# Introduction to React

*A JavaScript library for building user interfaces*
npx create-react-app@latest

# What is React?

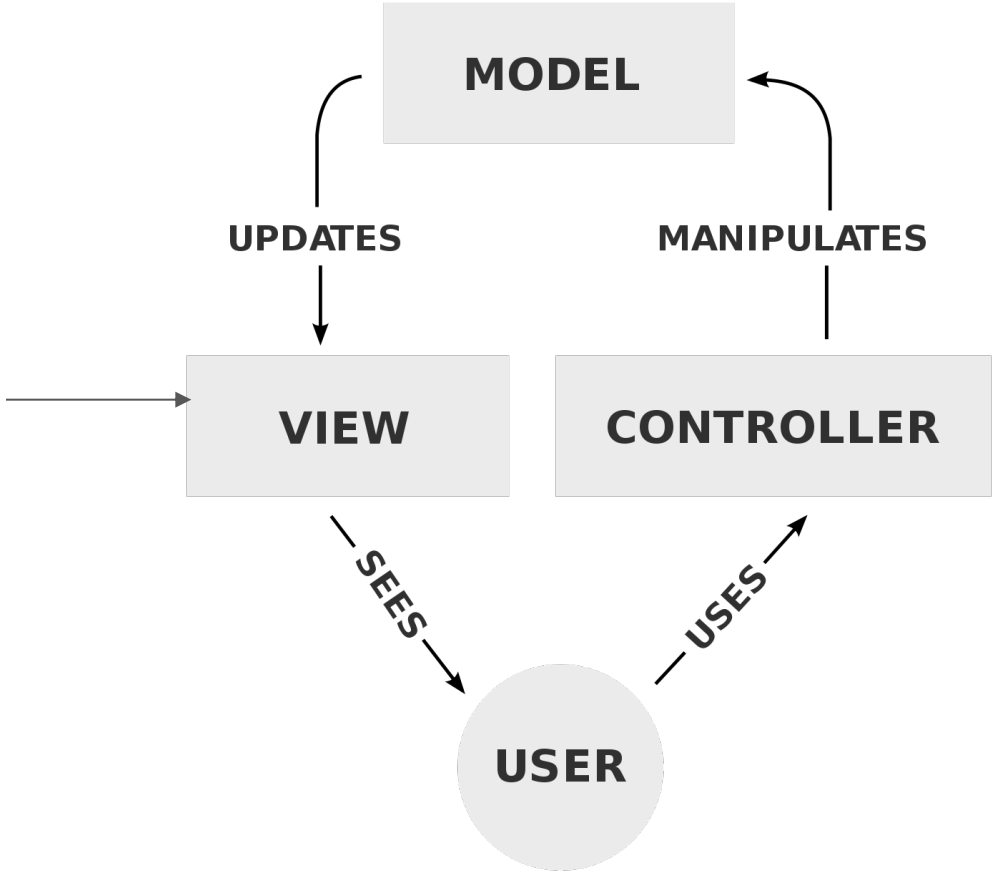
“The computer was born to solve problems that did not exist before.”

— *Bill Gates*

# What is React?

- React is a JavaScript library.
- React is an open-source project created by Facebook.
- React is the view layer of an MVC application (Model View Controller).
- React is component based.
- React is unidirectional data flow from parent to child.
- React uses virtual DOM to Render/Update UI.

# What is React?



# Introducing JSX

- This funny tag syntax is neither a string nor HTML.
- Embedding Expressions in JSX by wrapping it in curly braces.
- Specifying Children with JSX - Nesting JSX

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

const name = 'Josh Perez';
const hello = <h1>Hello, {name}</h1>;

const text = <div tabIndex="0">text</div>;
const avatar = <img src={user.avatarUrl}></img>;
```

# Rendering Elements

- Elements are the smallest building blocks of React apps.
- React elements are immutable. Once you create an element, you can’t change its children or attributes.
- React Only Updates What’s Necessary

# Components

- Almost everything in React consists of components
- Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.

# Components

# Function Components

- They are literally JavaScript functions.
- This function is a valid React component because it accepts a single “props” (which stands for properties) object argument with data and returns a React element.

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

# Class Components

- render() method returns a React element.
- this.props  - components params

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

# Conditional Rendering

- Use JavaScript operators like if or the conditional operator to create elements representing the current state.
- Preventing Component from Rendering - return null instead of its render output.

```
function UserGreeting(props) {
    return <h1>Welcome back!</h1>
}

function GuestGreeting(props) {
    return <h1>Please sign up.</h1>
}

function Greeting(props) {
    const isLoggedIn = props.isLoggedIn
    if (isLoggedIn) {
        return <UserGreeting />
    }
    return <GuestGreeting />
}
```

# Props

- Props are Read-Only.
- Whether you declare a component as a function or a class, it must never modify its own props.

```
class Car extends React.Component {
 render() {
   return <h2>I am a {this.props.brand}!</h2>;
 }
}

class Garage extends React.Component {
 render() {
   return (
     <div>
       <h1>Who lives in my garage?</h1>
       <Car brand="Ford" />
     </div>
   );
 }
}
```

# State

- State is similar to props, but it is private and fully controlled by the component.
- Do not modify state directly (instead, use setState()).
- State updates may be asynchronous.
- State updates are merged.
- In React, mutable state is typically kept in the state.

```
class MyComponent extends React.Component {
 constructor(props) {
   super(props);
   this.state = {
     name: "",
   };
 }
 onChange = (event) => {
   this.setState({ name: event.target.value });
 };

 render() {
   return (
     <div>
       <h1>My name is: {this.state.name}</h1>
       <input type="text" value={this.state.name}
onChange={this.onChange} />
     </div>
   );
 }
}
```

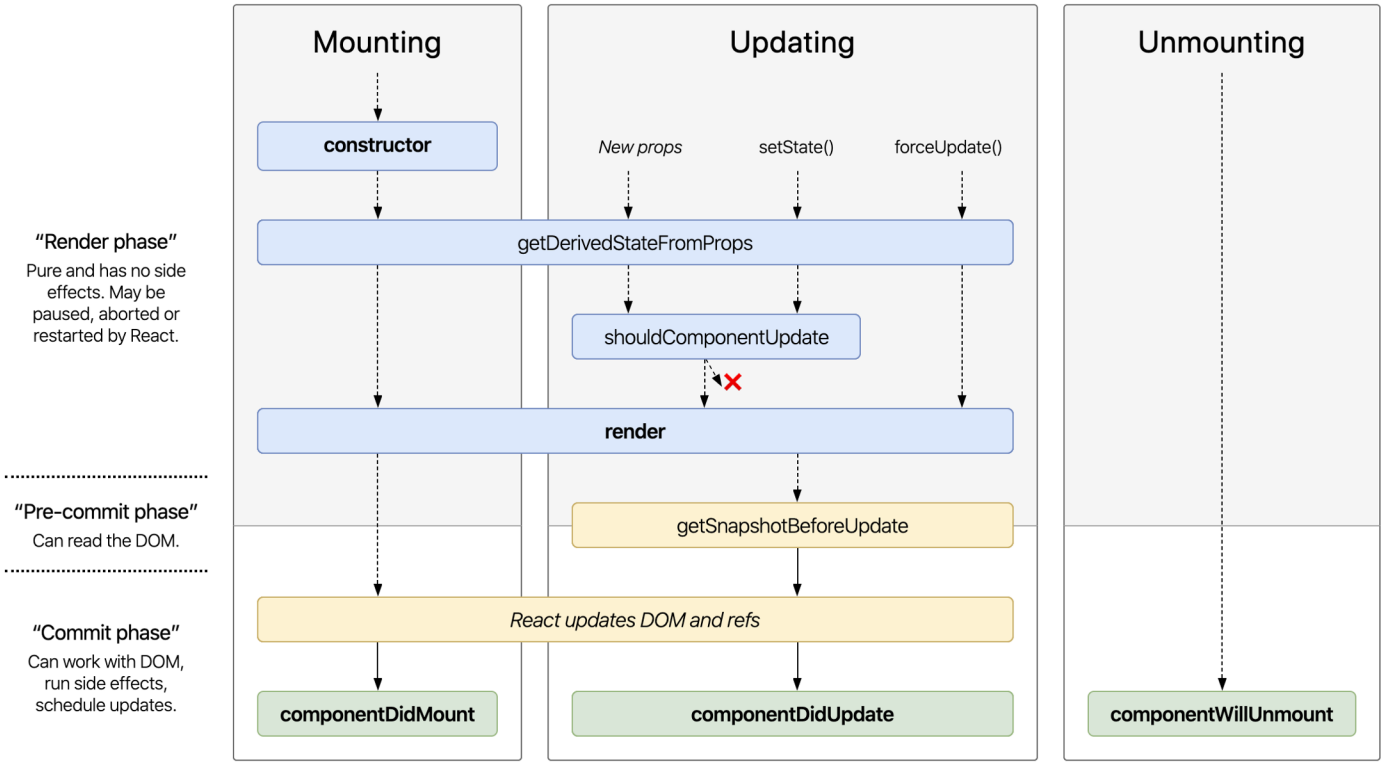
# The Component Lifecycle

Each component in React has a lifecycle which you can monitor and manipulate during its three main phases.

The three phases are: **Mounting**, **Updating**, and **Unmounting**.

```
class MyComponent extends React.Component {
    constructor(props) {
        super(props)
        console.log('constructor')
    }
    componentDidMount() {
        console.log('componentDidMount')
    }
    componentWillUnmount() {
        console.log('componentWillUnmount')
    }
    render() {
        console.log('render')
        return (
            <h1>Hello, world!</h1>
        )
    }
}
```

# The Component Lifecycle



# Lists and Keys

- A “key” is a special string attribute you need to include when creating lists of elements.
- Keys Must Only Be Unique Among Siblings.

```
function NumberList(props) {
    const numbers = props.numbers
    const listItems = numbers.map((number) =>
        <li key={number.toString()}>
            {number}
        </li>
    )

    return (
        <ul>{listItems}</ul>
    )
}
```

# Controlled Components

With a controlled component, the input's value is always driven by the React state. While this means you have to type a bit more code, you can now pass the value to other UI elements too, or reset it from other event handlers.

```
class SomeControlledComponent extends React.Component {
 constructor(props) {
   super(props);
   this.state = { name: "" };
 }

 onChange = (event) => {
   this.setState({ name: event.target.value });
 };

 onSubmit = (event) => {
   console.log(this.state);
 };

 render() {
   return (
     <div>
       <h1>{this.state.name}</h1>
       <input type="text" value={this.state.name}
onChange={this.onChange} />
       <button onClick={this.onSubmit}>Submit</button>
     </div>
   );
 }
}
```

# Uncontrolled Components

- Form data is handled by the DOM itself.
- Uncontrolled inputs are like traditional HTML form inputs, you have to ‘pull’ the value from the field when you need it.
- In most cases, it is recommend to use controlled components.

```
class SomeUncontrolledComponent extends React.Component
{
 constructor(props) {
   super(props);
   this.input = React.createRef();
 }
  onSubmit = (event) => {
   console.log(this.input.current.value);
 };

 render() {
   return (
     <div>
       <input type="text" ref={this.input} />
       <button onClick={this.onSubmit}>Submit</button>
     </div>
   );
 }
}
```

# Lifting State Up

- There should be a single “source of truth” for any data that changes in a React application.
- The state is first added to the component that needs it for rendering. Then, if other components also need it, you can lift it up to their closest common ancestor.

# Carry on with a TODO example in class

- Let’s make a TODO list app – a shopping list