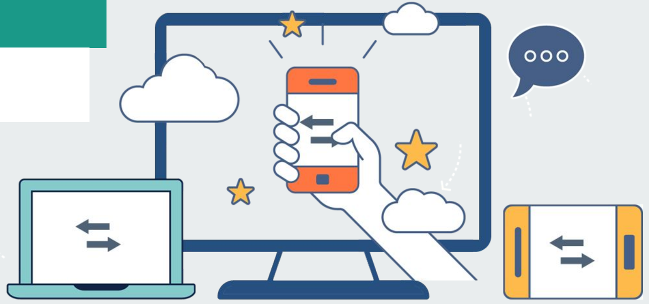
-Lecture 6-

Chapter 3 – **Introduction to React.JS**

Part II

Adil **CHEKATI**, PhD

adil.chekati@univ-constantine2.dz



Prerequisites

- ❏ JavaScript Fundamentals (Knowledge and Application)
- ❏ Front-End Development Basics (Application)
- ❏ Understanding of ES6 (Knowledge)

Introduction to React.JS

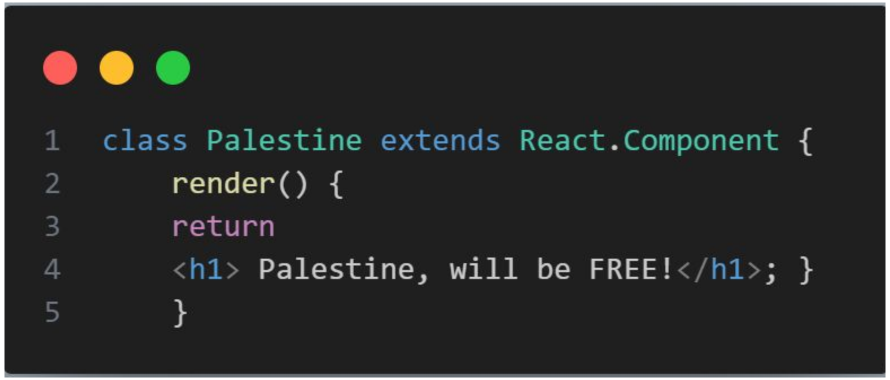
Part II

Objectives

- ➔ Understanding State and Props
- ➔ Understanding JSX and its use
- ➔ Understanding styling in React

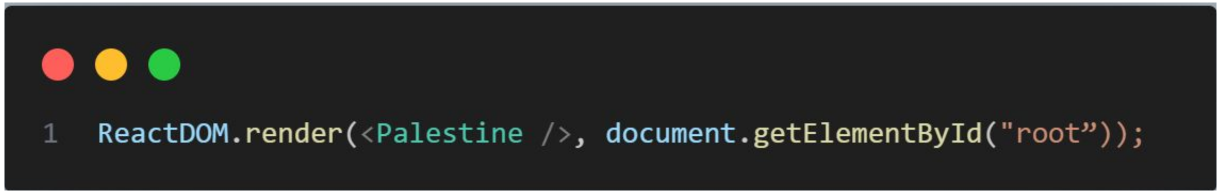
3.1 React Component Functioning

❑ A component is a React class containing the render method:



```
class Palestine extends React.Component {
    render() {
    return
    <h1> Palestine, will be FREE!</h1>; }
    }
```

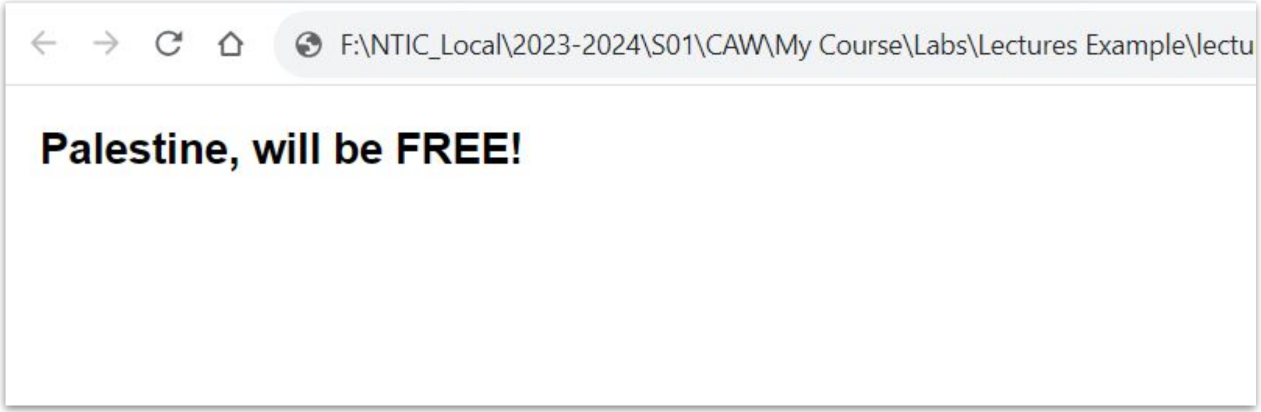
❑ The component is added to an HTML document using ReactDOM.render:



```
ReactDOM.render(<Palestine />, document.getElementById("root”));
```

3.1 React Component Functioning

❑ Display:


F:\NTIC_Local\2023-2024\S01\CAW\My Course\Labs\Lectures Example\lectu

Palestine, will be FREE!

3.2 Component display and convention

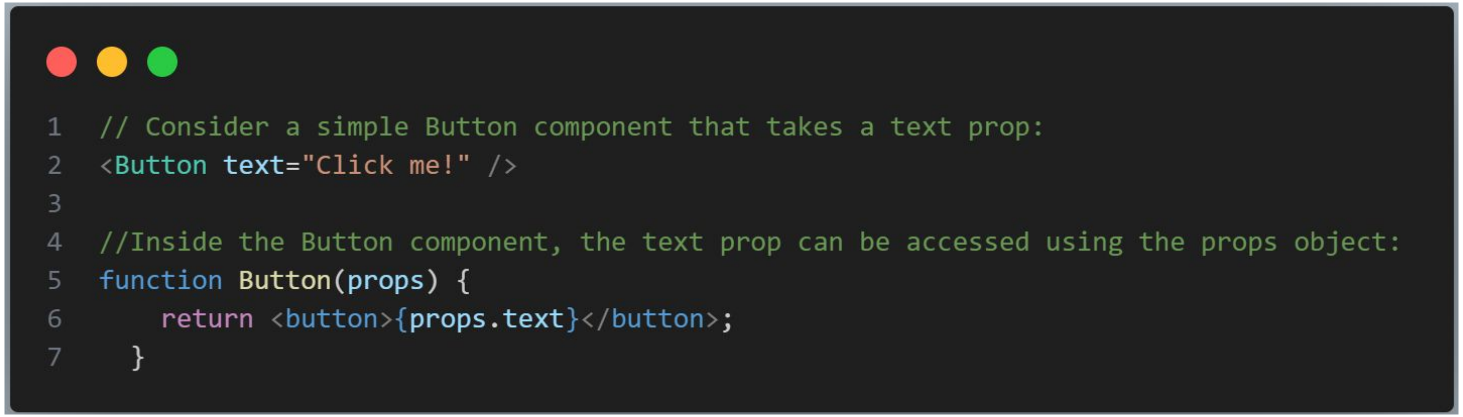
- ➔ There must be a single main component (often called `App`).
- ➔ All components must be located in a JavaScript file of the same name.
- ➔ A component can contain other components.
- ➔ The name of a component must always begin with a capital letter.
- ➔ A component can also be declared as a JavaScript function.

4. State and Props

4.1 Props

- ❑ Properties: the way data is passed to components in React.
- ❑ Similar to the attributes of an HTML element
- ❑ Can be any type of data, such as strings, numbers, arrays, or even functions.
- ❑ Passed to a component by including them as attributes when the component is used.

Example:



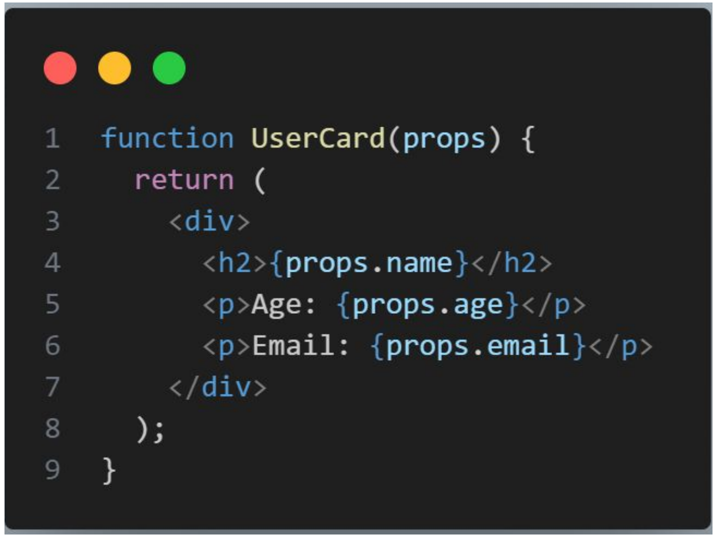
```
// Consider a simple Button component that takes a text prop:
<Button text="Click me!" />

//Inside the Button component, the text prop can be accessed using the props object:
function Button(props) {
    return <button>{props.text}</button>;
  }
```

4. State and Props

4.1.1 Using Props

➔ Props can be used to dynamically render content or pass data between components.

Example: a `UserCard` *component could receive user data as props and display it.*



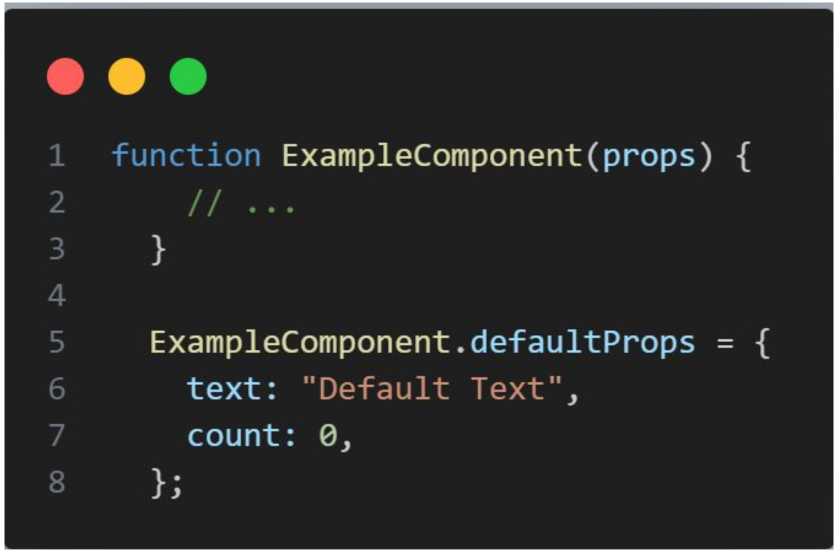
```
function UserCard(props) {
  return (
    <div>
      <h2>{props.name}</h2>
      <p>Age: {props.age}</p>
      <p>Email: {props.email}</p>
    </div>
  );
}
```

4. State and Props

4.1.2 Default Props

➔ Components can define default values for props using the `defaultProps` property.
➔ If a prop is not passed when the component is used, the default value will be used instead.

Example:



```
function ExampleComponent(props) {
    // ...
  }

  ExampleComponent.defaultProps = {
    text: "Default Text",
    count: 0,
  };
```

4. State and Props

4.2 State

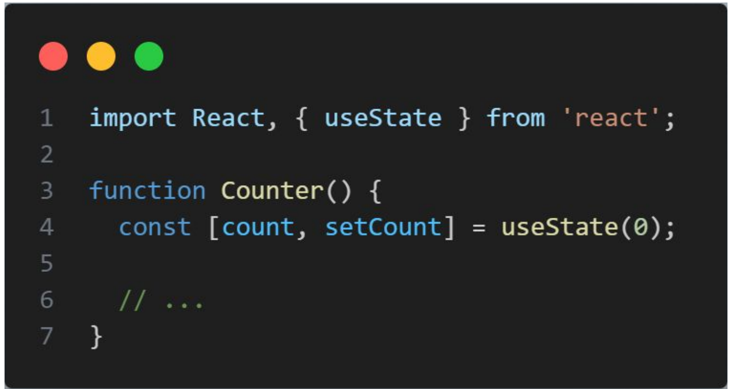
❑ State represents the values that can change during the lifetime of the component.

❑ When the state changes, React will automatically re-render the component, updating the UI accordingly.

➔ To use state in a component, we need to import the `useState` hook from React.

➔ The useState hook returns an array with two elements: the current state value and a function to update the state value.

Example:



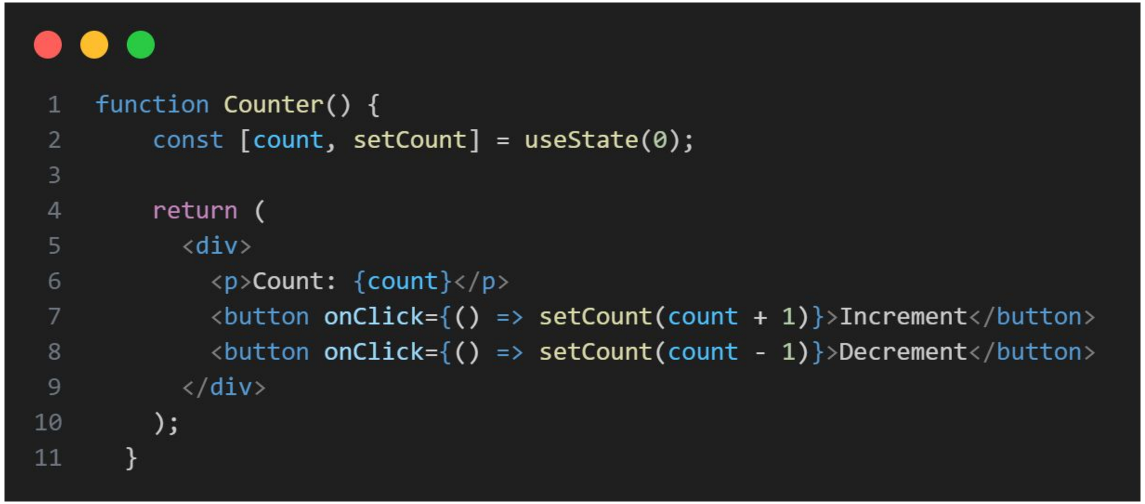
```
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  // ...
}
```

4. State and Props

4.2.1 Using State

- ➔ Once the state is set, we can use it within the component.
- ➔ State values can be accessed just like any other variables.

Example: Using the `count` *state from the previous example:*



```
function Counter() {
    const [count, setCount] = useState(0);

    return (
      <div>
        <p>Count: {count}</p>
        <button onClick={() => setCount(count + 1)}>Increment</button>
        <button onClick={() => setCount(count - 1)}>Decrement</button>
      </div>
    );
  }
```

4. State and Props

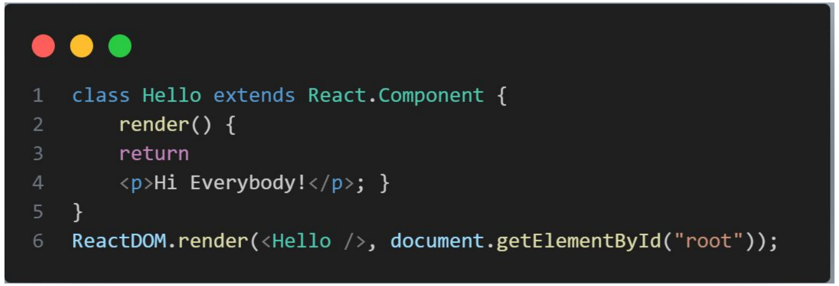
4.3 Props vs State

- ❑ Props and state have some similarities, but they serve different purposes in React.
- ❑ Props are used to pass data from parent components to child components, while state is used to manage data within a component itself.

- ❑ In general, it's recommended to use props when the data comes from a higher-level component and is expected to be static.
- ❑ State, on the other hand, is used when the data is expected to change over time within the component.

5. JSX: JavaScript XML

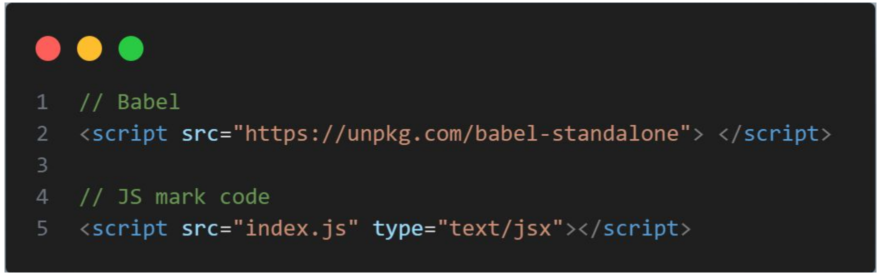
➔ It's tag-based code like HTML embedded in JavaScript code



```
class Hello extends React.Component {
    render() {
    return
    <p>Hi Everybody!</p>; }
}
ReactDOM.render(<Hello />, document.getElementById("root"));
```

➔ JSX is not JavaScript code; it must be translated into JavaScript using a transpiler:

- ◆ The most widely used is the **Babel** transpiler. Babel must be included in the HTML document.
- ◆ **Mark JS code** containing JSX with type="text/jsx".



```
// Babel
<script src="https://unpkg.com/babel-standalone"> </script>

// JS mark code
<script src="index.js" type="text/jsx"></script>
```

(For transcription of JSX into JavaScript see examples on: https://babeljs.io/repl)

5. JSX: JavaScript XML

2.2. Advantages of using JSX

a) Customizability

JSX enables the creation of custom components, giving you full control over the appearance and functionality of your user interface.

b) Readability

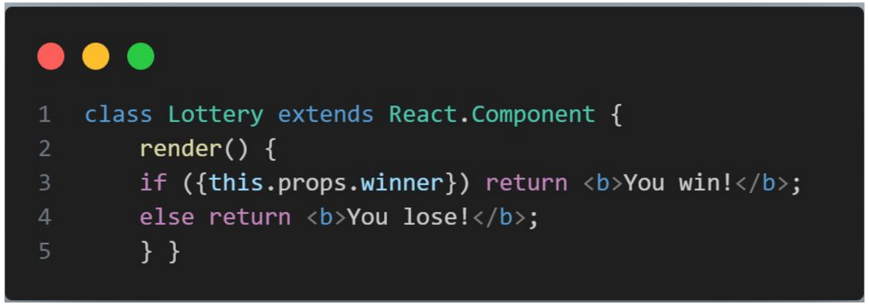
JSX provides a more readable syntax for creating React elements compared to using pure JavaScript.

By understanding the concepts of components and JSX, you can leverage the power of React.JS to build ***modular, reusable, and maintainable*** user interfaces.

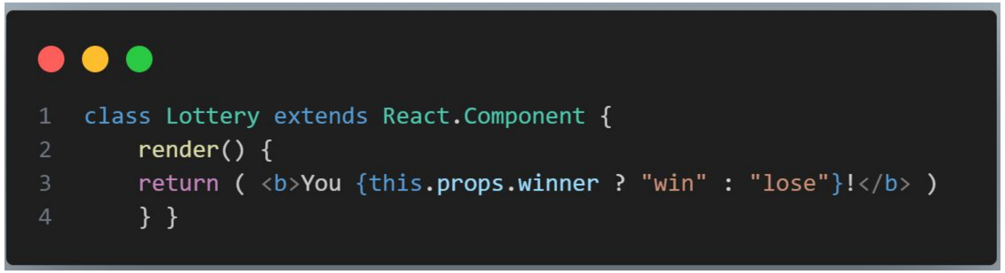
5. JSX: JavaScript XML

5.1. Conditional statements in JSX

➔ Conditional expressions (if or if...else) can be used with JSX.


```
class Lottery extends React.Component {
    render() {
    if ({this.props.winner}) return <b>You win!</b>;
    else return <b>You lose!</b>;
    } }
```

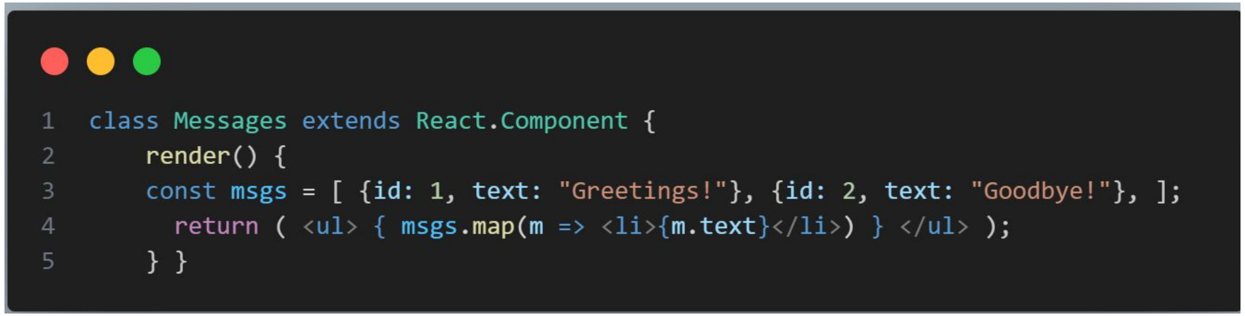
➔ Using the ternary operator (? :)


```
class Lottery extends React.Component {
    render() {
    return ( <b>You {this.props.winner ? "win" : "lose"}!</b> )
    } }
```

5. JSX: JavaScript XML

5.2 Loops in JSX

➔ We frequently use array.map(fn) to display elements of an array loop) in JSX:



```
class Messages extends React.Component {
    render() {
    const msgs = [ {id: 1, text: "Greetings!"}, {id: 2, text: "Goodbye!"}, ];
      return ( <ul> { msgs.map(m => <li>{m.text}</li>) } </ul> );
    } }
```

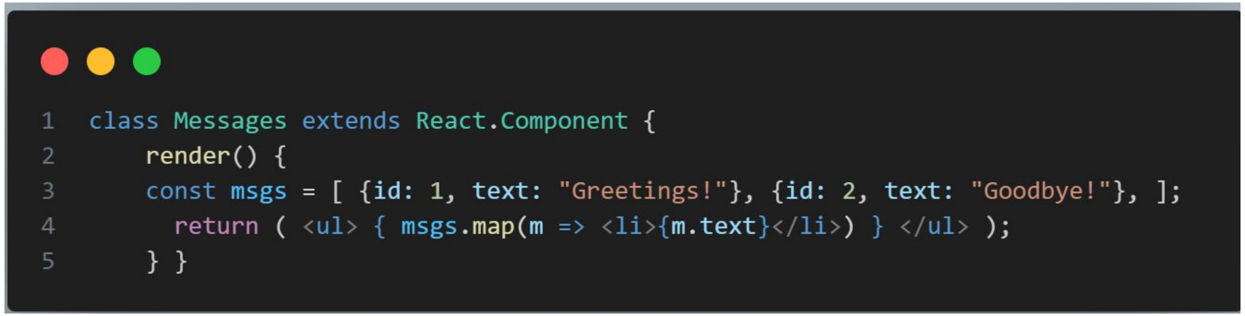
➔ This displays :

- `Greetings!`
- `Goodbye!`

5. JSX: JavaScript XML

5.2 Loops in JSX

➔ We frequently use array.map(fn) to display elements of an array loop) in JSX:



```
class Messages extends React.Component {
    render() {
    const msgs = [ {id: 1, text: "Greetings!"}, {id: 2, text: "Goodbye!"}, ];
      return ( <ul> { msgs.map(m => <li>{m.text}</li>) } </ul> );
    } }
```

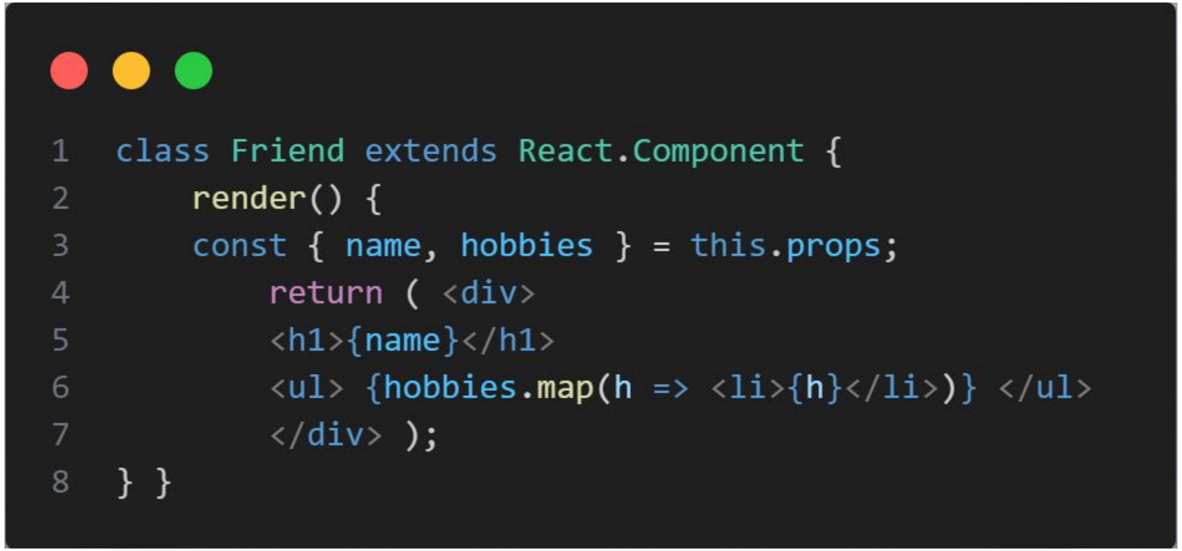
➔ This displays :

- Greetings!
- Goodbye!

5. JSX: JavaScript XML

5.2 Loops in JSX

Another example, the `demo/friends/Friend.js` file:

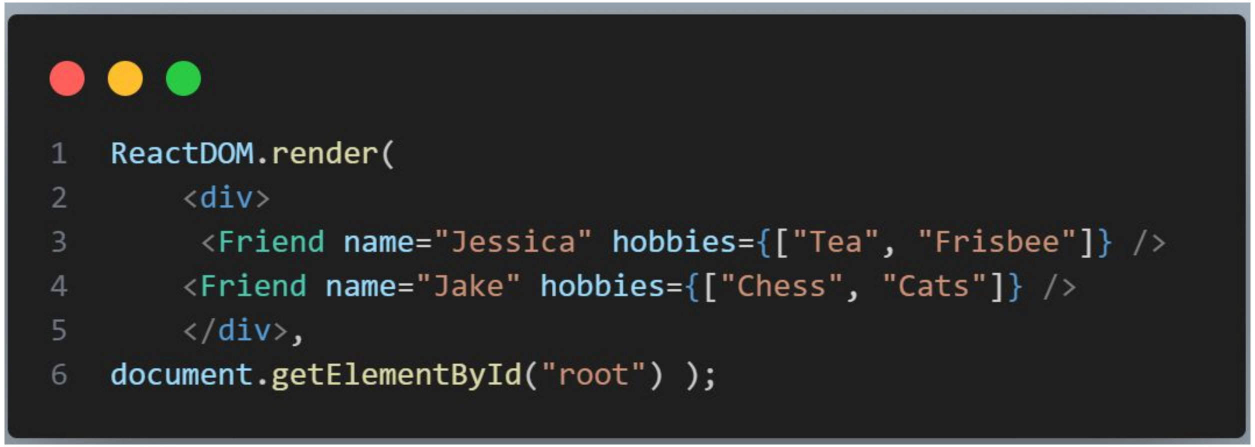


```
class Friend extends React.Component {
    render() {
    const { name, hobbies } = this.props;
        return ( <div>
        <h1>{name}</h1>
        <ul> {hobbies.map(h => <li>{h}</li>)} </ul>
        </div> );
} }
```

5. JSX: JavaScript XML

5.2 Loops in JSX

In the `demo/friends/Index.js` file:

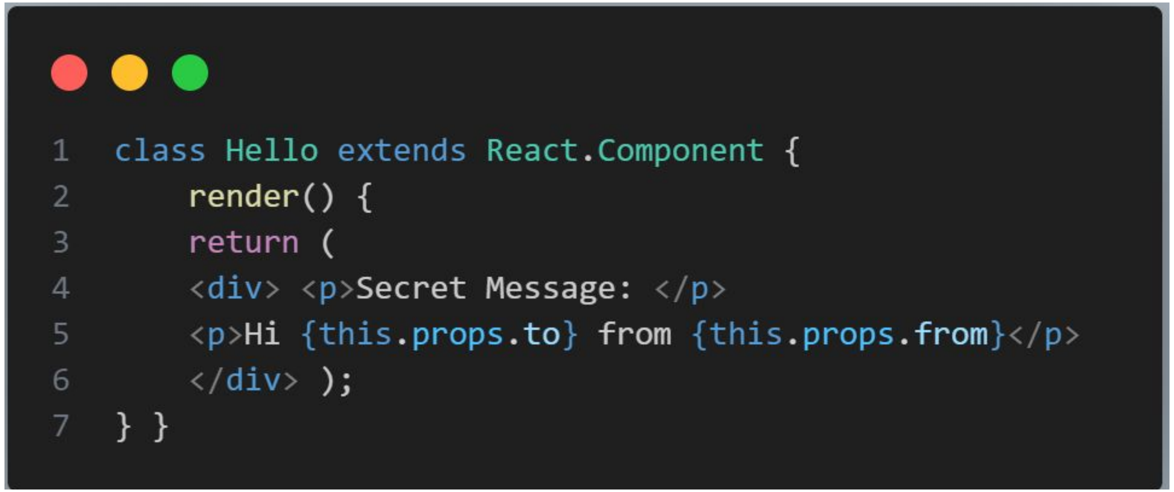


```
ReactDOM.render(
     <div>
      <Friend name="Jessica" hobbies={["Tea", "Frisbee"]} />
     <Friend name="Jake" hobbies={["Chess", "Cats"]} />
     </div>,
document.getElementById("root") );
```

6. Reusing components

→ A component can be used several times.

Xampl: displaying 2 Hello components in components in two different places in a document.

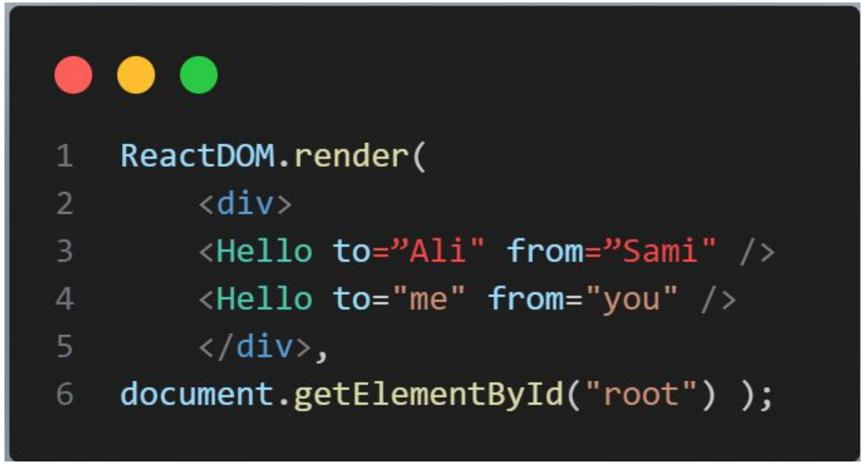


```
class Hello extends React.Component {
    render() {
    return (
    <div> <p>Secret Message: </p>
    <p>Hi {this.props.to} from {this.props.from}</p>
    </div> );
} }
```

6. Reusing components

➔ A component can be used several times.

Xampl: displaying 2 Hello components in components in two different places in a document.

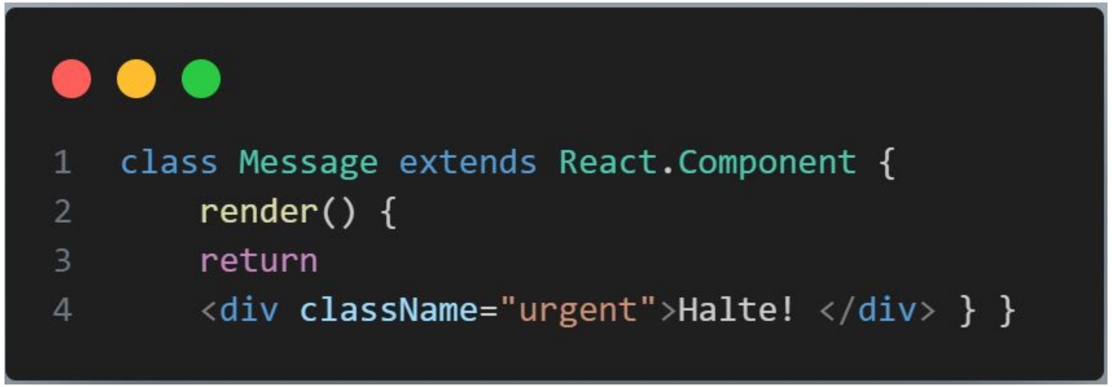


```
ReactDOM.render(
        <div>
        <Hello to=”Ali" from=”Sami" />
        <Hello to="me" from="you" />
        </div>,
document.getElementById("root") );
```

7. Styling in React

- ➔ It's possible to add CSS classes to JSX.
- ➔ We'll use `className` instead of `class` (since class is a word reserved for class in JavaScript).

Example:

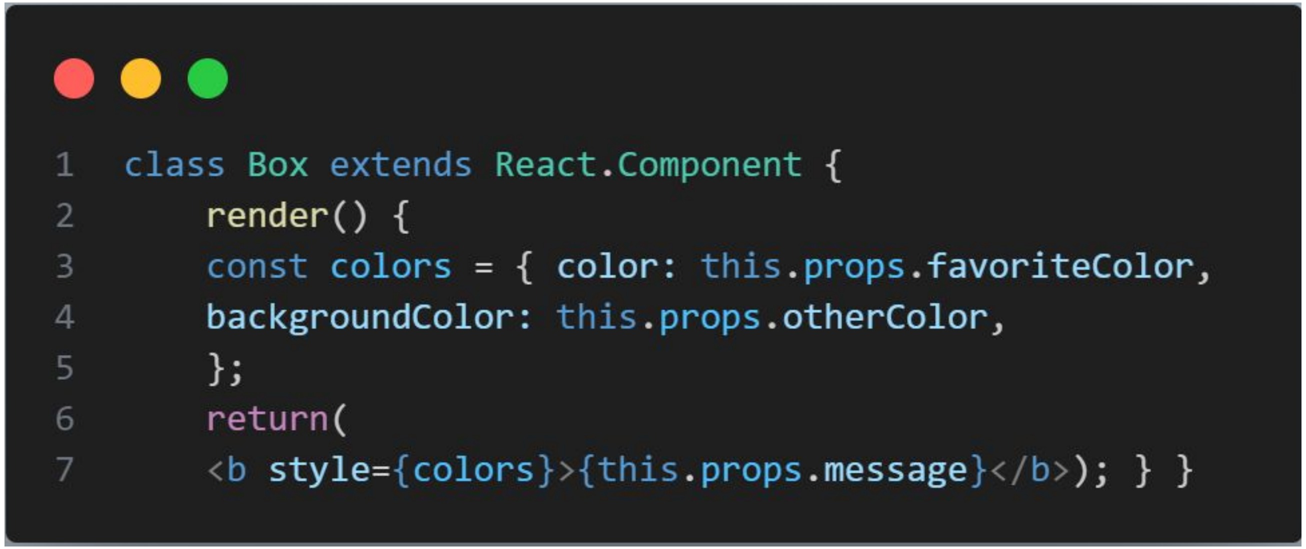


```
class Message extends React.Component {
     render() {
     return
     <div className="urgent">Halte! </div> } }
```

7. Styling in React

➔ It is possible to use the style attribute in JSX.
➔ We can directly use JavaScript objects as style values.

Example:



```
class Box extends React.Component {
    render() {
    const colors = { color: this.props.favoriteColor,
    backgroundColor: this.props.otherColor,
    };
    return(
    <b style={colors}>{this.props.message}</b>); } }
```

Lab Exercises Submission Guidelines

➔ **Deadline:**
At the end of each Lab session (no later than Saturday at 23:59)
To: adil.chekati@univ-constantine2.dz

➔ **Link to be submitted:**
Github repository link.

Textbook

➔ All academic materials will be available on:

Google Drive.
E-learning platform of Constantine 2 University.
Google Classroom.

aoa5lne

Google Classroom

SCAN ME!

References

➔ **Book:**
Alex Banks, Eve Porcello - *Learning React: Modern Patterns for Developing React Apps* (2020)

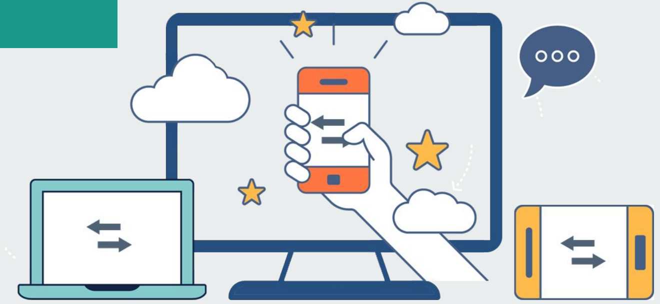
MOOC
React - The Complete Guide (incl. Hooks, React Router, Redux)" on Udemy
(https://github.com/PacktPublishing/React---The-Complete-Guide-includes-Hooks-React-Router-and-Redux-Second-Edition)

Online Resource:
React.js official documentation
(https://react.dev/learn)

Next Lecture

-Lecture 7-

Chapter 4– **Front end React development**



Adil **CHEKATI**, PhD

adil.chekati@univ-constantine2.dz

Questions,
& comments...

📧 adil.chekati@univ-constatine2.dz