React Fundamentals

Bok, Jong Soon
javaexpert@nate.com
https://github.com/swacademy/React

What is JSX?

- JavaScript XML or JavaScript Extension
- Is a syntax extension for JavaScript
- Primarily used with React to describe what the UI should look like.
- Allows developers to write HTML-like code within JavaScript.
- Making it easier to create and visualize the structure of user interfaces.

Key Features of JSX

- HTML-Like Syntax
  - JSX lets you write elements that look like HTML, but they are actually JavaScript objects.

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

- Embedding Expressions
  - Can embed any JavaScript expression within JSX by enclosing it in curly braces { }

```
const name = 'John';
const element = <h1>Hello, {name}!</h1>;
```

Key Features of JSX (Cont.)

- Attributes
  - Can use attributes similar to HTML, and they can also be dynamic.

```
const user = { avatarUrl: 'http://example.com/avatar.jpg' };
const element = <img src={ user.avatarUrl }
                     alt="User Avatar" />;
```

- JSX Prevents Injection Attacks
  - JSX escapes any values embedded in it before rendering them.
  - This means that it is safe to embed user input in JSX

```
const userInput = '<script>alert("hacked")</script>';
const element = <div>{userInput}</div>;
```

Key Features of JSX (Cont.)

- Components
  - JSX can be used to create React components.
  - Components can be ***functions*** or ***classes*** that return JSX.

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

- Fragment Syntax
  - JSX supports fragments, which allow you to group multiple elements without adding extra nodes to the DOM.

How JSX Works

- JSX is not understood directly by the browser.
- Therefore, it needs to be transformed into regular JavaScript by a compiler such as ***Babel***.

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

```
const element = React.createElement('h1', null, 'Hello, world!');
```

- This **`React.createElement`** function call creates an object representing the element.
- React uses this object to construct and update the DOM efficiently.

Benefits of Using JSX

- Readability
  - JSX syntax closely resembles HTML, making it easier to understand and write for developers familiar with web development.
- Component-Based
  - Encourages the use of reusable components, improving code modularity and maintainability.
- Integration with JavaScript
  - Allows for seamless integration of JavaScript logic within the UI, making it more powerful and flexible.

JSX is a powerful feature in React that enhances the developer experience by combining the best parts of JavaScript and HTML.

JSX Syntax and Expressions

- Elements and Attributes

```
const element = <div className="container">Hello, world!</div>;
```

- Embedding JavaScript Expressions

```
const user = { firstName: 'John', lastName: 'Doe' };
const element = <h1>Hello, {user.firstName} {user.lastName}!</h1>;
```

JSX Syntax and Expressions (Cont.)

- Conditionals and Loops

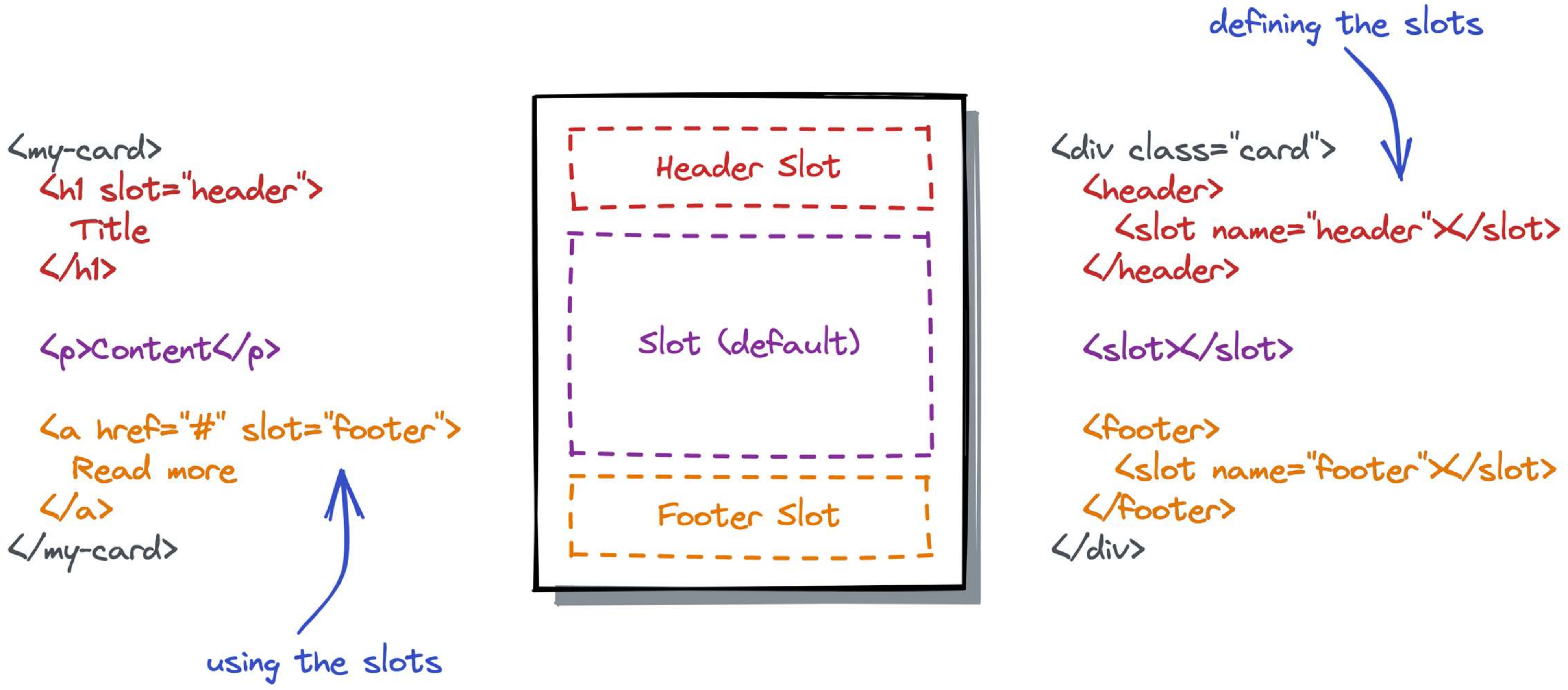
```
const messages = ['Hello', 'World'];
const element = (
  <ul>
    {messages.map((message, index) => <li key={index}>{message}</li>)}
  </ul>
);
```

Lab. JSX

Components

- Are the building blocks of React applications.
- Are self-contained, reusable pieces of UI that can be composed to create complex interfaces.
- Can be written as JavaScript functions or classes.
- Encapsulate the rendering logic and ***state*** for a portion of the UI.

Components (Cont.)

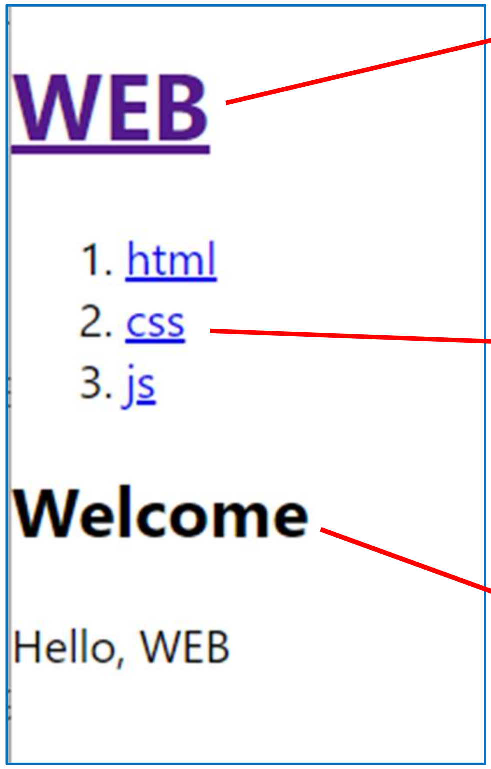


```
<my-card>
  <h1 slot="header">
    Title
  </h1>

  <p>Content</p>

  <a href="#" slot="footer">
    Read more
  </a>
</my-card>
```

```
<div class="card">
  <header>
    <slot name="header"></slot>
  </header>

  <slot></slot>

  <footer>
    <slot name="footer"></slot>
  </footer>
</div>
```

https://sandroroth.com/blog/react-slots/

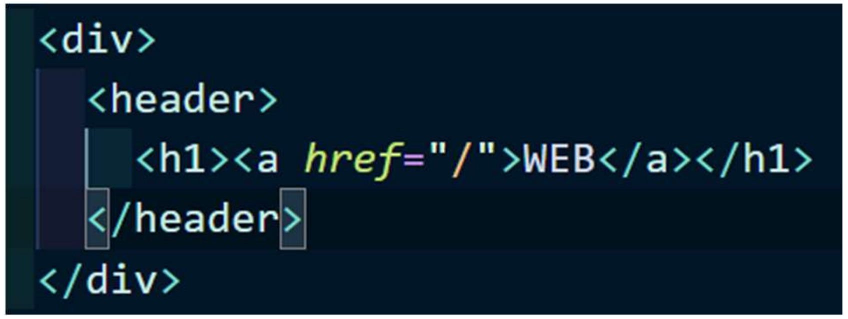
Components (Cont.)

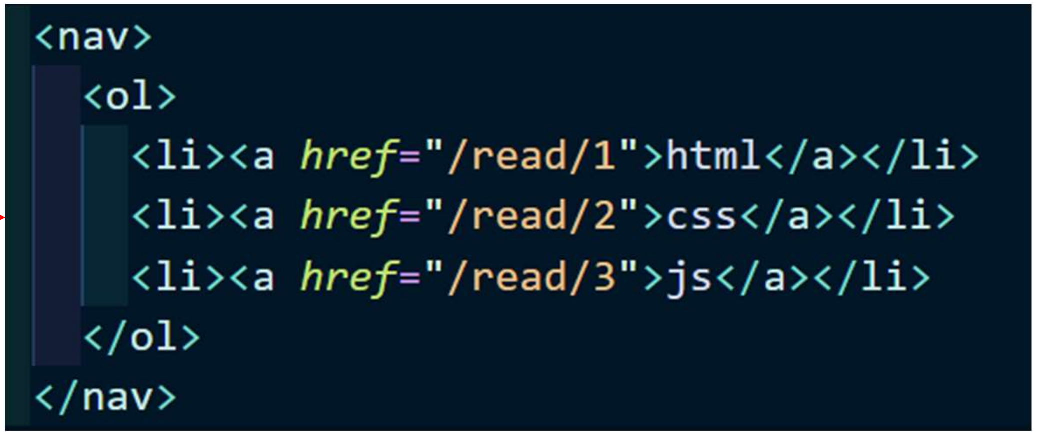

WEB

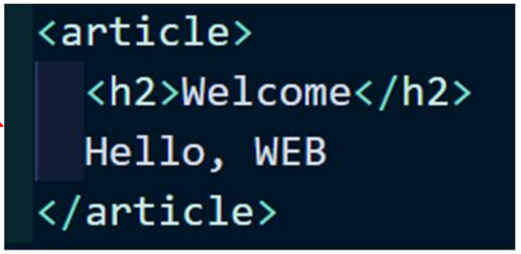
1. html
2. css
3. js

Welcome

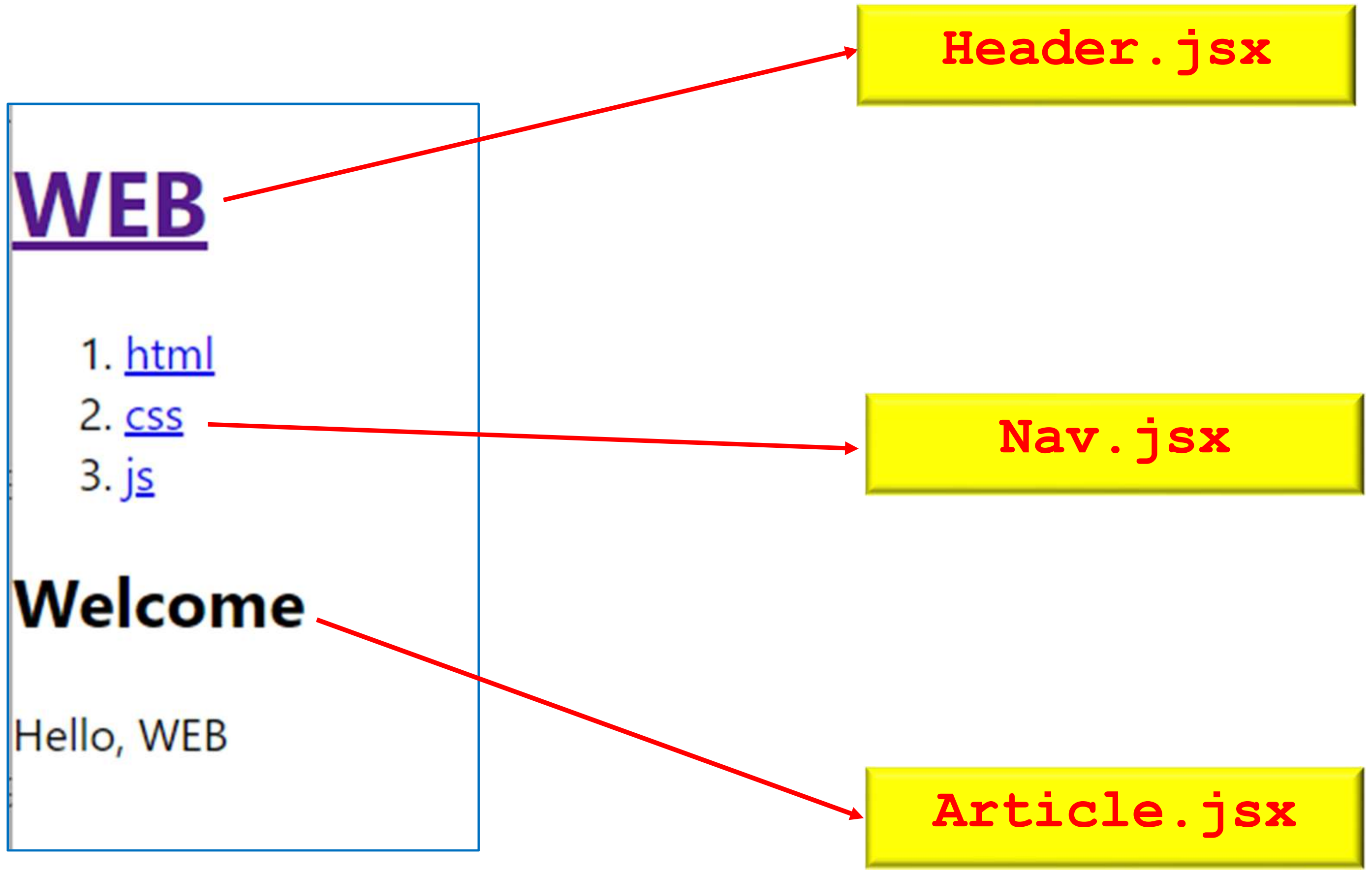
Hello, WEB

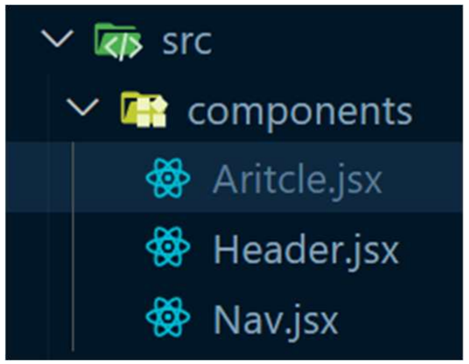

```
<div>
  <header>
    <h1><a href="/">WEB</a></h1>
  </header>
</div>
```


```
<nav>
  <ol>
    <li><a href="/read/1">html</a></li>
    <li><a href="/read/2">css</a></li>
    <li><a href="/read/3">js</a></li>
  </ol>
</nav>
```

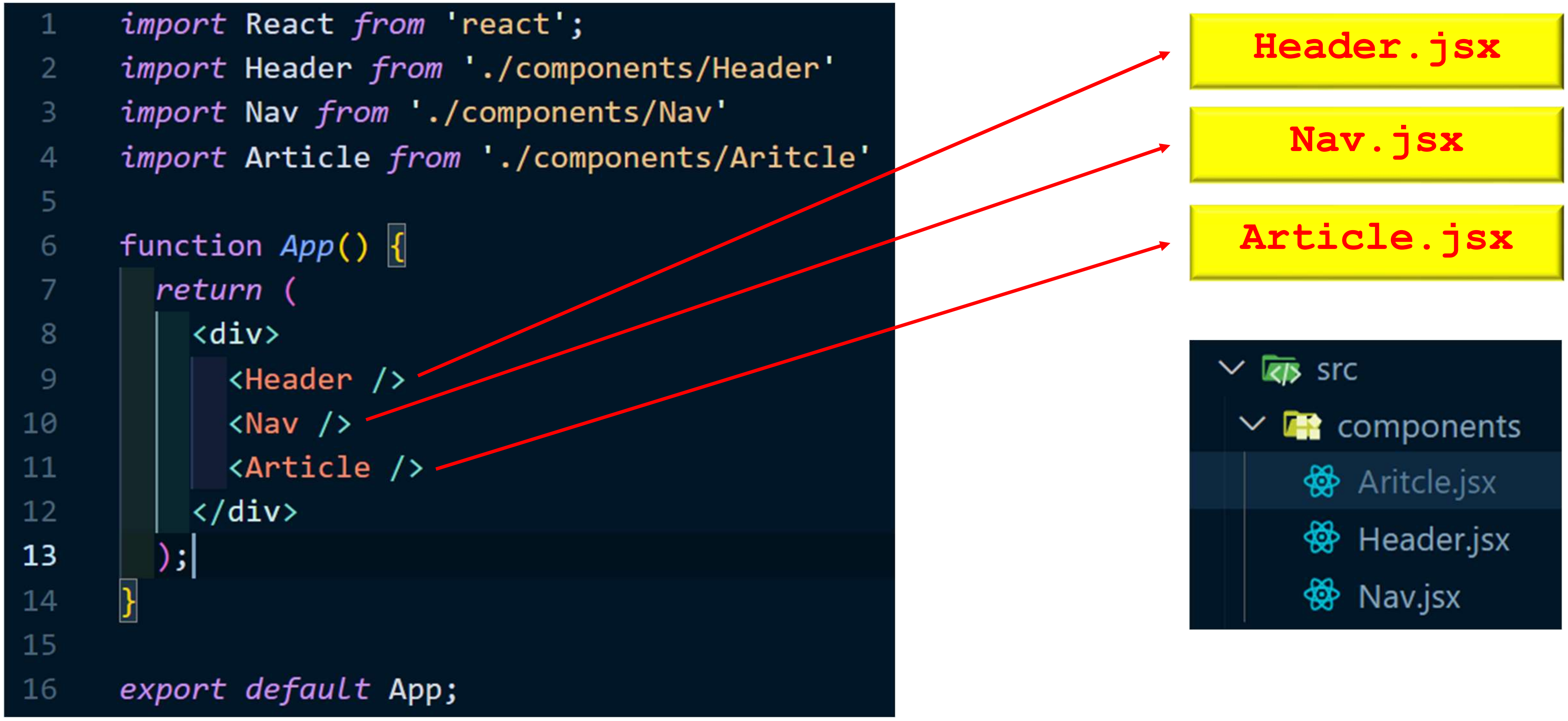

```
<article>
  <h2>Welcome</h2>
  Hello, WEB
</article>
```

Components (Cont.)





Components (Cont.)



```
import React from 'react';
import Header from './components/Header'
import Nav from './components/Nav'
import Article from './components/Aritcle'

function App() {
  return (
    <div>
      <Header />
      <Nav />
      <Article />
    </div>
  );
}

export default App;
```

Header.jsx

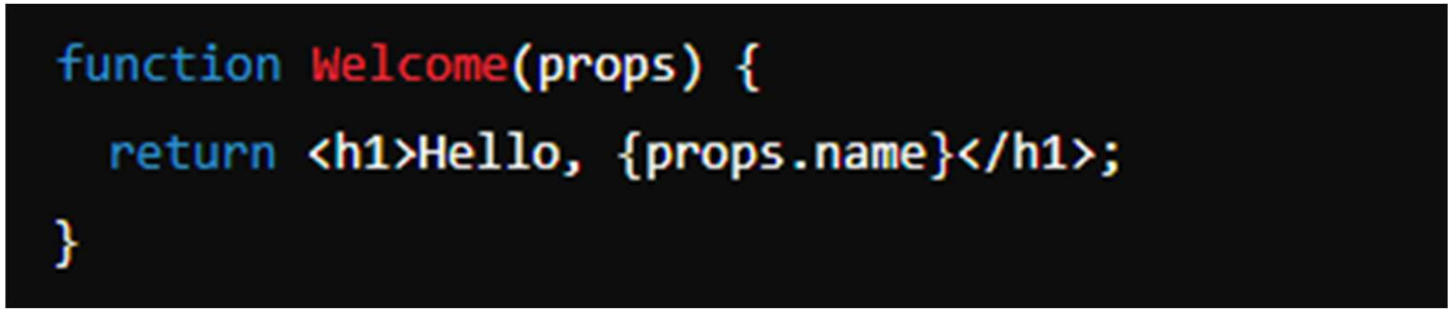
Nav.jsx

Article.jsx

Lab. Components

Components (Cont.)

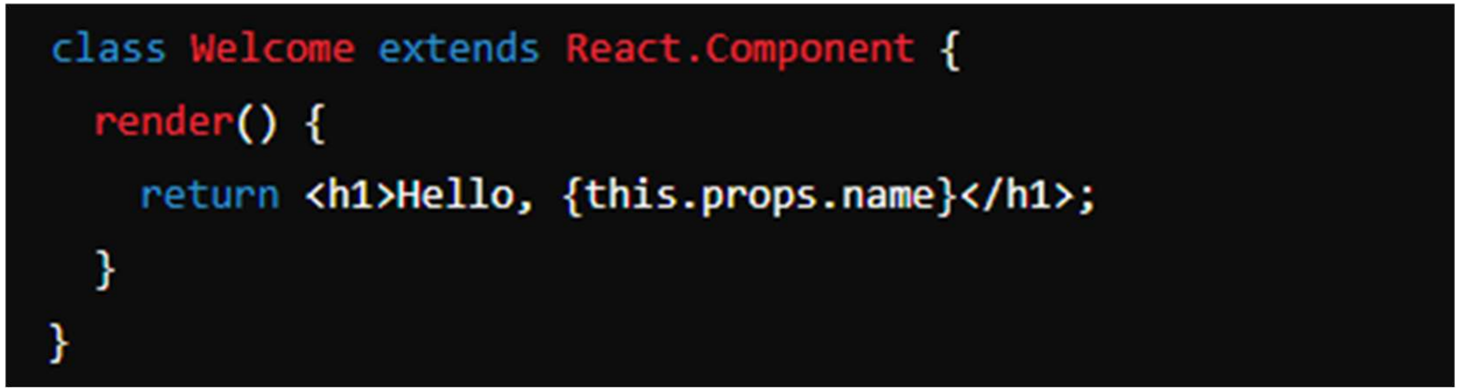
- Types of React Components
  - Function Components
    - Are simple JavaScript functions that return JSX.
    - Are ideal for components that do not require state or lifecycle methods.



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

Components (Cont.)

- Types of React Components
  - Class Components
    - Are ES6 classes that extend `React.Component`.
    - Can hold and manage state and have access to lifecycle methods.



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

Props

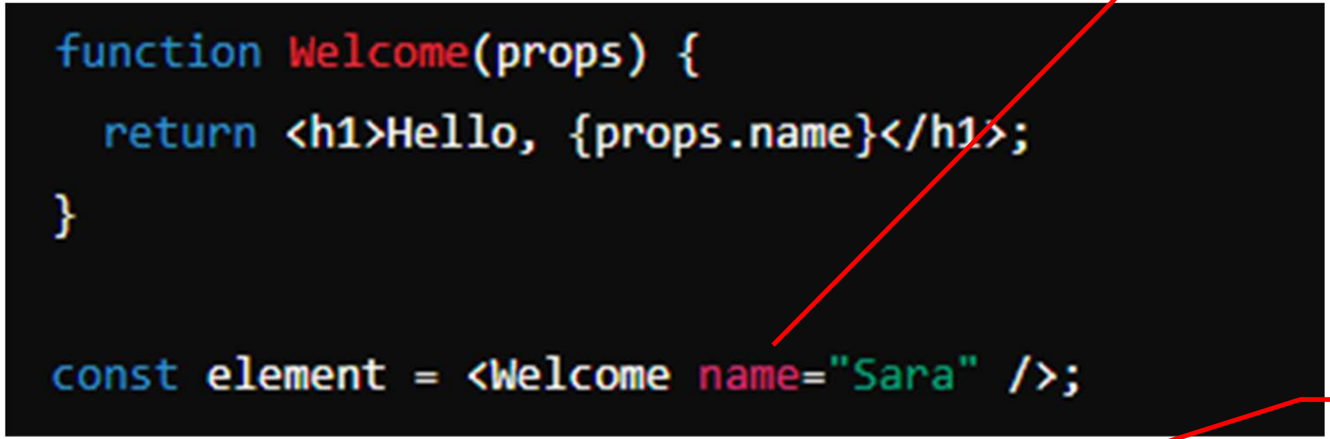
- Short for ***properties***.
- Are a fundamental concept used to pass data from one component to another, typically from a parent component to a child component.
- Are read-only and allow components to be dynamic and reusable by accepting dynamic data inputs.

Props (Cont.)

- Key Characteristics of Props
  - Read-Only
    - Cannot be modified by the receiving component.
    - Are immutable within the child component, ensuring a one-way data flow from parent to child.
  - Passed from Parent to Child
    - Are used to pass data and event handlers down the component tree.
  - Customizable
    - Can be any type of data: strings, numbers, objects, functions, arrays, and more.

Props (Cont.)

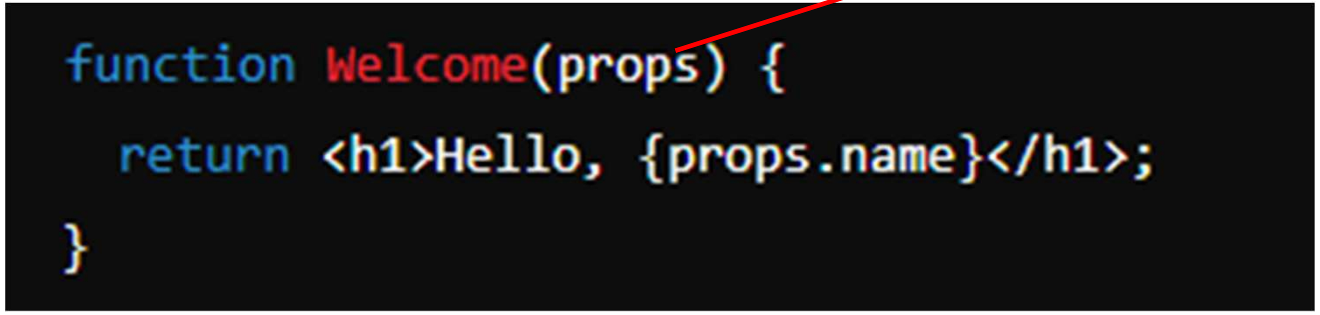
- Defining and Passing Props



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

const element = <Welcome name="Sara" />;
```

Props are passed to a component similar to HTML attributes.

- Accessing Props

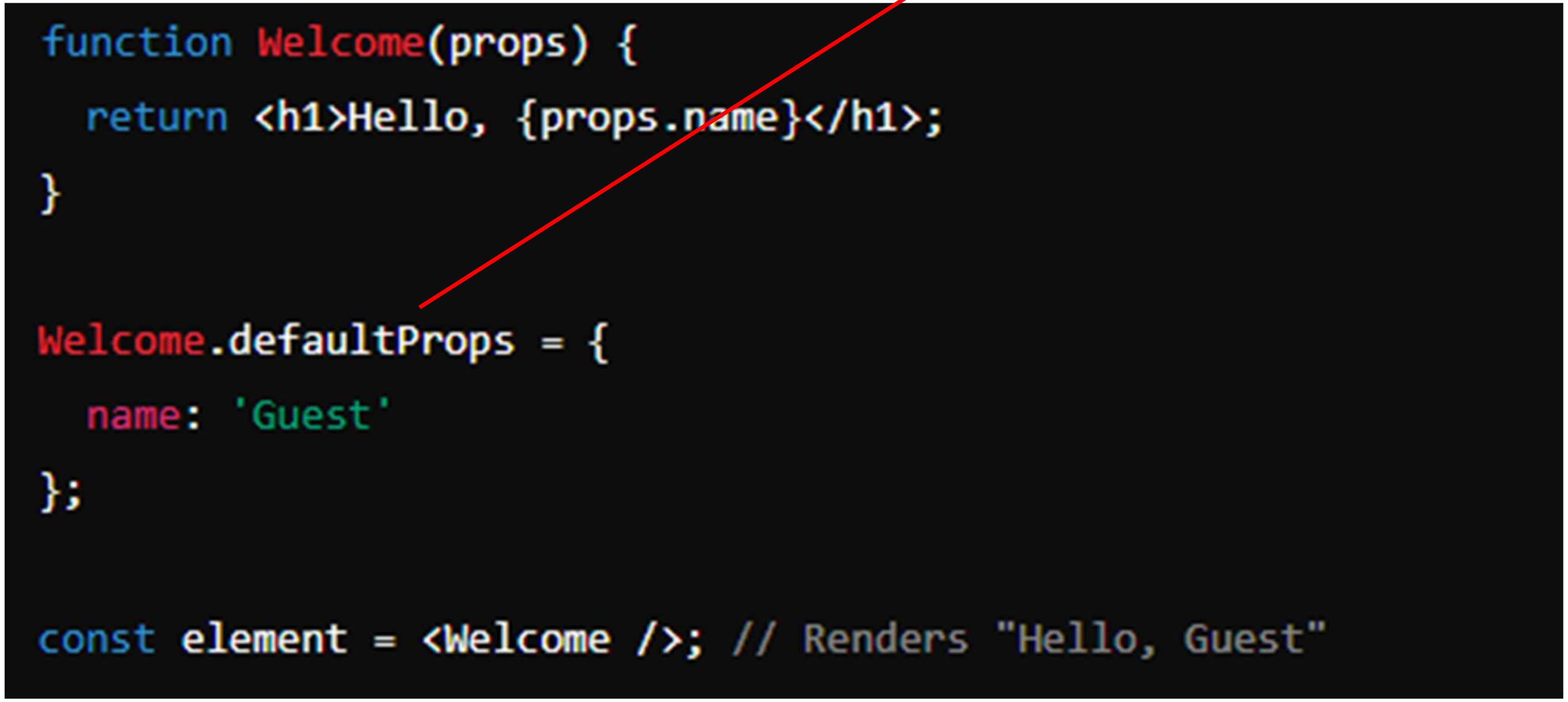


```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

Props are accessed in a component using props object.

Props (Cont.)

- Default Props

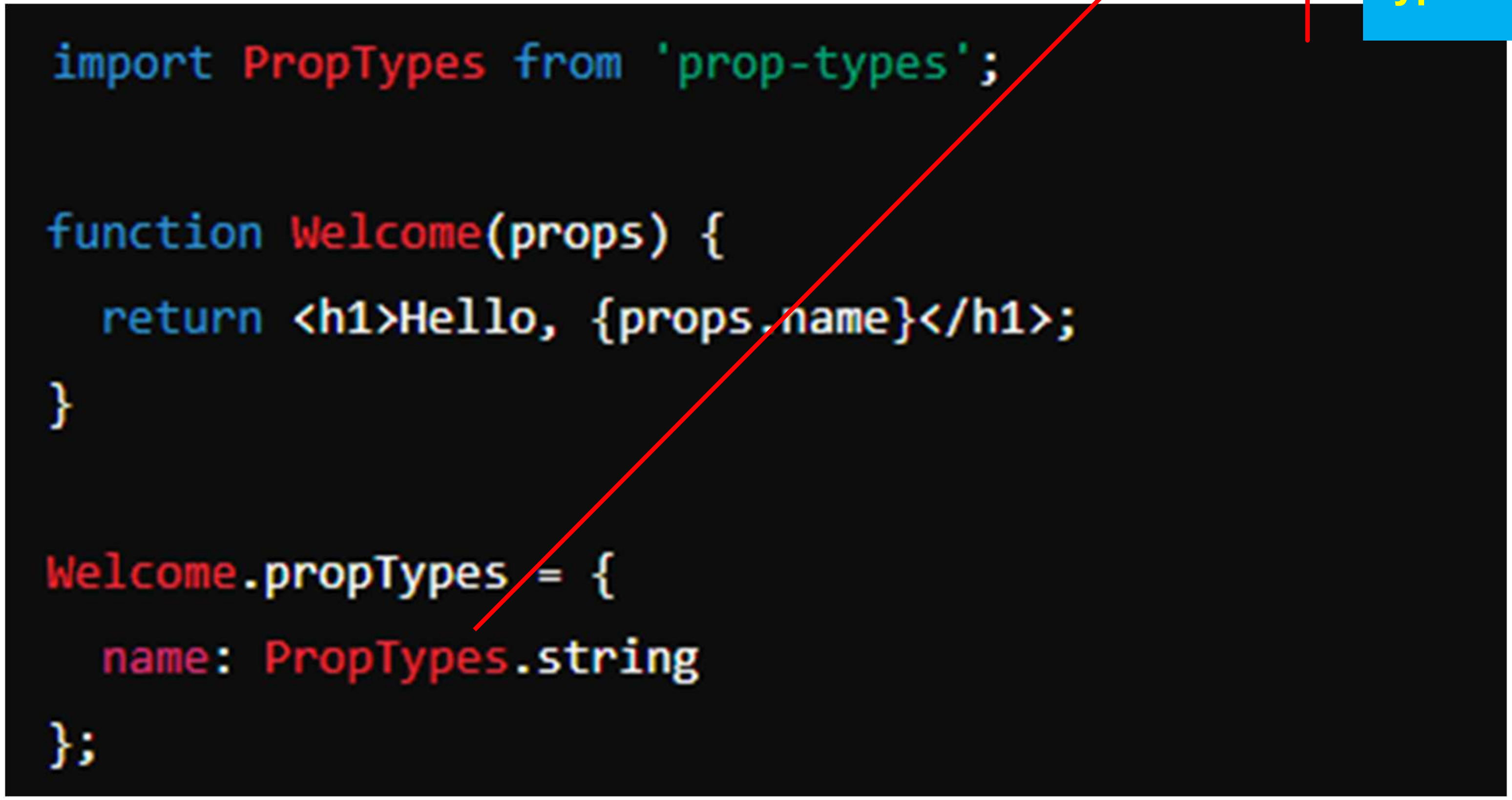


```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

Welcome.defaultProps = {
  name: 'Guest'
};

const element = <Welcome />; // Renders "Hello, Guest"
```

Can define default values for props in case they are not provided.

Props (Cont.)

- Props Types



```
import PropTypes from 'prop-types';

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

Welcome.propTypes = {
  name: PropTypes.string
};
```

Using PropTypes, can enforce type checking on props to ensure they are of the correct type.

Lab. Props

Lab. Event

State

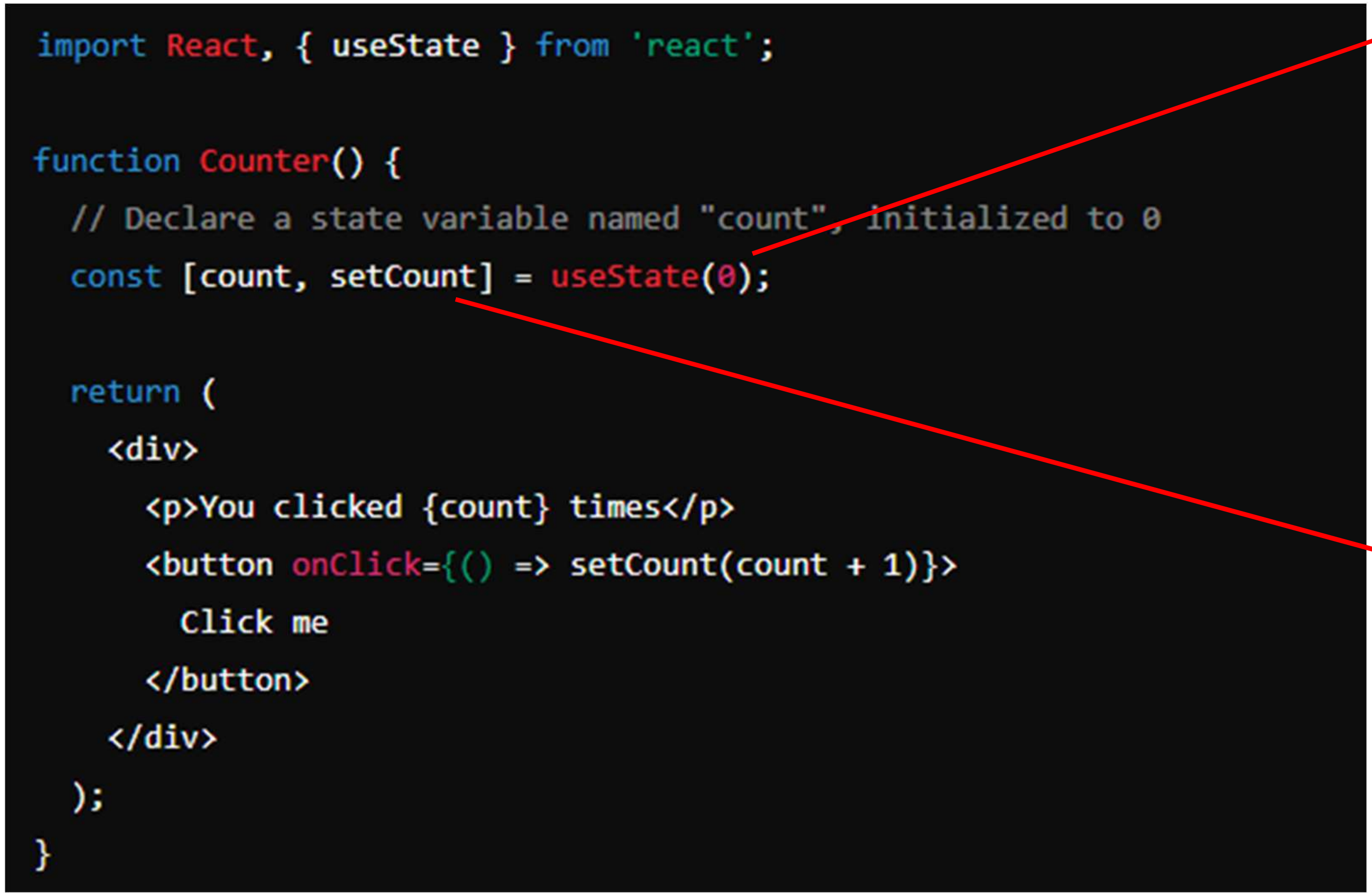
- Is a built-in object that allows components to create and manage their own data.
- Unlike ***props***, which are passed to a component and are immutable.
- Is managed within the component and can change over time.
- Is particularly useful for dynamic data that needs to be tracked and updated as a user interacts with the application.

State (Cont.)

- Key Characteristics of State
  - Mutable
    - Can be changed, allowing React components to react and update based on user interactions or other factors.
  - Local to the Component
    - Each component can have its own state.
    - The state of a component is private and fully controlled by the component.
  - Triggers Re-render
    - When the state of a component changes, React re-renders the component to reflect the new state.

State (Cont.)

- Function components use the **`useState`** hook to add state.



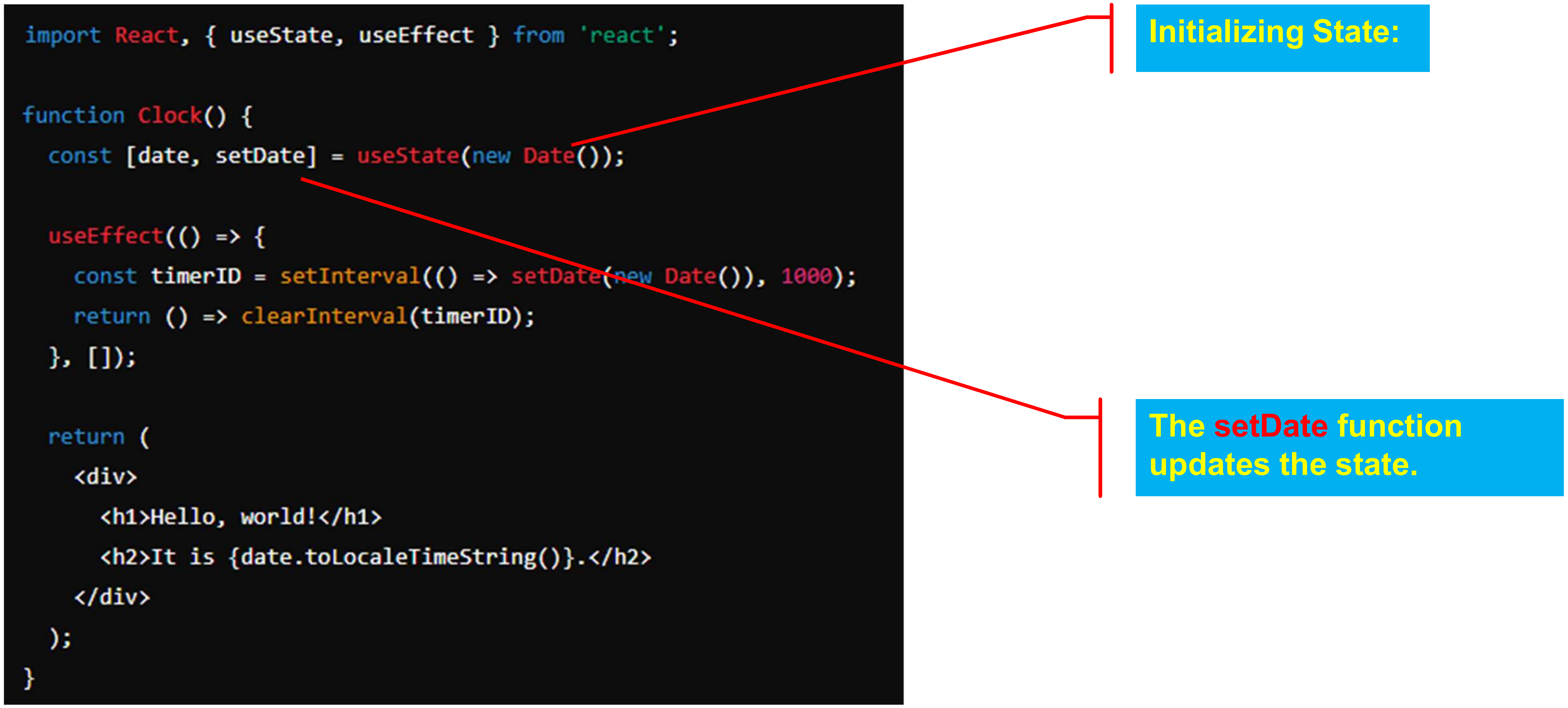
```
import React, { useState } from 'react';

function Counter() {
  // Declare a state variable named "count", initialized to 0
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Initializing State:

The setCount function updates the state. Calling setCount will re-render the component with the new state value.

State (Cont.)

- Function components use the **`useState`** hook to add state.



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

function Clock() {
  const [date, setDate] = useState(new Date());

  useEffect(() => {
    const timerID = setInterval(() => setDate(new Date()), 1000);
    return () => clearInterval(timerID);
  }, []);

  return (
    <div>
      <h1>Hello, world!</h1>
      <h2>It is {date.toLocaleTimeString()}.</h2>
    </div>
  );
}
```

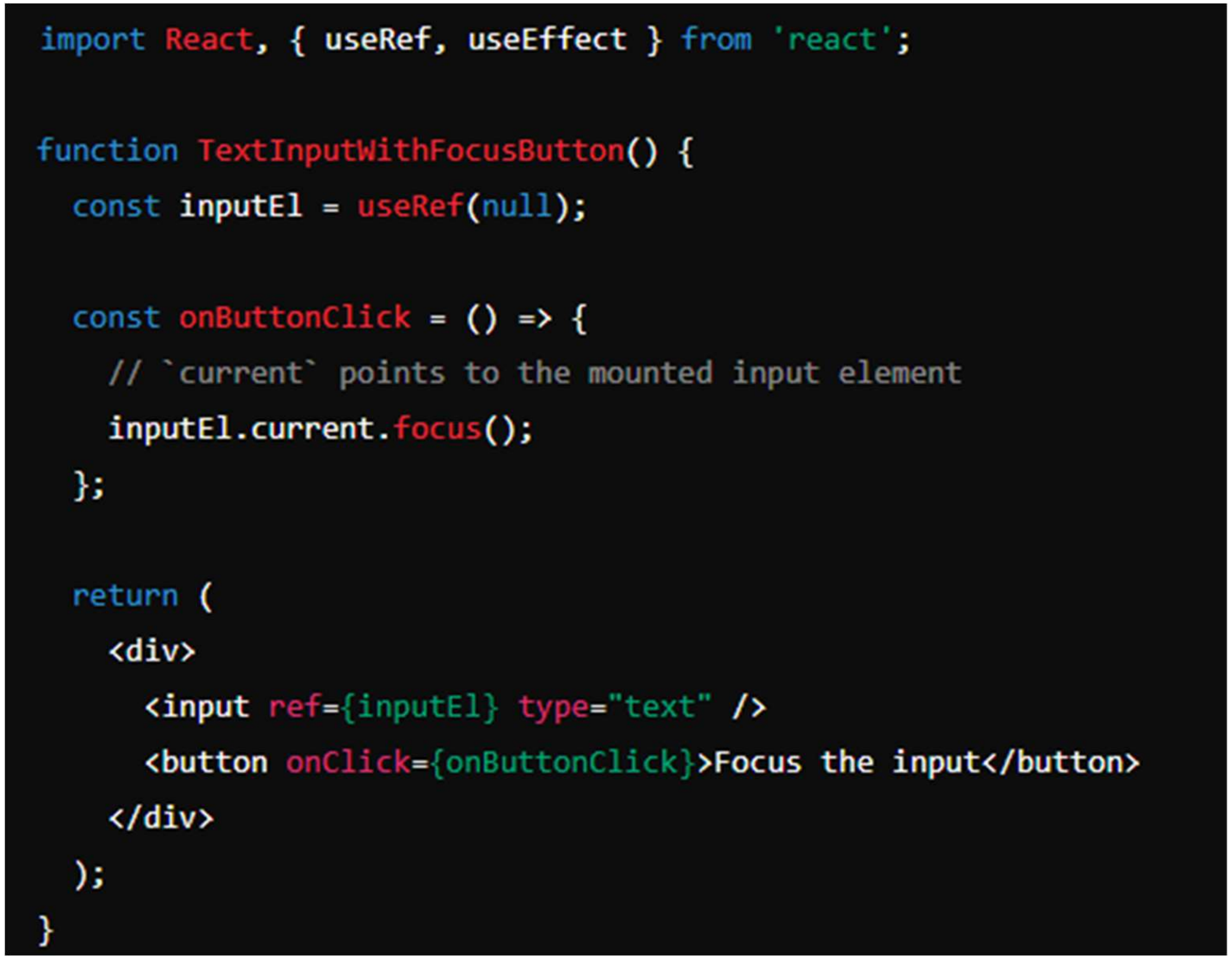
Lab. State

Lab. State & Form

useRef

- Is a hook that returns a mutable **`ref`** object whose ***.current*** property is initialized to the passed argument (***initialValue***).
- This **`ref`** object persists for the full lifetime of the component.
- Accessing DOM elements directly
  - Is commonly used to reference a DOM element directly.
  - For instance, can set focus on an input field when the component mounts.
- Persisting values
  - Unlike **`useState`**, updating a **`ref`** does not cause the component to re-render.
  - This makes **`useRef`** useful for keeping any mutable value around, like a timer ID.

useRef (Cont.)



```
import React, { useRef, useEffect } from 'react';

function TextInputWithFocusButton() {
  const inputEl = useRef(null);

  const onButtonClick = () => {
    // `current` points to the mounted input element
    inputEl.current.focus();
  };

  return (
    <div>
      <input ref={inputEl} type="text" />
      <button onClick={onButtonClick}>Focus the input</button>
    </div>
  );
}
```

Lab. useRef

Lab. Semi React App

React Lifecycle

- Refers to the sequence of events (lifecycle methods)
- Happen from the mounting of a component to its unmounting.
- Understanding the lifecycle helps in managing component behavior during its existence.
- The React component lifecycle is generally divided into three main phases:

Mounting | Updating | Unmounting

- In function-based (or functional) components, React lifecycle methods are managed using ***hooks***.
- The primary hooks that replicate the behavior of lifecycle methods in class components are **`useState`**, **`useEffect`**, **`useRef`**, and **`useContext`**.

React Lifecycle (Cont.) - Mounting

- Is the phase where a component is ***created*** and ***inserted*** into the DOM.
- When a component is ***rendered***.
- Initialization (constructor equivalent)
  - Use the **`useState`** hook to initialize state.

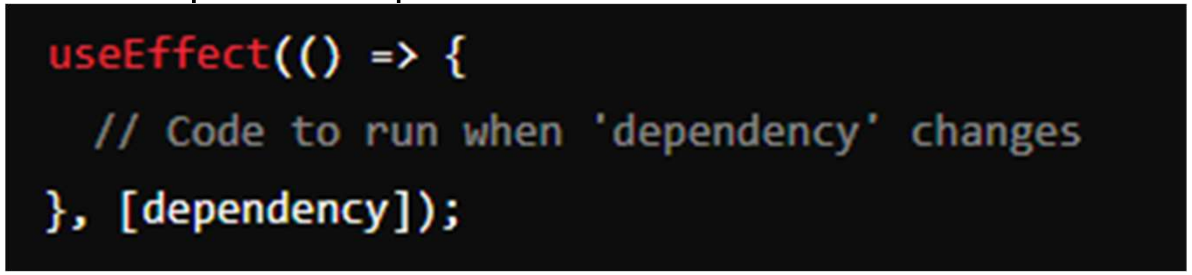
```
const [state, setState] = useState(initialState);
```

- componentDidMount
  - Use the **`useEffect`** hook with an empty dependency array **`[]`** to run code once after the initial render.

```
useEffect(() => {
  // Code to run on mount, such as fetching data
  return () => {
    // Cleanup function for component unmount
  };
}, []);
```

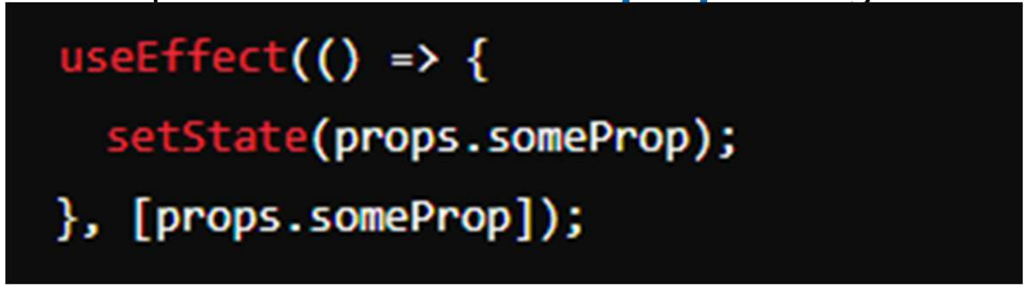
React Lifecycle (Cont.) - Updating

- Is the phase when a component is being ***re-rendered*** as a result of changes to either its ***props*** or ***state***.
- componentDidUpdate
  - Use the **`useEffect`** hook with specific dependencies to run code when those dependencies change.



```
useEffect(() => {
  // Code to run when 'dependency' changes
}, [dependency]);
```

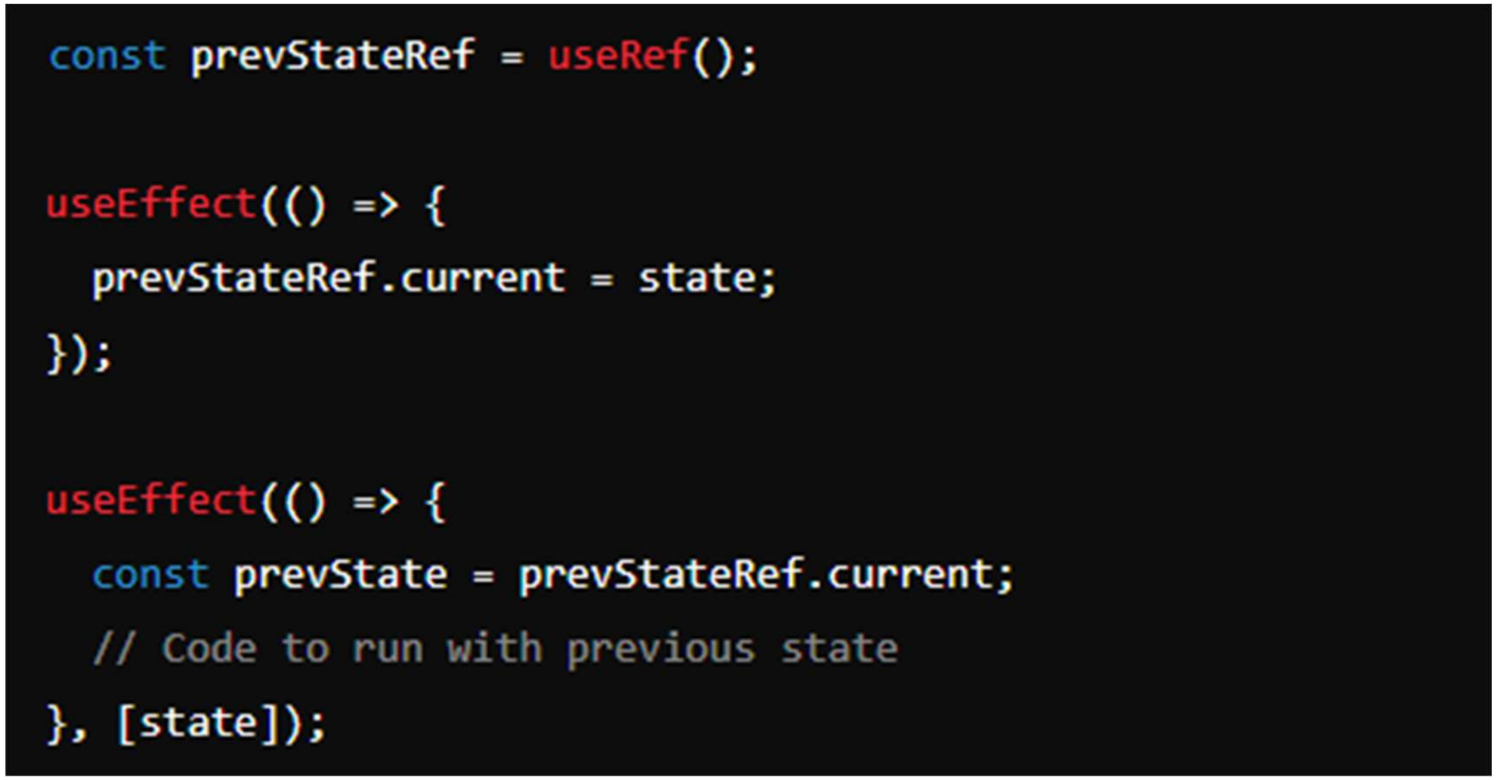
- getDerivedStateFromProps
  - Use the **`useEffect`** hook to update ***state*** based on ***prop*** changes.



```
useEffect(() => {
  setState(props.someProp);
}, [props.someProp]);
```

React Lifecycle (Cont.) - Updating

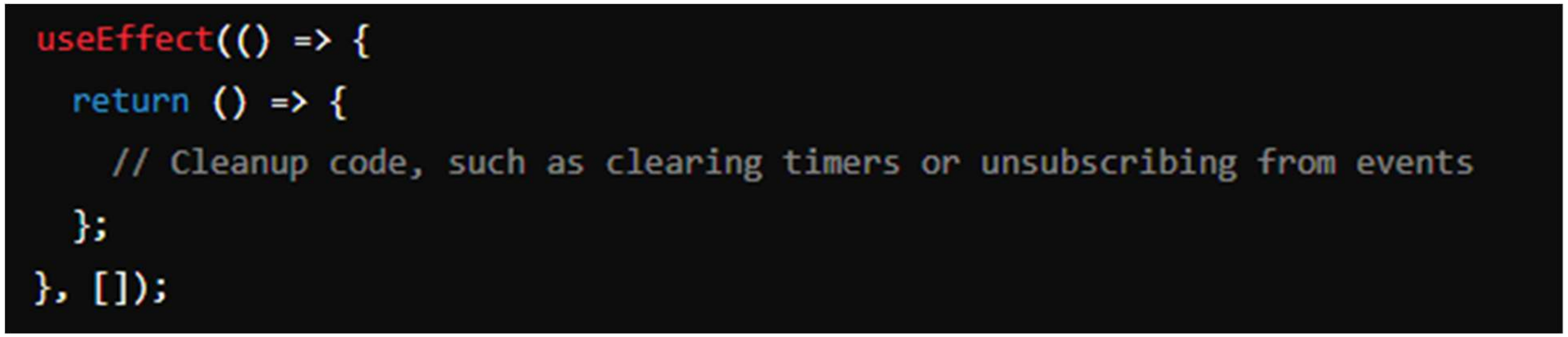
- getSnapshotBeforeUpdate
  - Use a combination of `useRef` and `useEffect` to capture some information before the DOM is updated.



```
const prevStateRef = useRef();

useEffect(() => {
  prevStateRef.current = state;
});

useEffect(() => {
  const prevState = prevStateRef.current;
  // Code to run with previous state
}, [state]);
```

React Lifecycle (Cont.) - Unmounting

- Is the phase where a component is ***removed*** from the DOM.
- componentWillUnmount
  - Return a cleanup function from **`useEffect`** to run code when the component unmounts.



```
useEffect(() => {
  return () => {
    // Cleanup code, such as clearing timers or unsubscribing from events
  };
}, []);
```

React Lifecycle (Cont.) – Lifecycle Controlling

- Mounting
  - Server에서 Data를 불러오는 작업
- Updating
  - 어떤 값이 변경되었는지 Console에 출력하는 작업
- Unmounting
  - Component가 사용하던 메모리를 정리하는 작업

Lab. useEffect & Lifecycle