Recitation #3

17-356/17-766

TAs

- **Mehul Agarwal**
  - email: mehula@andrew.cmu.edu
  - office hours: TBD

- **Rohit Shreenivas**
  - email: rshreeni@andrew.cmu.edu
  - office hours: TBD

Full-stack Development

Different levels of the stack:

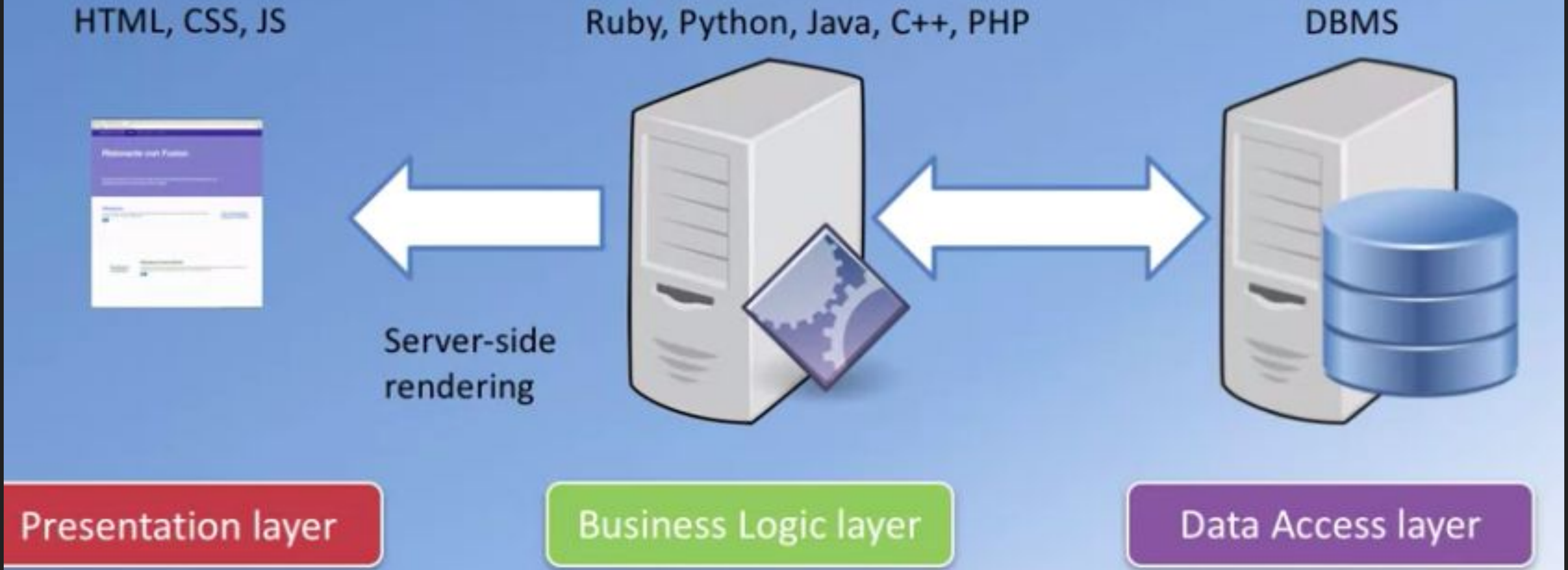
- **Backend**
- Frontend $\longrightarrow$ Today's topic
- Database
- Deployment
- Testing

and more

Frontend

- What the user sees + interacts with
- “Client-side code”
- Probably know of HTML/CSS/JS, you can build vanilla frontends with them
- Today we will learn React (https://reactjs.org)

Traditional Web Development



Styling and CSS

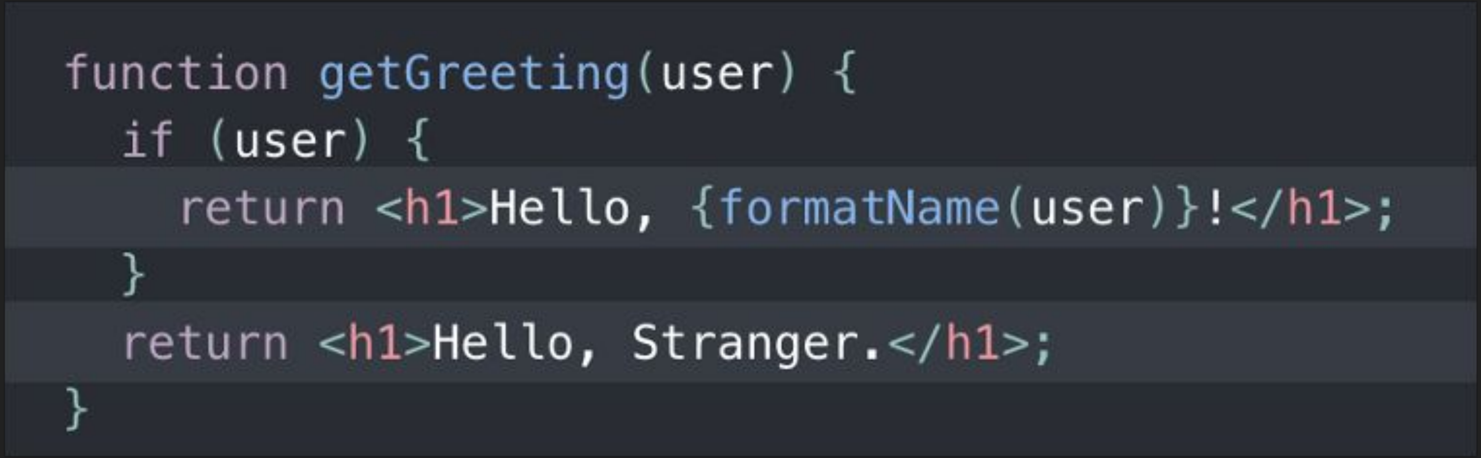
- We don’t have a dedicated recitation for styling, because there are so many many systems you can follow.
- Google is your friend.
- Course's personal pick: Flexbox
  https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox

  https://www.youtube.com/watch?v=JJSoEo8JSnc

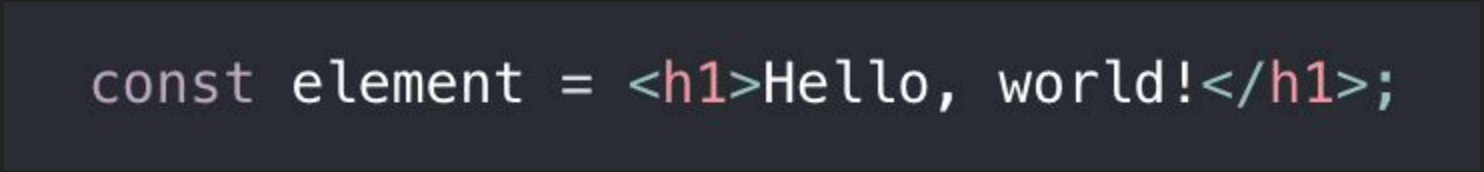
FRONTEND DEVELOPMENT USING REACT.JS

React

- Created 2011 (by Facebook)
- “Frontend JS Library” (technically not a framework, but its chill)
- Declarative, Component-Based
- Uses JSX syntax (HTML inside your JS)



```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)}!</h1>;
  }
  return <h1>Hello, Stranger.</h1>;
}
```

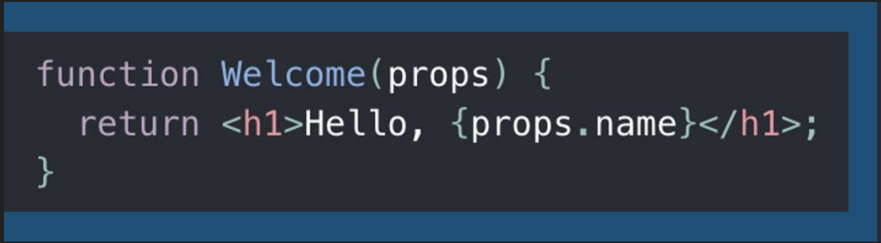


```
const element = <h1>Hello, world!</h1>;
```

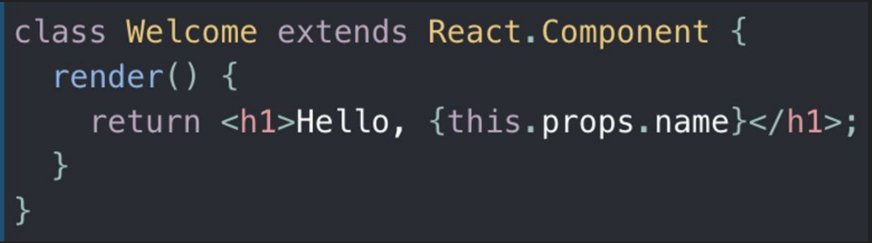
This funny tag syntax is neither a string nor HTML.

Components

- “React Only Updates What’s Necessary”
- 2 Ways: Functions and Class components
- You can nest components



```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```



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

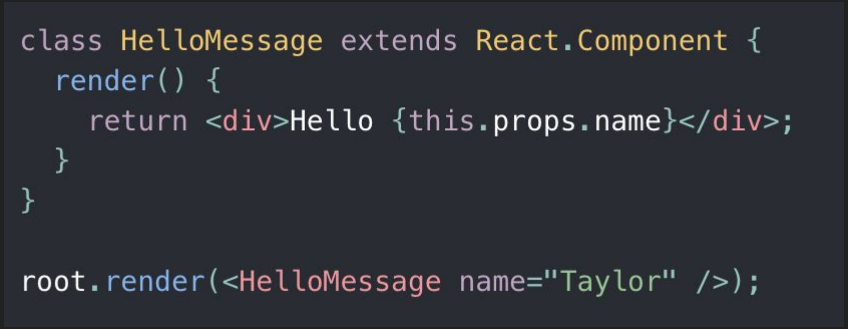
Props

- Props are similar to function parameters
- Props are read-only. Most important rule in React: “All React components must act like pure functions with respect to their props.”
- React components use props to communicate with each other. Every parent component can pass some information to its child components by giving them props.

```
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <img className="Avatar"
          src={props.author.avatarUrl}
          alt={props.author.name}
        />
        <div className="UserInfo-name">
          {props.author.name}
        </div>
      </div>
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

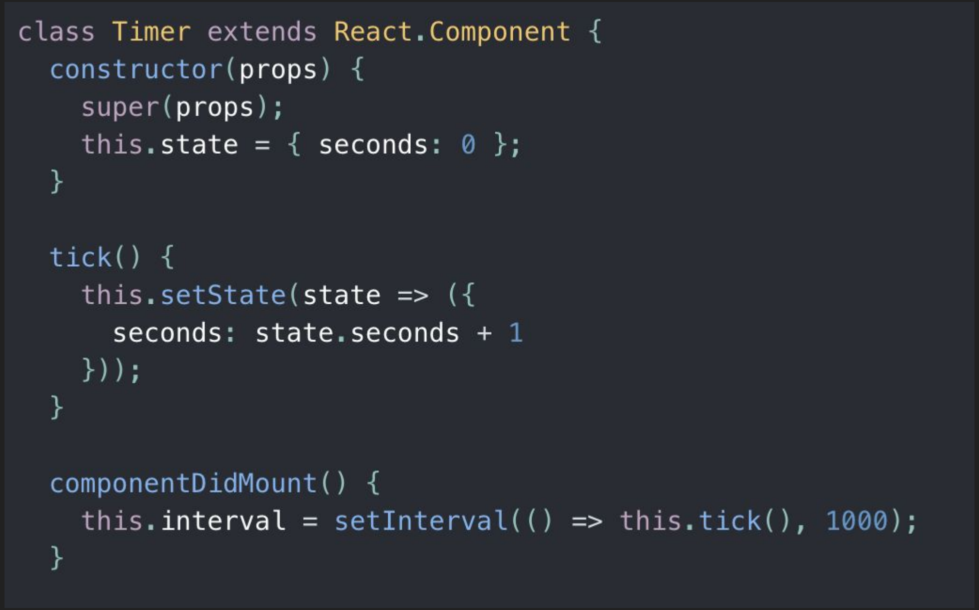
States and Lifecycle

- Component State = saved (and usually important) information about a component
- Changing state -> trigger a component reload
- Do not modify state directly (will not trigger reload). Use React’s state funcs(setState()).
- componentWillMount() and componentWillUnmount() are used to identify the lifecycle of a component



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

root.render(<HelloMessage name="Taylor" />);
```

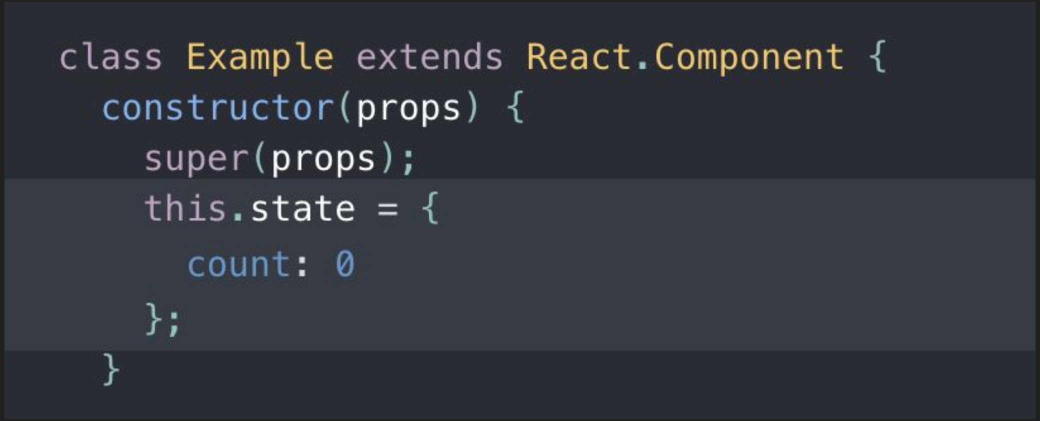
A simple React Component



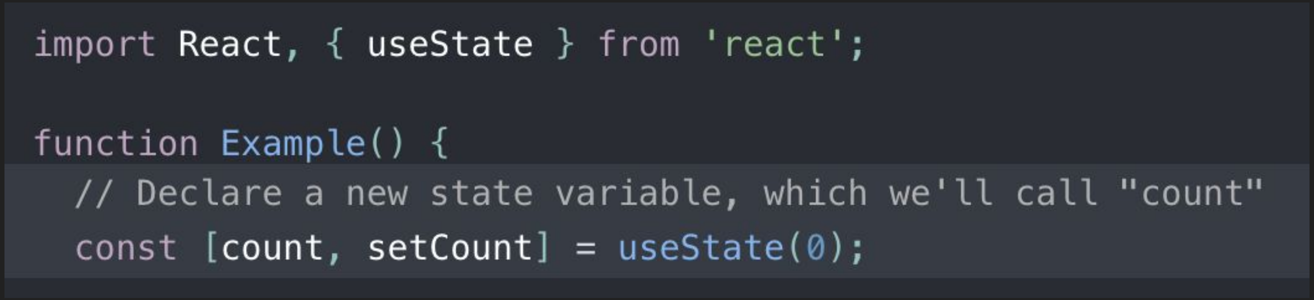
```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }

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

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

A stateful React Component

Hooks: Worth looking into



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



```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);
```

Data (States & Props) Flow

- Parent-Child relationships (think Tree)
- State is always local, but can flow downwards (to children) as props.
- Common Workaround: pass a state-modifying function as prop to child.
- Child can then call the passed function to indirectly modify parent state.

How to think + code like a React dev

- Break UI into component list/hierarchy (form the Tree)
- Build static version of UI first
- Compatible with data models, but no interactions
- Find simplest representation of UI state for each component
- Identify where state should live
- Add inverse data flow (changes go back up)

Component Libraries/Frameworks

- SUPER USEFUL (and fun to explore)
- You no longer have to style everything by hand
- Find one that you enjoy and read the docs on how to use it!
- Popular ones: MaterialUI (google), Bootstrap, Ant Design (Ant Financial, Alibaba), Evergreen
- We’re gonna use Geist UI (more obscure, to get used to learning weird things)

React demo app

- *node -v*
- *mkdir <project-name>* and *cd* into it
- *npx create-react-app .*
- *sudo npm install --global yarn*
- *yarn add @geist-ui/react*
-

https://github.com/CMU-17-356/cmu-17-356.github.io/tree/main/resources/recitations/2021/Recitation%203/todo-app-rec3/todo-app-frontend