Schedule

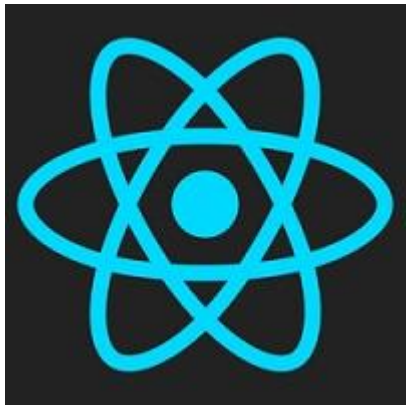
Today:

- Recall: React Components

- Props
- State

- React Component Lifecycle

- If we have time:
  - React Events
  - React Forms
  - React CSS



Recall: React Components

React components

- independent and reusable bits of code
- are like **functions that return HTML** via `render()`

- *2 types of component:*
  - Class component
  - Function component

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

```
function Car() {
    return <h2>Hi, I am also a Car!</h2>;
}
```

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

Components in Components

- Refer to components inside other components

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

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

Components in Files

- React is all about re-using code
  - be smart to insert some of your components in separate files.
- Create a new `.js` file and put the code inside it:
- **Note**: the file
  - MUST start by importing React (as before),
  - HAS TO end with the statement `export default Car;`.

```
class Car extends React.Component {
    render() {
        return <h2>Hi, I am a Car!</h2>;

    }
}

export default Car;
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import Car from './App.js';

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

Component Constructor

- Called when the component gets initiated
  - initiate the **component's properties**
  - inherit parent component super()
- In React, component's properties should be kept in an object called **state**

  e.g. add color property & use it in render()

```
class Car extends React.Component {
    constructor() {
        super();
        this.state = { color: "red" };
    }
    render() {
        return <h2>I am a {this.state.color} Car!</h2>;
    }
}
```

Props

- Props = **function arguments**
  - Passed to components via HTML attributes

  e.g. add a *brand* attribute to Car component

```
const myelement = <Car brand = "Ford" /> ;
```

- The arguments are received as **props** object

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

Passing Data

How you pass data from one component to another?

Passing Data

How you pass data from one component to another?

→ Props

- e.g. Send "brand" from Garage to Car

```
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>
    );
  }
}

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

Passing Data

- **Note**: pass a variable – NOT a string

→ Put the variable name inside curly brackets

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

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

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

Passing Data

- **Note**: pass a variable – NOT a string
- OR an object

→ Put the variable name inside curly brackets

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

class Garage extends React.Component {
  render() {
    const carinfo = {name: "Ford", model: "Mustang"};
    return (
      <div>
      <h1>Who lives in my garage?</h1>
      <Car brand={carinfo} />
      </div>
    );
  }
}

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

Props in the constructor

- If constructor,
  - the props should **always** be passed to the constructor via the super() method

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

ReactDOM.render(<Car model="Mustang"/>, document.getElementById('root'));
```

Important note on Props

React Props are **read-only**!

- You will get an error if you try to change their value.

State

- React components has a built-in **state object**.

- Is where you store property values that belongs to the component

- **When the state object changes**
  → the component re-renders.

Creating the state object

- The **state** object
  - is initialized in the constructor
  - can contain as many properties as you like

```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      brand: "Ford",
      model: "Mustang",
      color: "red",
      year: 1964
    };
  }
  render() {
    return (
      <div>
        <h1>My Car</h1>
      </div>
    );
  }
}
```

Using the state object

- Refer to the **state** object anywhere in the component by using syntax:

  `this.state.propertyname`

```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      brand: "Ford",
      model: "Mustang",
      color: "red",
      year: 1964
    };
  }
  render() {
    return (
      <div>
        <h1>My {this.state.brand}</h1>
        <p>
          It is a {this.state.color}
          {this.state.model}
          from {this.state.year}.
        </p>
      </div>
    );
  }
}
```

Changing the state object

- Use `this.setState()` method.

- When a value in the state object changes,
  → the component will re-render,
  → the output will change according to the new value(s).

```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      brand: "Ford",
      model: "Mustang",
      color: "red",
      year: 1964
    };
  }
  changeColor = () => {
    this.setState({color: "blue"});
  }
  render() {
    return (
      <div>
        <h1>My {this.state.brand}</h1>
        <p>
          It is a {this.state.color}
          {this.state.model}
          from {this.state.year}.
        </p>
        <button
          type="button"
          onClick={this.changeColor}
        >Change color</button>
      </div>
    );
  }
}
```

Important note on State

Always use the `setState()` method
to change the state object.

- it will ensure that the component knows its been updated

→ calls the render() method

→ (and all the **other lifecycle methods**) ???

Example: Present

- [Code demo]

React Component Lifecyle

- 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**.

Read:
https://www.w3schools.com/react/react_lifecycle.asp

More next week!