ReactJS

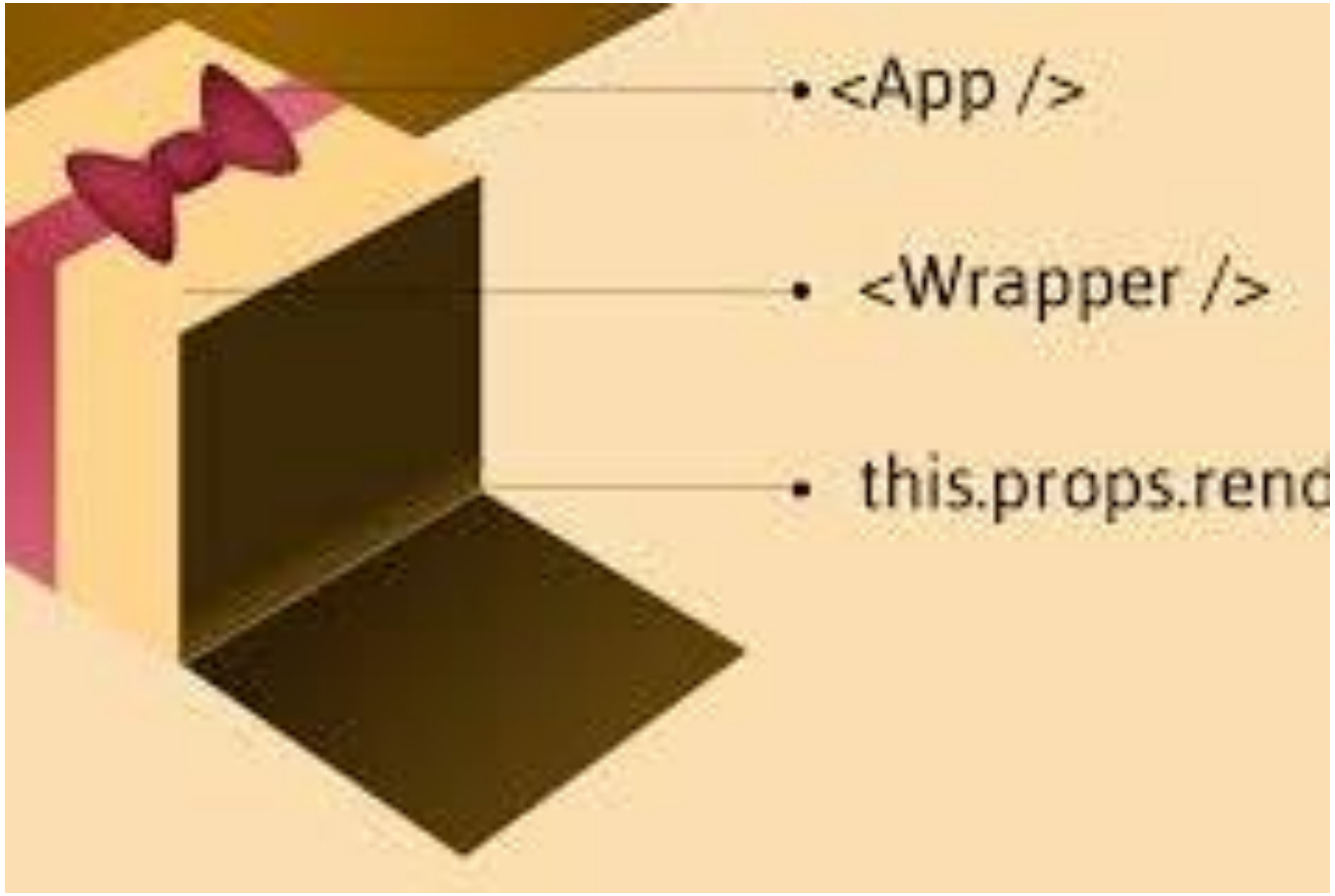
June 2021

Course Objective

- React Introduction
- Environment Setup
- ES6
- React Render Html
- JSX
- React Components
- Props and State
- Component API
- Component Life cycle
- React Forms, Events Refs, Keys
- Rest API
- React Router

Session Plan

- Props and State
- Component API
- Component Life cycle



Props & State

Props

Props stand for "Properties." They are read-only components.

Props are arguments passed into React components.

Props are passed to components via HTML attributes.

Props are immutable so we cannot modify the props from inside the component.

Props

```
import React from 'react';
import ReactDOM from 'react-dom';

class MySample extends React.Component {
  render() {
    return <h2>I am working in {this.props.name}!</h2>;
  }
}

class Sample extends React.Component {
  render() {
    return (
      <div>
      <h1>You are working for?</h1>
      <MySample name="Hexaware" />
      </div>
    );
  }
}

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

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport"
      content="width=device-width, initial-scale=1" />
    <title>React App</title>
  </head>
  <body>

    <div id="root"></div>

  </body>
</html>
```

Props

```
import React from 'react';
import ReactDOM from 'react-dom';

class MySample extends React.Component {
  render() {
    return <h2>I am working in
{this.props.name}!</h2>;
  }
}
```

We can pass the value through variable

```
class Sample extends React.Component {
  render() {
const org="Hexaware"
    return (
      <div>
      <h1>You are working for?</h1>
      <MySample name={org} />
      </div>
    );
  }
}

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

Props as object

```
class Sample extends React.Component {
  render() {
  const org={orgName :"Hexaware“, dept :”HexaVarsity”}
    return (
      <div>
      <h1>You are working for?</h1>
      <MySample name={org} />
      </div>
    );
  }
}

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

```
class MySample extends
React.Component {
  render() {
    return <h2>I am working in
{this.props.name.dept}!</h2>;
  }
}
```

Validating Props

Properties validation is a useful way to force the correct usage of the components.

```
import React from 'react';

class App extends React.Component {
  render() {
    return (
      <div>
        <h3>Array: {this.props.propArray}</h3>
        <h3>Bool: {this.props.propBool ? "True..." :
"False..."}</h3>
        <h3>Func: {this.props.propFunc(3)}</h3>
        <h3>Number: {this.props.propNumber}</h3>
        <h3>String: {this.props.propString}</h3>
        <h3>Object: {this.props.propObject.objectName1}</h3>
        <h3>Object: {this.props.propObject.objectName2}</h3>
        <h3>Object: {this.props.propObject.objectName3}</h3>
      </div>
    );    } }
```

```
App.propTypes = {
  propArray: React.PropTypes.array.isRequired,
  propBool: React.PropTypes.bool.isRequired,
  propFunc: React.PropTypes.func,
  propNumber: React.PropTypes.number,
  propString: React.PropTypes.string,
  propObject: React.PropTypes.object
}
```

```
App.defaultProps = {
  propArray: [1,2,3,4,5],
  propBool: true,
  propFunc: function(e){return e},
  propNumber: 1,
  propString: "String value...",
export default App;
```

Props Vs state

Props	state
props get passed to the component	state is managed within the component
Function Parameters	Variables declared within the body of the component
props are immutable	state can be changed
props-functional components This.props –class components	useState Hook –Functional Components This.state –class components

State

React components has a built-in state object.

The 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:

State Demo

```
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>
    );  } }
```

State object in the component is referred with this.object.propertyName

State Demo

```
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>
   ); } }

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

React Component API

- ReactJS component is a top-level API.
- React Application is reusable because of Component API
- By using Component API methods :
  - Create elements
  - Transform elements
  - Fragments
- Component API Methods
  - setState()
  - forceUpdate()
  - findDOMNode()

Component setState method

Components setState method is used to update the state of the component.
This method will not replace the state, but only add changes to the original state.

Syntax:
this.stateState(object newState[, function callback]);

forceUpdate()

This method allows us to update the component manually.

Syntax:

```
Component.forceUpdate(callback);
```

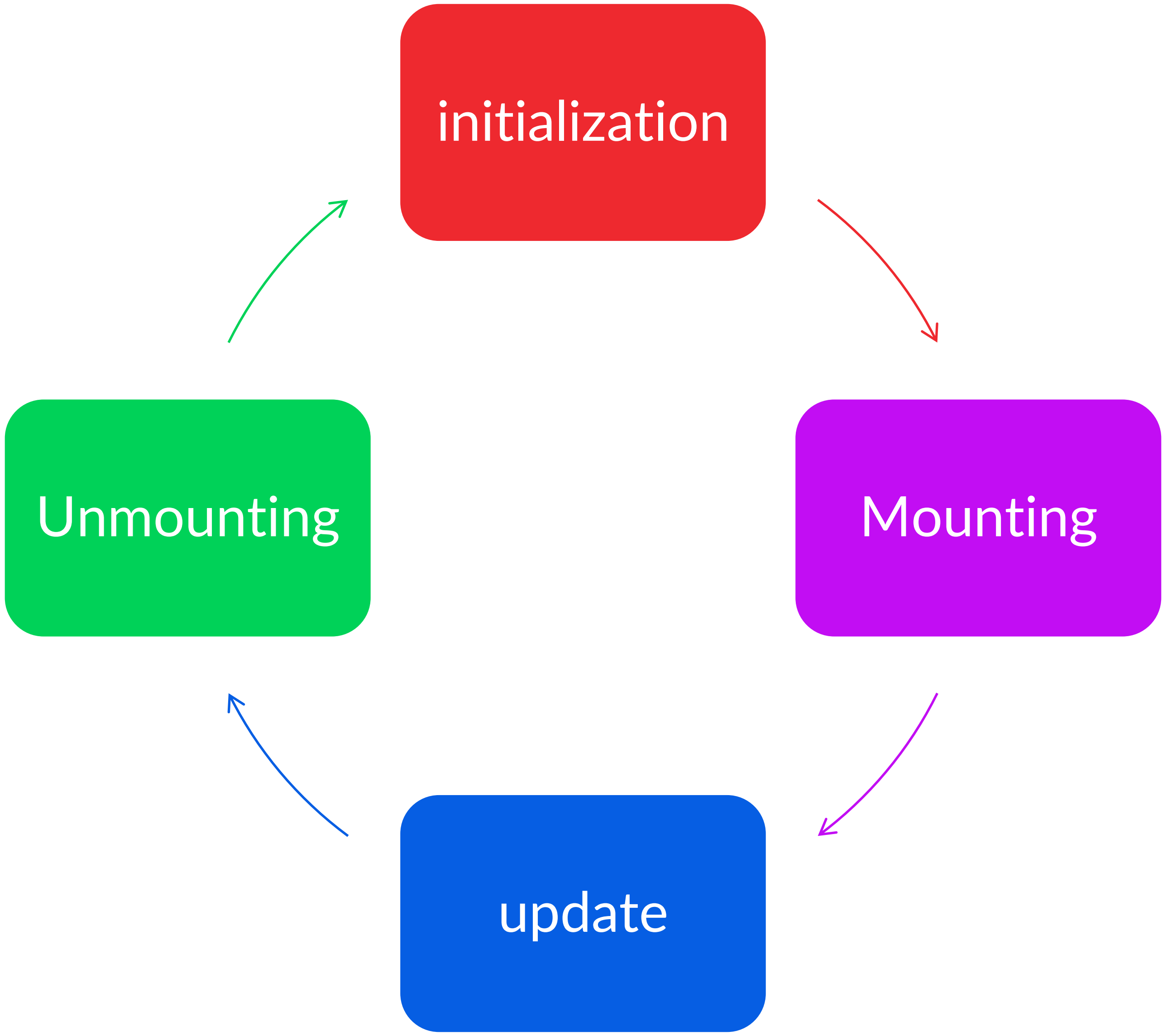
findDOMNode

- ReactDOM.findDOMNode() method is used for DOM manipulation.
- This method allows us to find or access the underlying DOM node.

Syntax

```
ReactDOM.findDOMNode(component);
```

React Components Life Cycle Methods



Life Cycle Hooks

Initialization:

In this phase, the React Component is setting up the *initial states and default props*.

Mounting

In this phase React component is ready to be mounted to the DOM. The hook is executed just before the React Component is about to mount on the DOM. This method is executed just before a component’s first render.

Life Cycle Hooks - Update

- This phase consists of the React component growing by receiving new updates and get new Props and change State.
- when the component receives new props or state is being updated, React re-render it by default when it returns true.
- Update phase can be called repeatedly.

Life Cycle Methods -Unmounting Phase

- Final phase of the react component lifecycle.
- It will be called when a component instance is destroyed and unmounted from the DOM.
- During the phase it performs cleanup task such as invalidating timers, event listener, canceling network requests, or cleaning up DOM elements.
- An unmounted component instance cannot be mount it again.

Thank you

Innovative Services

Passionate Employees

Delighted Customers