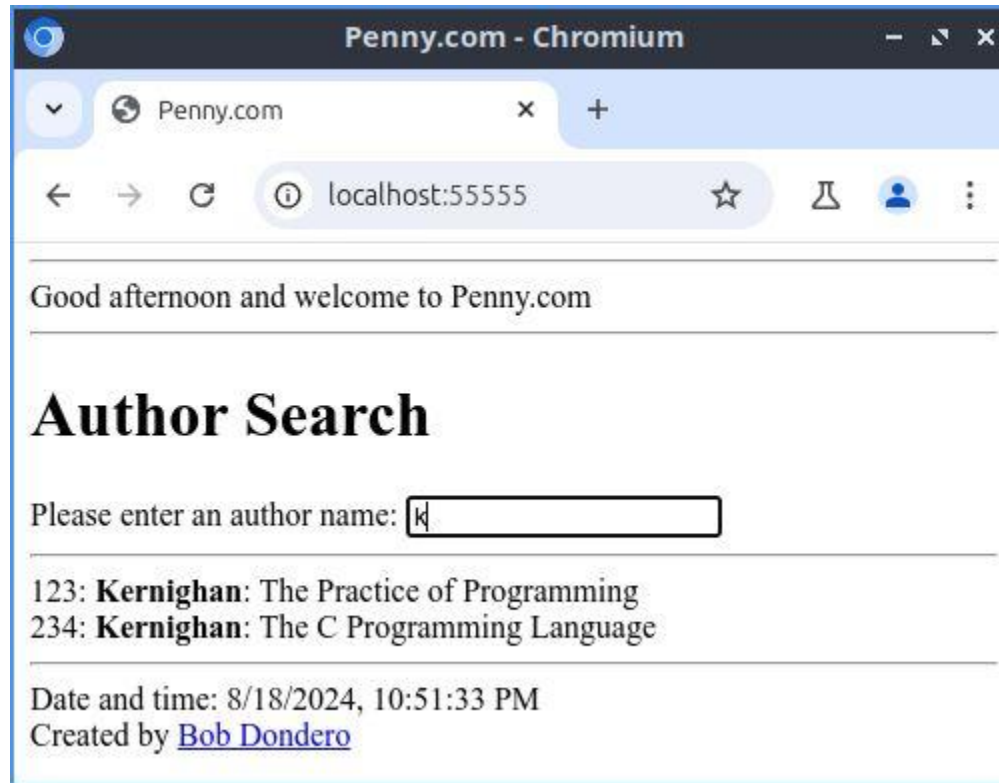
- See **PennyReact1** app



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

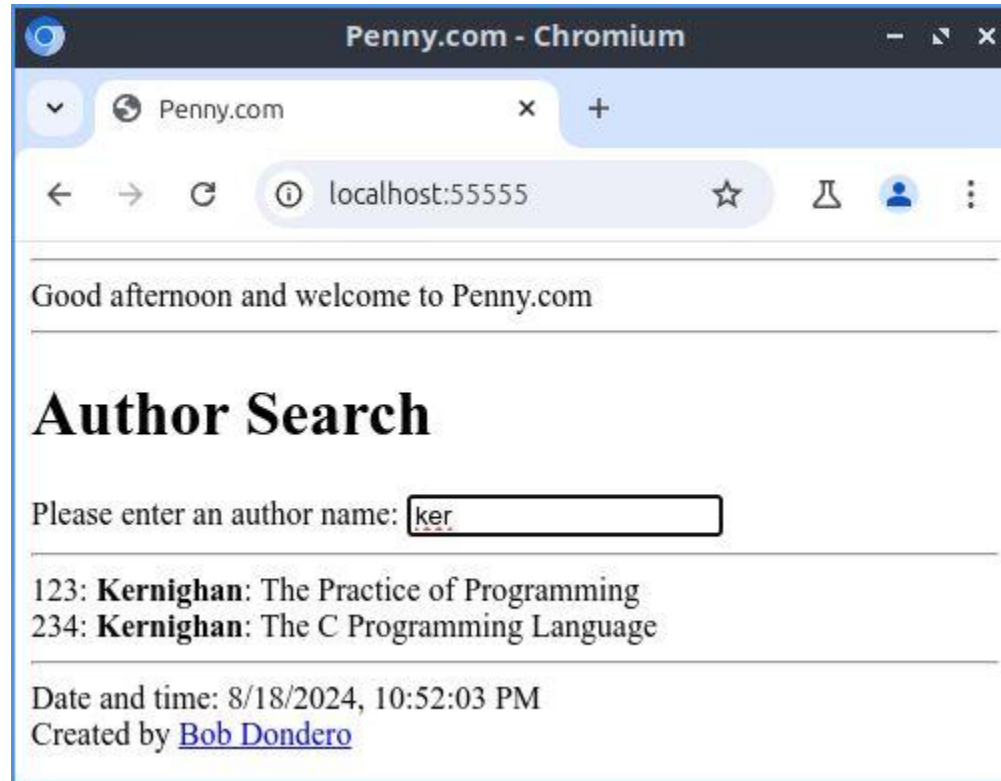
# React Examples

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



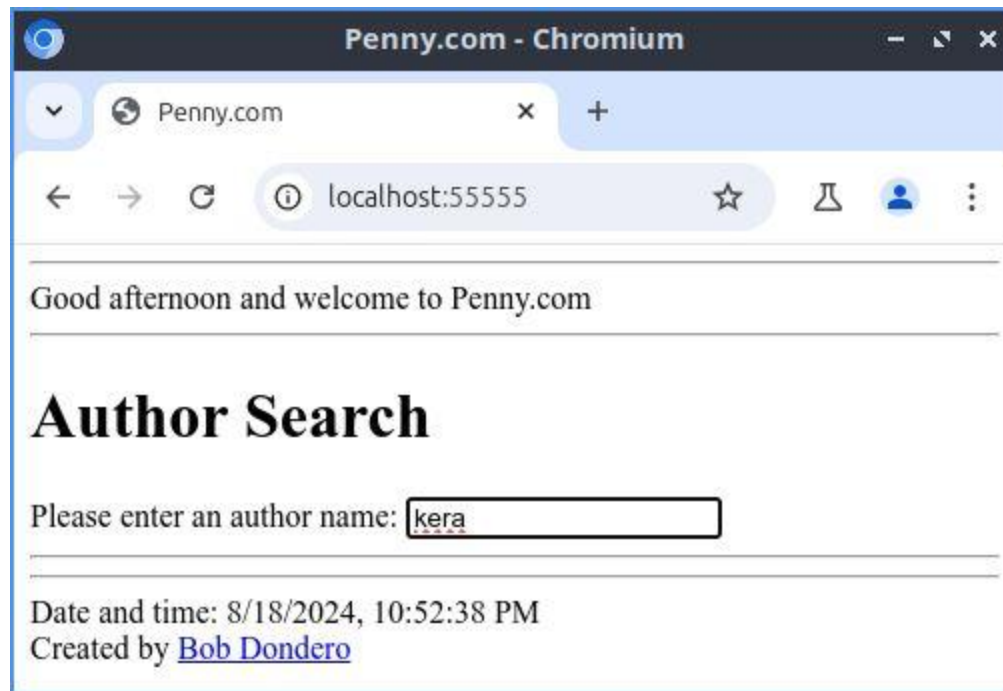
# React Examples

- 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**

## PennyReact1/index.html (Page 1 of 3)

```

1: <!DOCTYPE html>
2: <html>
3:   <head>
4:     <title>Penny.com</title>
5:   </head>
6:   <body>
7:     <div id="root"></div>
8:
9:     <script src=
10:       "https://cdn.jsdelivr.net/npm/react@18.3.1/umd/react.production
11:       .min.js">
12:     </script>
13:
14:     <script src=
15:       "https://cdn.jsdelivr.net/npm/react-dom@18.3.1/umd/react-dom.pr
16:       oduction.min.js">
17:     </script>
18:
19:     <script src=
20:       "https://cdn.jsdelivr.net/npm/babel-standalone@6.26.0/babel.min
21:       .js">
22:     </script>
23:
24:     <script type="text/babel">
25:
26:       'use strict';
27:
28:       //-----
29:
30:       function PennyHeader() {
31:         const [datetime, setDatetime] = React.useState(new Date());
32:
33:         function updateHeader() {
34:           window.setInterval(
35:             () => {setDatetime(new Date());},
36:             1000
37:           );
38:         }
39:
40:         React.useEffect(updateHeader, []);
41:
42:         let hours = datetime.getHours();
43:         let ampm = (hours < 12) ? 'morning' : 'afternoon';
44:         return (
45:           <div>
46:             <hr />
47:             Good {ampm} and welcome to Penny.com
48:             <hr />
49:           </div>
50:         );
51:       }
52:
53:       //-----
54:
55:       function PennySearch() {
56:         const [author, setAuthor] = React.useState('');
57:         const [books, setBooks] = React.useState([]);
58:
59:         function fetchBooks() {
60:           function handleResponse() {
61:             if (this.status !== 200) {
62:               alert('Error: Failed to fetch data from server');

```

## PennyReact1/index.html (Page 2 of 3)

```

63:           }
64:           let books = JSON.parse(this.response);
65:           setBooks(books);
66:         }
67:
68:         function handleError(request) {
69:           if (request.statusText !== 'abort')
70:             alert('Error: Failed to fetch data from server');
71:         }
72:
73:         let encodedAuthor = encodeURIComponent(author);
74:         let url = '/searchresults?author=' + encodedAuthor;
75:         let request = new XMLHttpRequest();
76:         request.onload = handleResponse;
77:         request.onerror = handleError;
78:         request.open('GET', url);
79:         request.send();
80:         return () => {request.abort();}
81:       }
82:
83:       React.useEffect(fetchBooks, [author]);
84:
85:       let timer = null;
86:
87:       function debouncedSetAuthor(author) {
88:         clearTimeout(timer);
89:         timer = setTimeout(()=>{setAuthor(author);}, 500);
90:       }
91:
92:       return (
93:         <div>
94:           <h1>Author Search</h1>
95:           Please enter an author name:&nbsp;
96:           <input
97:             type='text'
98:             onInput={ (event) => {
99:               debouncedSetAuthor(event.target.value);
100:             }}
101:             autoFocus
102:           />
103:           <hr />
104:           {books.map((book) => (
105:             <div key={book.isbn}>
106:               {book.isbn}:&nbsp;
107:               <strong>{book.author}</strong>:&nbsp;
108:               {book.title}
109:               <br />
110:             </div>
111:           ))}
112:         </div>
113:       );
114:     }
115:
116:     //-----
117:
118:     function PennyFooter() {
119:       const [datetime, setDatetime] = React.useState(new Date());
120:
121:       function updateFooter() {
122:         window.setInterval(
123:           () => {setDatetime(new Date());},
124:           1000
125:         );
126:       }
127:

```

## PennyReact1/index.html (Page 3 of 3)

```

128:     React.useEffect(updateFooter, []);
129:
130:     return (
131:       <div>
132:         <hr />
133:         Date and time: {datetime.toLocaleString()}
134:         <br />
135:         Created by&nbsp;
136:         <a href="https://www.cs.princeton.edu/~rdondero">
137:           Bob Dondero</a>
138:       </div>
139:     );
140:   }
141:
142:   //-----
143:
144:   function App() {
145:     return (
146:       <div>
147:         <PennyHeader />
148:         <PennySearch />
149:         <PennyFooter />
150:       </div>
151:     );
152:   }
153:
154:   //-----
155:
156:   let domRoot = document.getElementById('root');
157:   let reactRoot = ReactDOM.createRoot(domRoot);
158:   reactRoot.render(
159:     <React.StrictMode>
160:       <App />
161:     </React.StrictMode>
162:   );
163:
164: </script>
165: </body>
166: </html>
167:

```

## blank (Page 1 of 1)

1: This page is intentionally blank.



# 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**

## PennyReact2/index.html (Page 1 of 3)

```

1: <!DOCTYPE html>
2: <html>
3:   <head>
4:     <title>Penny.com</title>
5:   </head>
6:   <body>
7:
8:     <div id="root"></div>
9:
10:
11:   <script src=
12:     "https://cdn.jsdelivr.net/npm/react@18.3.1/umd/react.production
.min.js">
13:   </script>
14:
15:   <script src=
16:     "https://cdn.jsdelivr.net/npm/react-dom@18.3.1/umd/react-dom.pr
oduction.min.js">
17:   </script>
18:
19:   <script src=
20:     "https://cdn.jsdelivr.net/npm/babel-standalone@6.26.0/babel.min
.js">
21:   </script>
22:
23:   <script type="text/babel">
24:
25:     'use strict';
26:
27:     //-----
28:
29:     function PennyHeader() {
30:       const [datetime, setDatetime] = React.useState(new Date());
31:
32:       function updateHeader() {
33:         window.setInterval(
34:           () => {setDatetime(new Date());},
35:           1000
36:         );
37:       }
38:
39:       React.useEffect(updateHeader, []);
40:
41:       let hours = datetime.getHours();
42:       let ampm = (hours < 12) ? 'morning' : 'afternoon';
43:       return (
44:         <div>
45:           <hr />
46:           Good {ampm} and welcome to Penny.com
47:           <hr />
48:         </div>
49:       );
50:     }
51:
52:     //-----
53:
54:     function PennyInput(props) {
55:       return (
56:         <div>
57:           <h1>Author Search</h1>
58:           Please enter an author name:&nbsp;  
59:           <input
60:             type='text'
61:             onInput={ (event) => {
62:               props.callback(event.target.value);

```

## PennyReact2/index.html (Page 2 of 3)

```

63:           }}
64:           autoFocus
65:         />
66:       <hr />
67:     </div>
68:   );
69: }
70:
71: //-----
72:
73: function PennyOutput(props) {
74:   return (
75:     <div>
76:       {props.books.map((book) => (
77:         <div key={book.isbn}>
78:           {book.isbn}:&nbsp;  
79:           <strong>{book.author}</strong>:&nbsp;  
80:           {book.title}
81:           <br />
82:         </div>
83:       ))}
84:     </div>
85:   );
86: }
87:
88: //-----
89:
90: function PennySearch() {
91:   const [author, setAuthor] = React.useState('');
92:   const [books, setBooks] = React.useState([]);
93:
94:   function fetchBooks() {
95:     function handleResponse() {
96:       if (this.status !== 200) {
97:         alert('Error: Failed to fetch data from server');
98:         return;
99:       }
100:      let books = JSON.parse(this.response);
101:      setBooks(books);
102:    }
103:
104:    function handleError(request) {
105:      if (request.statusText !== 'abort')
106:        alert('Error: Failed to fetch data from server');
107:    }
108:
109:    let encodedAuthor = encodeURIComponent(author);
110:    let url = '/searchresults?author=' + encodedAuthor;
111:    let request = new XMLHttpRequest();
112:    request.onload = handleResponse;
113:    request.onerror = handleError;
114:    request.open('GET', url);
115:    request.send();
116:    return () => {request.abort();}
117:  }
118:
119:  React.useEffect(fetchBooks, [author]);
120:
121:  let timer = null;
122:
123:  function debouncedSetAuthor(author) {
124:    clearTimeout(timer);
125:    timer = setTimeout(()=>{setAuthor(author);}, 500);
126:  }
127:

```

## PennyReact2/index.html (Page 3 of 3)

```

128:         return (
129:             <div>
130:                 <PennyInput callback={debouncedSetAuthor} />
131:                 <PennyOutput books={books} />
132:             </div>
133:         );
134:     }
135:
136:     //-----
137:
138:     function PennyFooter() {
139:         const [datetime, setDatetime] = React.useState(new Date());
140:
141:         function updateFooter() {
142:             window.setInterval(
143:                 () => {setDatetime(new Date());},
144:                 1000
145:             );
146:         }
147:
148:         React.useEffect(updateFooter, []);
149:
150:         return (
151:             <div>
152:                 <hr />
153:                 Date and time: {datetime.toLocaleString()}
154:                 <br />
155:                 Created by&nbsp;
156:                 <a href="https://www.cs.princeton.edu/~rdondero">
157:                     Bob Dondero</a>
158:             </div>
159:         );
160:     }
161:
162:     //-----
163:
164:     function App() {
165:         return (
166:             <div>
167:                 <PennyHeader />
168:                 <PennySearch />
169:                 <PennyFooter />
170:             </div>
171:         );
172:     }
173:
174:     //-----
175:
176:     let domRoot = document.getElementById('root');
177:     let reactRoot = ReactDOM.createRoot(domRoot);
178:     reactRoot.render(
179:         <React.StrictMode>
180:             <App />
181:         </React.StrictMode>
182:     );
183:
184: </script>
185: </body>
186: </html>
187:

```

## blank (Page 1 of 1)

1: This page is intentionally blank.

# 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
$
```



## arrow1.js (Page 1 of 1)

```

1: //-----
2: // arrow1.js
3: // Author: Bob Dondero
4: //-----
5:
6: 'use strict';
7:
8: class Car {
9:
10:   constructor(color) {
11:     this._color = color;
12:   }
13:
14:   writeColor() {
15:     // Error: this is undefined.
16:     setTimeout(
17:       function () { process.stdout.write(this._color + '\n'); },
18:       2000
19:     );
20:   }
21: }
22:
23: function main() {
24:   let blueCar = new Car('blue');
25:   blueCar.writeColor();
26: }
27:
28: if (require.main === module)
29:   main();

```

## arrow2.js (Page 1 of 1)

```

1: //-----
2: // arrow2.js
3: // Author: Bob Dondero
4: //-----
5:
6: 'use strict';
7:
8: class Car {
9:
10:   constructor(color) {
11:     this._color = color;
12:   }
13:
14:   writeColor() {
15:     let self = this;
16:     setTimeout(
17:       function () { process.stdout.write(self._color + '\n'); },
18:       2000
19:     );
20:   }
21: }
22:
23: function main()
24: {
25:   let blueCar = new Car('blue');
26:   blueCar.writeColor();
27: }
28:
29: if (require.main === module)
30:   main();

```

# 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  
$
```

## arrow1.js (Page 1 of 1)

```

1: //-----
2: // arrow1.js
3: // Author: Bob Dondero
4: //-----
5:
6: 'use strict';
7:
8: class Car {
9:
10:   constructor(color) {
11:     this._color = color;
12:   }
13:
14:   writeColor() {
15:     // Error: this is undefined.
16:     setTimeout(
17:       function () { process.stdout.write(this._color + '\n'); },
18:       2000
19:     );
20:   }
21: }
22:
23: function main() {
24:   let blueCar = new Car('blue');
25:   blueCar.writeColor();
26: }
27:
28: if (require.main === module)
29:   main();

```

## arrow2.js (Page 1 of 1)

```

1: //-----
2: // arrow2.js
3: // Author: Bob Dondero
4: //-----
5:
6: 'use strict';
7:
8: class Car {
9:
10:   constructor(color) {
11:     this._color = color;
12:   }
13:
14:   writeColor() {
15:     let self = this;
16:     setTimeout(
17:       function () { process.stdout.write(self._color + '\n'); },
18:       2000
19:     );
20:   }
21: }
22:
23: function main()
24: {
25:   let blueCar = new Car('blue');
26:   blueCar.writeColor();
27: }
28:
29: if (require.main === module)
30:   main();

```

# 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  
$
```

**arrow3.js (Page 1 of 1)**

```
1: //-----
2: // arrow3.js
3: // Author: Bob Dondero
4: //-----
5:
6: 'use strict';
7:
8: class Car {
9:
10:   constructor(color) {
11:     this._color = color;
12:   }
13:
14:   writeColor() {
15:     setTimeout(
16:       () => { process.stdout.write(this._color + '\n'); },
17:       2000
18:     );
19:   }
20: }
21:
22: function main() {
23:   let blueCar = new Car('blue');
24:   blueCar.writeColor();
25: }
26:
27: if (require.main === module)
28:   main();
```

# 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