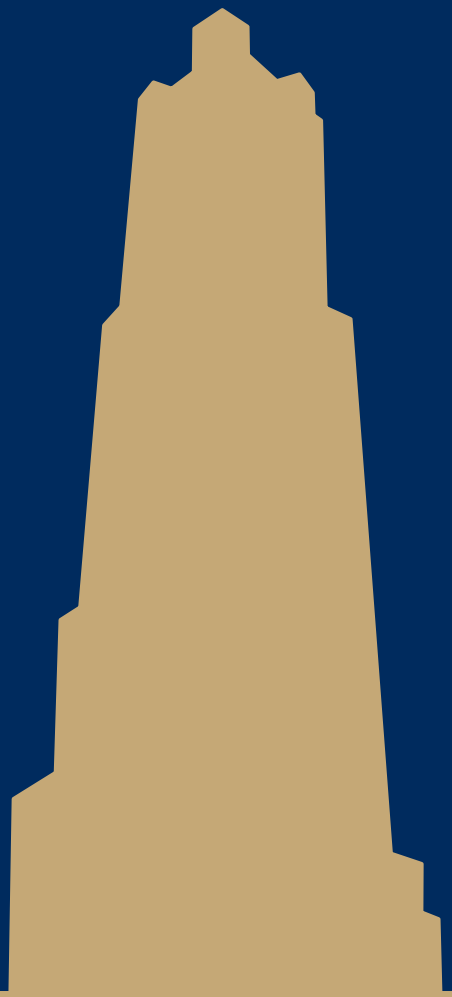
CS/COE 1520

pitt.edu/~ach54/cs1520

React



A terrible introduction to React:

- ???

```
class Square extends React.Component {
    render() {
        return (
            <button className="square" onClick={
                        function() { alert('click'); }
                    }>
                {this.props.value}
            </button>
        );
    }
}
```

???

Creating new elements

- Using DOM manipulation:
  - ```
    var element =  document.createElement('h1');
    element.setAttribute('class', 'greeting');
    var tn = document.createTextNode('Hello, world!');
    element.appendChild(tn);
    ```
- Using React:
  - ```
    const element = React.createElement('h1',
       {className: 'greeting'},
       'Hello, world!'
    );
    ```

Well, not exactly...

- React.createElement() returns an object something like this:
  - ```
    { type: 'h1',
      props: { className: 'greeting',
        children: 'Hello, world!'
      }
    }
    ```
- Cannot be attached directly into the DOM hierarchy
  - Must be rendered using the ReactDOM
    ```

ReactDOM

- A "virtual DOM"
- Keeps a cache of data structures representing what the rendered page should look like
- Compute differences with the displayed DOM, and issues the minimal amount of actual DOM manipulation calls to reflect the requested changes

Using the ReactDOM

- ```
  ReactDOM.render(
      element,
      document.getElementById('targetDiv')
  );
  ```
  ```

JSX

- `element = <h1 className="greeting">Hello, world!</h1>;`
- Syntax extensions to JavaScript
  - "Compiled" into `React.createElement()` calls
    - Hence, evaluate to JS objects, and can be used as such
- Can also embed JS expressions within JSX via {}:
  -
```
const element = (
    <h1>
        Hello, {getName() + "!"}
    </h1>;
);
```

JSX properties

- As we saw, can use string literals as params with ""
- Can also use JS expressions as params, wrapped with {}, do not wrap the {} with "". E.g.:
  - `element = <h1 className={getGreet()}>Hello, world!</h1>;`
- React DOM uses camelCase property naming convention instead of HTML attribute names
  - `class` is instead `className`
  - `tabindex` is `tabIndex`

"Compiling" JSX

- Add this line to your page:
  - `<script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>`
- Can now use JSX in future invoked scripts by setting the type attribute to type="text/babel"
  - Will use Babel to transpile JSX into compatible JavaScript!

Rendering JSX elements

- Can use `ReactDom.render()` just like in the earlier example
- How do you update JSX elements?
  - You can't!
    - They are *immutable*
  - You can, however, generate a new element and render that in place of the old one
    - E.g., by using the same `targetDiv` as the render root

Components

- ```
  function Welcome(props) {
  
  return <h1>Hello, {props.who}</h1>;
  
  }
  ```
- ```
  const element = <Welcome who="world" />;
  
  ReactDOM.render(
  
  element,
  
  document.getElementById('root')
  
  );
  ```

Encapsulation

- Think back to the proposed tutorial:
  - Building a tic-tac-toe board
- Need to keep track of whether or not a square on the board has been clicked in order to render it
  - How can we maintain this state?
    - Define our Component using classes, not functions

But JavaScript doesn't have classes!

- Well, it didn't…
  - Until ES6

ES6 classes

```
class Person {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }
    display() {
        console.log("Name: " + this.name);
        console.log("Age: " + this.age + "\n");
    }
}
```

ES6 class notes

- Classes cannot be instantiated before their definition
  - I.e.., class definitions are not "hoisted"
- Class methods are not constructible
  - Cannot be used on the right of a new
- Can have generators as methods
  - Using *method_name()
- Class bodies are evaluated in strict mode
- All instance attributes must be defined in method bodies

Class properties/methods

- Class properties must be set outside of the class body:
  - `Person.species = "Homo sapiens";`
- The `static` keyword can be used to define class methods

Inheritance

```
class Student extends Person {
    constructor(name, age) {
        super(name, age);
        this.classes = [];
    }
    add_class(new) {
        this.classes.push(new);
    }
    display() {
        super.display()
        console.log("Classes: " + this.classes + "\n");
    }
}
```

Getter/setter methods

```
class Rectangle {
   constructor(height, width) {
      this.height = height;
      this.width = width;
   }
   get area() {
      return this.calcArea();
   }
   calcArea() {
      return this.height * this.width;
   }
}
const square = new Rectangle(10, 10);
console.log(square.area);  // 100
```

Back to React...

- Could also have defined this:
  - ```
    function Welcome(props) {
        return <h1>Hello, {props.who}</h1>;
    }
    ```
- As:
  - ```
    class Welcome extends React.Component {
        render() {
            return <h1>Hello, {this.props.who}</h1>;
        }
    }
    ```

But now we can use the object to encapsulate state

```
class Clock extends React.Component {
    constructor(props) {
        super(props);
        this.state = {date: new Date()};
    }
    render() {
        return (
            <div>
                <h1>Hello, world!</h1>
                <h2>{this.state.date.toLocaleTimeString()}</h2>
            </div>
        );
    }
}
ReactDOM.render(<Clock />, document.getElementById('root'));
```

That's a boring clock...

- Let's add these definitions:
  -
```
componentDidMount() {
    this.timerID = setInterval(
        () => this.tick(),
        1000
    );
}
componentWillUnmount() {
    clearInterval(this.timerID);
}
tick() {
    this.setState({ date: new Date() });
}
```

Modifying the state

- Never directly modify the attributes of a components .state
  - Aside from the constructor
  - Call setState()
- Never reference the current state in a call to setState()
  - setState() calls may be performed asynchronously
  - setState() has a second form that accepts a function
    - First argument, cur state
    - Second argument, cur props

Modifying state examples:

- Bad:
  - ```
    this.setState({
        counter: this.state.counter + this.props.increment
    });
    ```
- Good:
  - ```
    this.setState((prevState, props) => ({
        counter: prevState.counter + props.increment
    }));
    ```

Event handling

- See example

Separation of markup and logic

- Brought up when we introduced JavaScript
  - Acting along similar lines to separation of structure and presentation from HTML/CSS
- React kind of does away with that…
  - Thoughts?

React Native

- Allows you to write Android/iOS apps using JS and React!
- Creates an app that will load a JS engine in a thread and run the React Native code that you write in that
- The UI you build via React Native will be rendered using native UI elements to the platform that you are running on (Android or iOS)