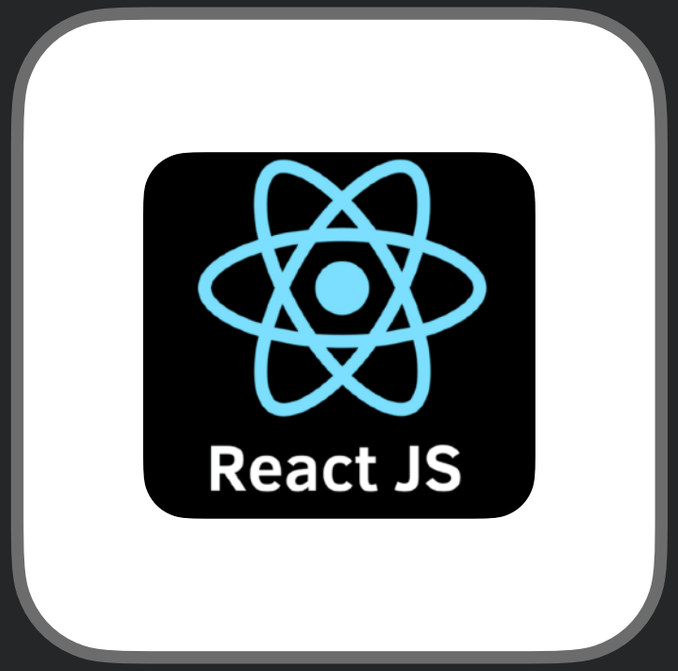
SWE 432 -Web Application Development

Fall 2022

George Mason University

Dr. Kevin Moran

Week 9: Introduction to React



Administrivia

- *Midterm* - Grades will be posted today (Sorry for delay).

- *HW Assignment 2 -* Grades posted today - please check!

- *HW Assignment 3* - Out now, Due October 27th (*next week!*), before class

Class Overview

- This Week - ***Part 1:*** An Introduction to React Programming

- Next Week - ***Part 2:*** More React and Hands-On with the React Tutorial (In-Class Activity).

Review: HTML: HyperText Markup Language

- Language for describing *structure* of a document
- Denotes hierarchy of elements
- What might be elements in this document?

HTML Example

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="style.css">
    <title>Prof Moran's Webpage</title>
```

Use <h1>, <h2>, …, <h5> for headings

```
        src="http://www.wonder-tonic.com/geocitiesizer/images/laptop-01.gif" />
        <br>
        <div class="marquee"> This is Prof Moran's ACTUAL homepage from 19991 </div> </p>
        <h2>Welcome, students!</h2>
        <p> <a href="https://www.youtube.com/watch?v=dOw4w9WgXW">See how to make this
        page</a> </p>
        <h2> Some funny links </h2>
<p>
<ul>
<li><a href="http://www.homestarrunner.com">Homestar Runner</a></li>
<li><a href="http://www.wb3w.net/The/6200riginal1420Hamsterdance.htm">Hamster
Dance</a></li> </ul>
</p> <h3> About Prof Moran </h3>

<p> Prof Moran's office is at 4442 Engineering Building. His email address is <a
href="mailto:kpmoran@gmu.edu">kpmoran@gmu.edu</a>. </p>

<p> Last updated: September 28th, 1999 </p> </div> </body>
</html>
```

Prof Kevin Moran

This is Prof Moran's ACTUAL homepage from 19991

Welcome, students!

See how to make this page

Some funny links

- Homestar Runner
- Hamster Dance

About Prof Moran

Prof Moran's office is at 4442 Engineering Building. His email address is kpmoran@gmu.edu.

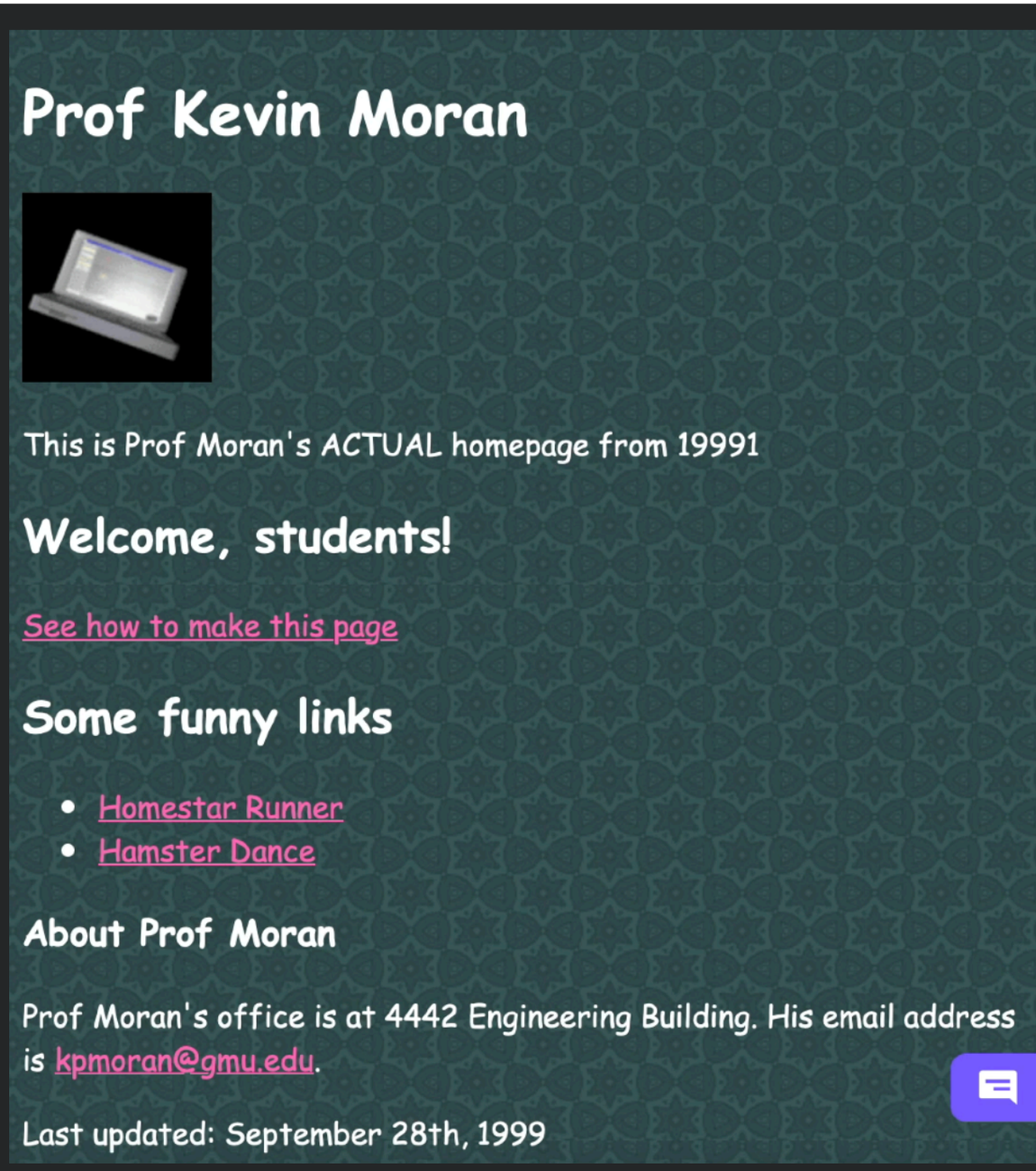
Last updated: September 28th, 1999

https://replit.com/@kmoran/html-example#index.html

HTML Example

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="style.css">
    <title>Prof Moran's Webpage</title>
    </head>
    <body>
      <h1> Prof Kevin Moran </h1>
      <div>
        <p> <img alt="My really cool laptop"
        src="http://www.wonder-tonic.com/geocitiesizer/images/laptop-01.gif" />
        <br>
        <div class="marquee"> This is Prof Moran's ACTUAL homepage from 19991 </div> </p>

                                                                              this

<li><a href="http://www.wb3w.net/The/6200riginal1420Hamsterdance.htm">Hamster
Dance</a></li> </ul>
</p> <h3> About Prof Moran </h3>

<p> Prof Moran's office is at 4442 Engineering Building. His email address is <a
href="mailto:kpmoran@gmu.edu">kpmoran@gmu.edu</a>. </p>

<p> Last updated: September 28th, 1999 </p> </div> </body>
</html>
```

Paragraphs (<p>) consist of related content. By default, each paragraph starts on a new line.



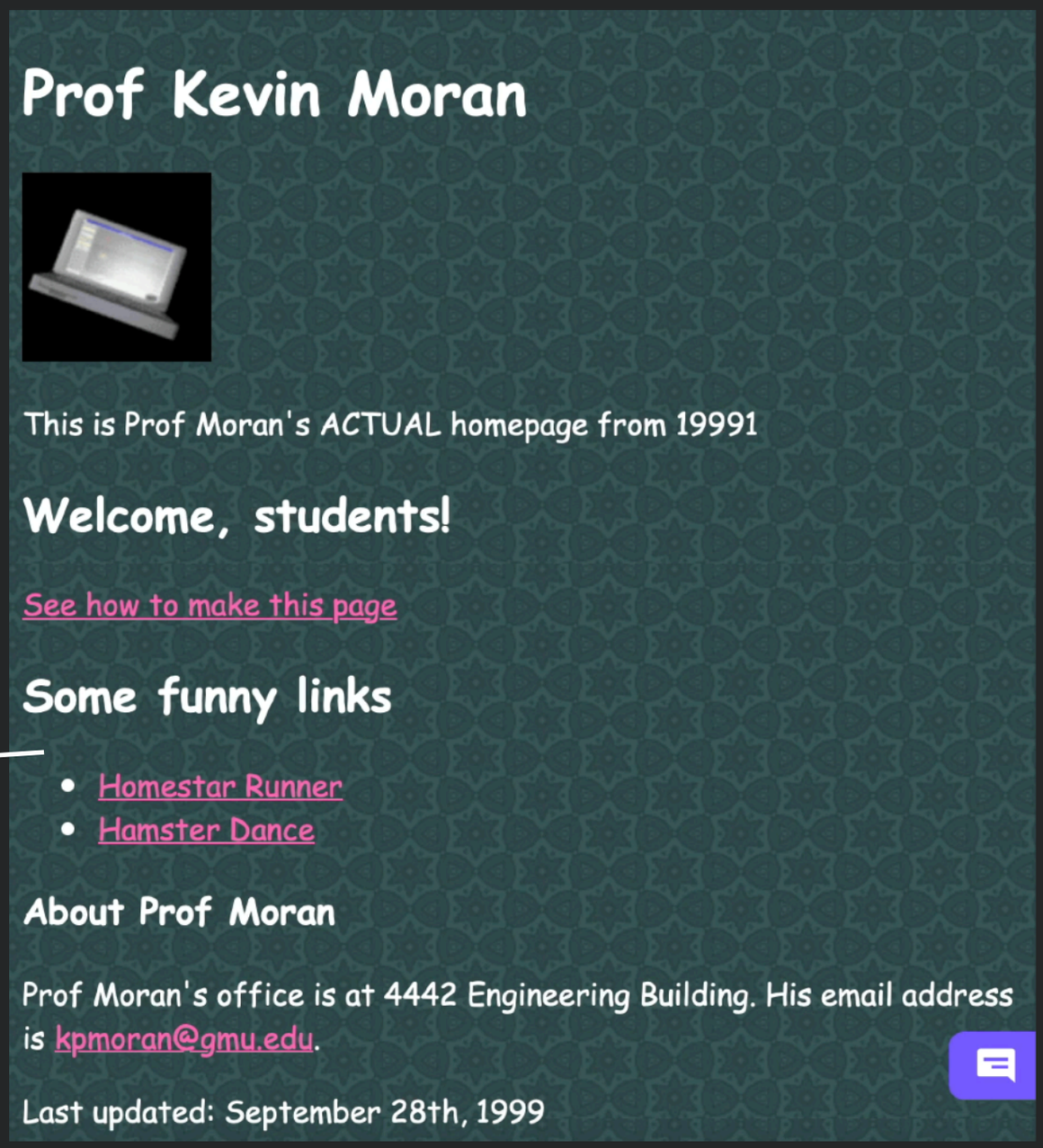
https://replit.com/@kmoran/html-example#index.html

HTML Example

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="style.css">
    <title>Prof Moran's Webpage</title>
    </head>
    <body>
      <h1> Prof Kevin Moran </h1>
      <div>
        <p> <img alt="My really cool laptop"
        src="http://www.wonder-tonic.com/geocitiesizer/images/laptop-01.gif" />
        <br>
        <div class="marquee"> This is Prof Moran's ACTUAL homepage from 19991 </div> </p>
        <h2>Welcome, students!</h2>
        <p> <a href="https://www.youtube.com/watch?v=dOw4w9WgXW">See how to make this
```

Unordered lists (<ul>) consist of list items (<li>) that each start on a new line. Lists can be nested arbitrarily deep.

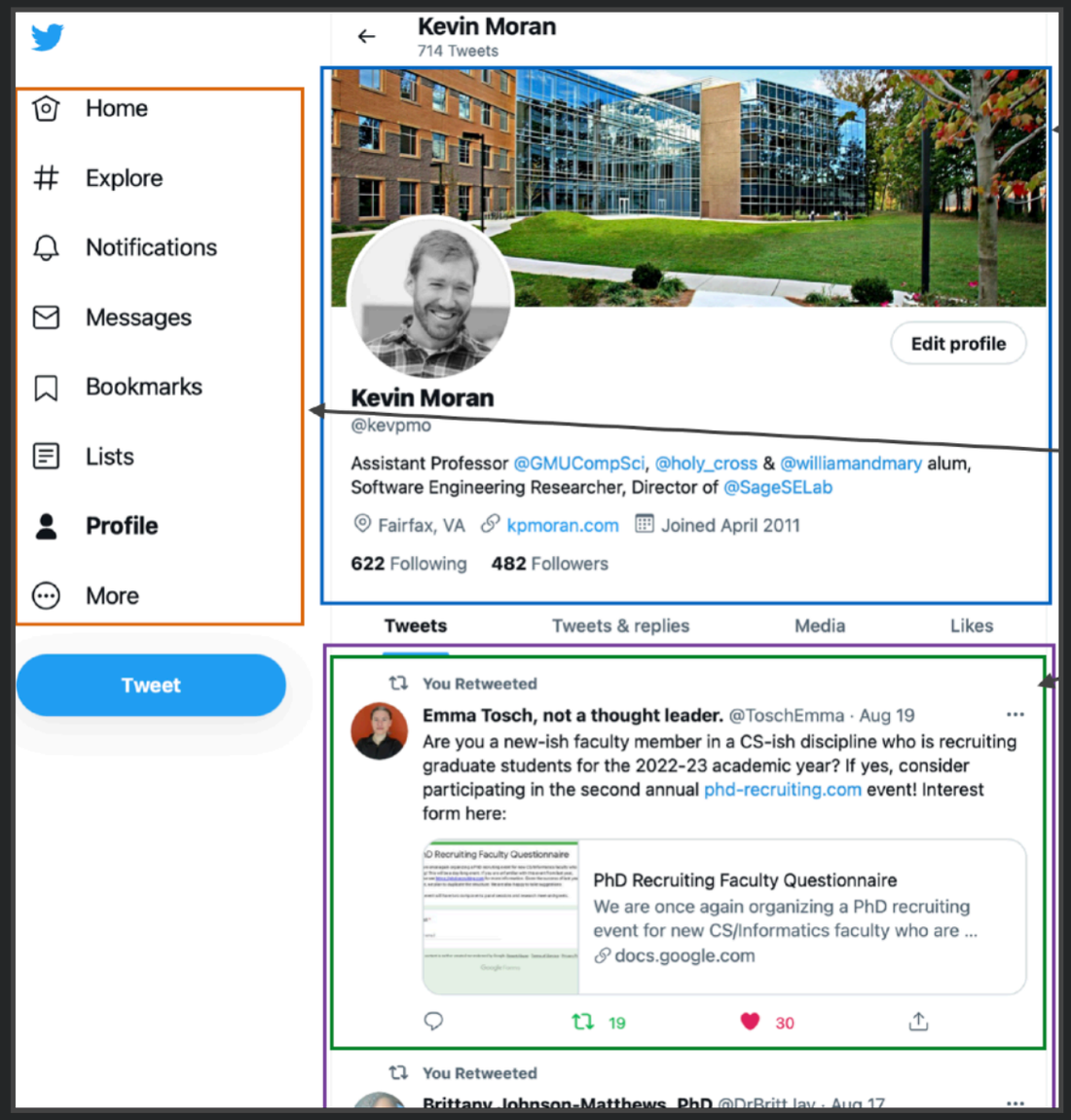
```
<p> Prof Moran's office is at 4442 Engineering Building. His email address is <a
href="mailto:kpmoran@gmu.edu">kpmoran@gmu.edu</a>. </p>

<p> Last updated: September 28th, 1999 </p> </div> </body>
</html>
```



https://replit.com/@kmoran/html-example#index.html

Review: Components

- Web pages are complex, with lots of logic and presentation
- How can we organize web page to maximize modularity?
- Solution: Components
  - Templates that correspond to a specific widget
  - Encapsulates related logic & presentation using language construct (e.g., class)



Anatomy of a React Component

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};

    // This binding is necessary to make `this` work in the callback
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(prevState => ({ isToggleOn: !prevState.isToggleOn }));
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}

ReactDOM.render(
  <Toggle />, document.getElementById('root')
);
```

https://reactjs.org/docs/handling-events.html

Reacting to change

- What happens when state of component changes?
  - e.g., user adds a new item to list
- Idea

1. Your code updates this.state of component when event(s) occur (e.g., user enters data, get data from network) using this.setState(newState)

2. Calls to this.setState *automatically* cause *render* to be invoked by framework

3. Reconciliation: Framework *diffs* output of render with *previous* call to render, updating only part of DOM that *changed*

What is state?

- All internal component data that, when changed, should trigger UI update
  - Stored as single JSON object this.state
- What isn’t state?
  - Anything that could be computed from state (redundant)
  - Other components - should build them in render
  - Data duplicated from properties.

Properties vs. State

- Properties should be ***immutable***.
  - Created through attributes when component is instantiated.
  - Should *never* update within component
  - Parent may create a new instance of component with new properties

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

- State ***changes*** to reflect the current state of the component.
  - Can (and should) change based on the current internal data of your component.

Working with State

- Constructor should initialize state of object

```
constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }
```

- Use this.setState to update state

```
    this.setState({
      date: new Date()
    });
```

- Doing this (asynchronously) will eventually result in render being invoked
  - Multiple state updates may be batched together and result in a single render call (handled by the framework)

Partial State Updates

- State is an object, may contain whatever properties you add
- Can use setState to update only parts of state you'd like to update

```
fetchPosts().then(response => {
  this.setState({
    posts: response.posts
  });
});

fetchComments().then(response => {
  this.setState({
    comments: response.comments
  });
});
```

Handling Events

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};

    // This binding is necessary to make `this` work in the callback
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(prevState => ({ isToggleOn: !prevState.isToggleOn }));
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}

ReactDOM.render(
  <Toggle />, document.getElementById('root')
);
```

https://reactjs.org/docs/handling-events.html

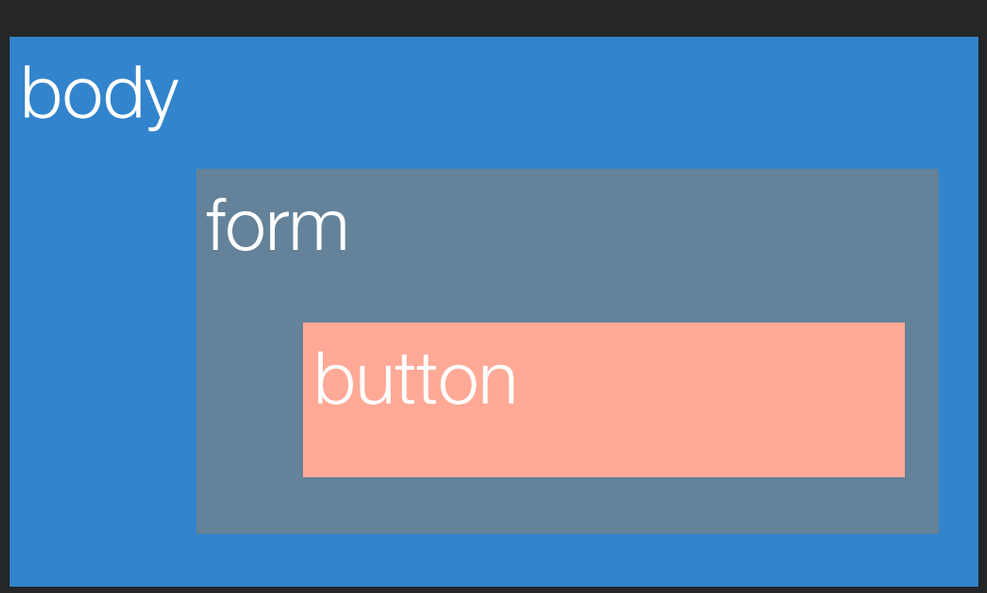
Handling Events

```
class LoggingButton extends React.Component {
  // This syntax ensures `this` is bound within
handleClick.
  // Warning: this is *experimental* syntax.
  handleClick = () => {
    console.log('this is:', this);
  }
  render() {
    return (
      <button onClick={this.handleClick}>
        Click me
      </button>
    );
  }
}
```

https://reactjs.org/docs/handling-events.html

Event Dispatching

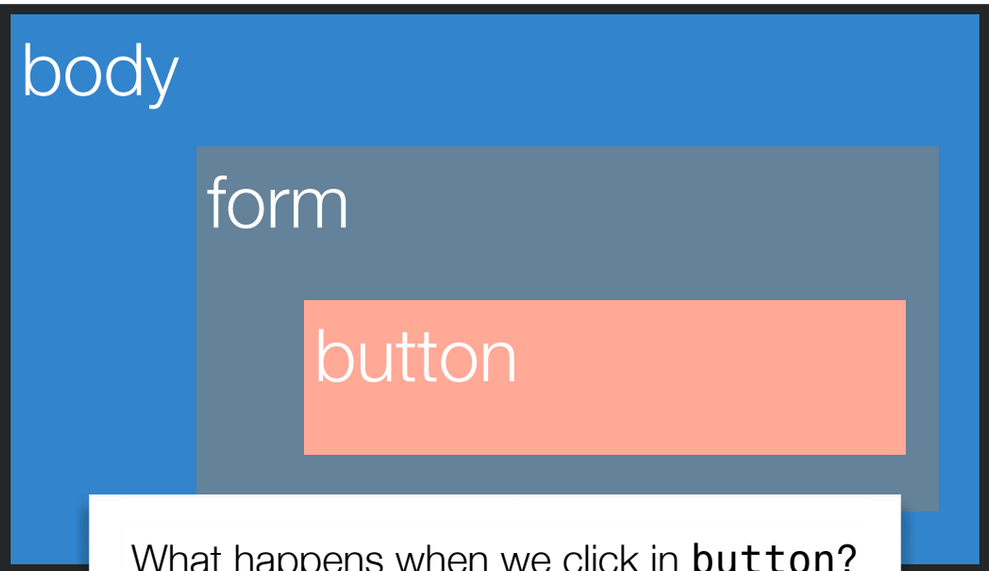
- Each event target can have (0…n) listeners registered for any given event type, called in arbitrary order
- What happens with nested elements?




```
Listener1: body onClick
Listener2: form onClick
Listener3: button onClick
```

What happens when we click in `button`?

Event Bubbling



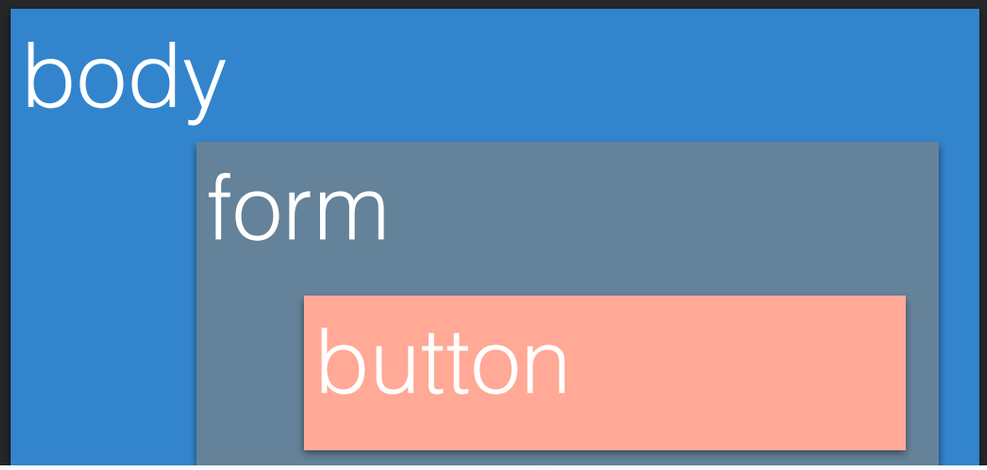
What happens when we click in `button`?

Called →

```
Listener1: body onClick
Listener2: form onClick
Listener3: button onClick
```

This is the default behavior

Event Capturing



What happens when we click in **`button`**?

Called →

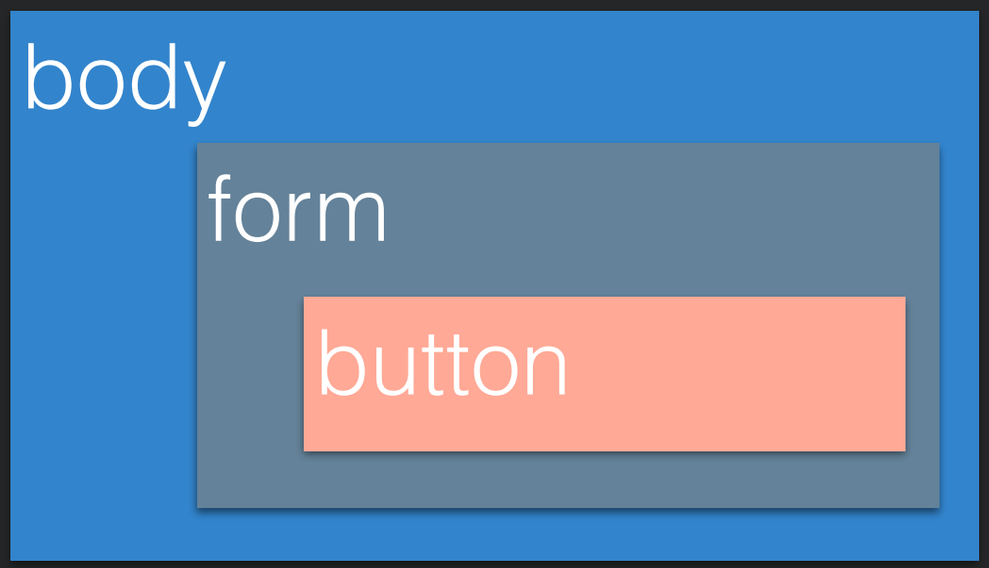
```
Listener1: body onClick
Listener2: form onClick
Listener3: button onClick
```

Enable event capturing when you register your listener:

```
element.addListener('click', myListener, true);
```

Event Dispatching

- An individual listener can stop bubbling/capturing by calling
- `event.stopPropagation();`
  - Assuming that `event` is the name of your handler's parameter




```
Listener1: body onClick
Listener2: form onClick
Listener3: button onClick
```

Nesting Components

- UI is often composed of nested components
  - Like containers in HTML, corresponds to hierarchy of HTML elements
  - But…now each element is a React component that is generated
- Parent *owns* instance of child
  - Occurs whenever component instantiates other component in render function
  - Parent configures child by passing in properties through attributes

The Data Flows Down

- State that is common to multiple components should be owned by a common ancestor
  - State can be passed into descendants as properties
- When this state can manipulated by descendants (e.g., a control), change events should invoke a handler on common ancestor
  - Handler function should be passed to descendents

https://reactjs.org/docs/state-and-lifecycle.html#the-data-flows-down

The Data Flows Down

```
class Calculator extends React.Component {
  constructor(props) {
    super(props);
    this.handleCelsiusChange = this.handleCelsiusChange.bind(this);
    this.state = {temperature: '', scale: 'c'};
  }

  handleCelsiusChange(temperature) {
    this.setState({scale: 'c', temperature});
  }

  render() {
    const scale = this.state.scale;
    const temperature = this.state.temperature;
    const celsius = scale === 'f' ? tryConvert(temperature, toCelsius) : temperature;

    return (
      <div>
        <TemperatureInput
          scale="c"
          temperature={celsius}
          onTemperatureChange={this.handleCelsiusChange} />
      </div>
    );
  }
}
```

Nesting components

```
render() {
    return (
        <div>
            <PagePic pagename={this.props.pagename} />
            <PageLink pagename={this.props.pagename} />
        </div>
    );
}
```

Establishes ownership by creating in render function.

Sets pagename property of child to value of pagename property of parent

Single Page App

- In a single page app, there is only one single HTML page loaded by browser.
- When new views are opened by user or data arrives from server, client side JavaScript code generates new views.

What's wrong with this code?

```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
    this.interval = setInterval(() => this.tick(), 1000);
  }

  tick() {
    this.setState(prevState => ({
      seconds: prevState.seconds + 1
    }));
  }

  render() {
    return (
      <div> Seconds: {this.state.seconds} </div>
    );
  }
}

ReactDOM.render(<Timer />, mountNode);
```

Component Lifecycle

```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }

  tick() {
    this.setState(prevState => ({
      seconds: prevState.seconds + 1
    }));
  }

  componentDidMount() {
    this.interval = setInterval(() => this.tick(), 1000);
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }

  render() {
    return (
      <div>
        Seconds: {this.state.seconds}
      </div>
    );
  }
}

ReactDOM.render(<Timer />, mountNode);
```

Component Lifecycle

```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }

  tick() {
    this.setState(prevState => ({
      seconds: prevState.seconds + 1
    }));
  }

  componentDidMount() {
    this.interval = setInterval(() => this.tick(), 1000);
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }

  render() {
    return (
      <div>
        Seconds: {this.state.seconds}
      </div>
    );
  }
}

ReactDOM.render(<Timer />, mountNode);
```

ReactDOM.render(...)
[component created]
constructor(...)
render()
componentDidMount()

tick()
render()
...

[component rendered
again by parent]
componentWillUnmount()
[component created]
...

Controlled Components

```
class EssayForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      value: 'Please write an essay about your favorite DOM element.'
    };

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    alert('An essay was submitted: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <textarea value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
```

Controlled Components

- Single source of truth
- Whenever a control changes its value
  - React is notified
  - State is updated
- Whenever state is updated
  - If necessary, render function executes and generates control with new value

Acknowledgements

- Slides Adapted from Dr. Thomas LaToza's SWE-432 course