React

CS568 – Web Application Development I

Computer Science Department

Maharishi International University

Maharishi International University - Fairfield, Iowa

Content

- React overview
- Create the first React app
  - App.js
  - Package.json
- React Element
- JSX
- React Component

What is React?

React is a JavaScript **library** for building **user interfaces**.

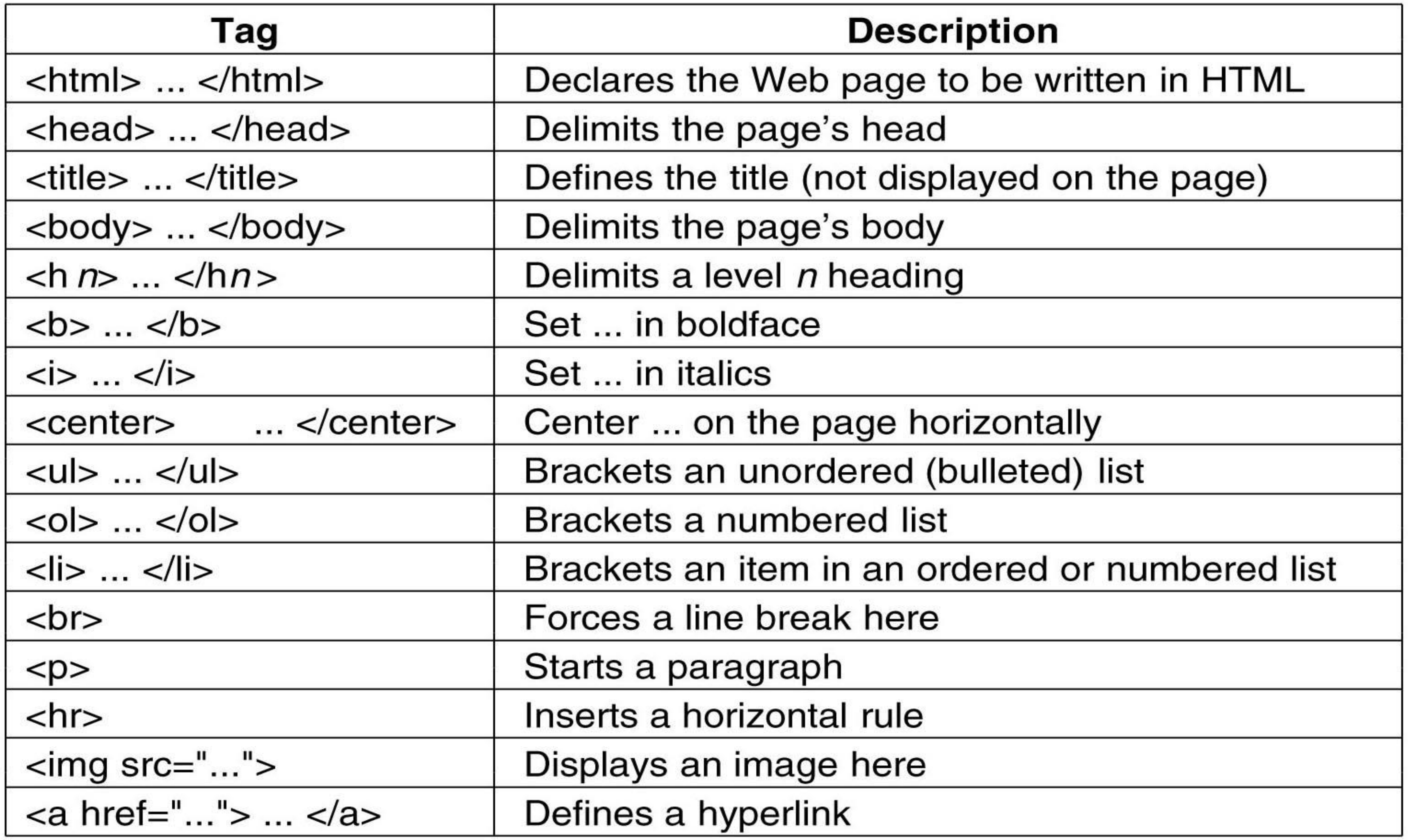
One of the most popular libraries, with over 100,000 stars on GitHub. React is an open-source project created by Facebook. React is used to build user interfaces (UI) on the front end.

React is not a framework unlike Angular. There is NextJS which is a React framework. React utilizes **component-based programming model** which is a trending front-end programming model. Once you learned React, you can learn any other component-based libraries and frameworks easily such as Flutter, Vue, etc.

Framework provides more features and is more comprehensive that you get all pieces to build your applications from one place. Whereas in React, you have to use many other libraries with React to build one app.

Prerequisite to learn React is HTML, JS, and DOM.

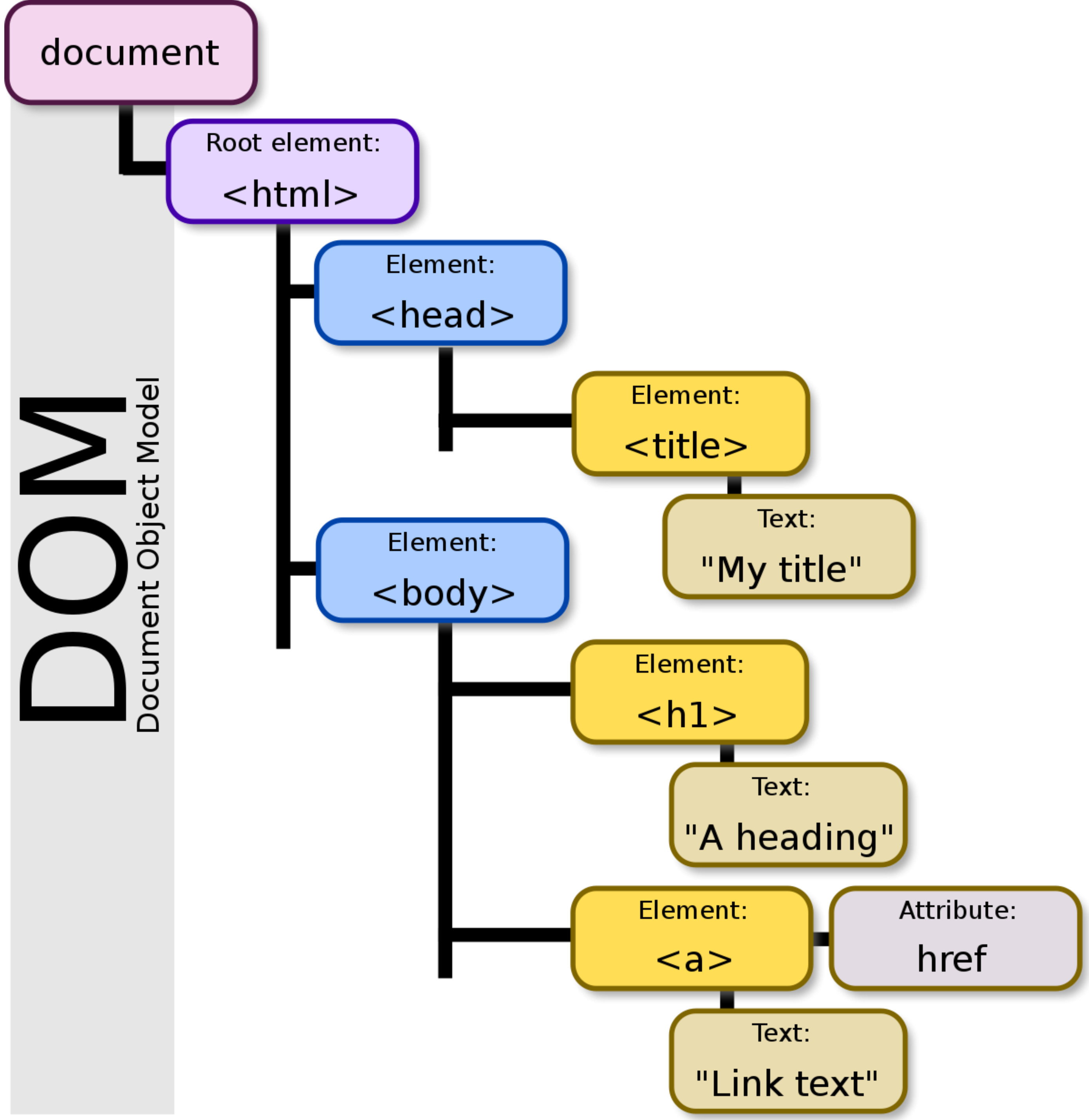
You should know some basic HTML docs



Tag	Description
<html> ... </html>	Declares the Web page to be written in HTML
<head> ... </head>	Delimits the page's head
<title> ... </title>	Defines the title (not displayed on the page)
<body> ... </body>	Delimits the page's body
<h *n*> ... </h*n*>	Delimits a level *n* heading
<b> ... </b>	Set ... in boldface
<i> ... </i>	Set ... in italics
<center> ... </center>	Center ... on the page horizontally
<ul> ... </ul>	Brackets an unordered (bulleted) list
<ol> ... </ol>	Brackets a numbered list
<li> ... </li>	Brackets an item in an ordered or numbered list
<br>	Forces a line break here
<p>	Starts a paragraph
<hr>	Inserts a horizontal rule
<img src="...">	Displays an image here
<a href="..."> ... </a>	Defines a hyperlink

DOM

- Represents HTML as a tree. Each node is an object. You can manipulate DOM using JS to update HTML pages.
- Manipulating HTML elements through DOM is slower. CSS also has a tree structure. Updating these entire DOM and CSS trees is slow. React updates only the required portion needed to be updated. It doesn’t redraw the DOM tree.
- Bottom line is, make DOM changes as less as possible for better performance. React is more performant than jQuery (it was so popular before React) because it does less DOM updates.


DOM
Document Object Model
document
Root element:
<html>
Element:
<head>
Element:
<title>
Text:
"My title"
Element:
<body>
Element:
<h1>
Text:
"A heading"
Element:
<a>
Attribute:
href
Text:
"Link text"

Important Files in a React app created with create-react-app

- **App.js**: This is the file for App Component. App Component is the main component in React which acts as a container for all other components.
- **Package.json**: This File has the list of node dependencies which are needed.
- **Package-lock.json**: In package.json, you only define dependencies you need. Those dependencies also have dependencies. The package-lock stores all dependencies needed to run your app with **the exact verions**. Exact versions are very important. If versions mismatch, your app doesn’t even start!
- **README.md** – App doc. Better write if you have time.
- **node_modules** – Actual library code. You can regenerate with npm install. So **don’t push it to git**. Include it in gitignore file.
- public/**index.html** – Your app is rendered in this html.
- **Index.js** – imported in the index.html. Calls the app.js or the entry point. If you instantiate a variable here, that will be available across your app.

React Elements

```
import React from "react";

export default function App() {
  return React.createElement(
    "div",
    null,
    React.createElement(
      "p",
      { className: "App" },
      "Hello World. This is my first React App."
    )
  );
}
```

React.createElement()

It needs at least 3 arguments (component, props, ...children)

- The element we want to render to DOM
- Properties or an object for configuration
- Children

Configuration – Use camelCase naming standard:

- id
- className
- style

JSX

JSX just provides syntactic sugar for the React.createElement function. It is NOT a HTML. It is javascript!

```
function App() {
 return (
   <div className="App">
       <p>
         Hello World. This is my first React App.
       </p>
   </div>
 );
}
```

JSX

- User-Defined Components Must Be Capitalized.
- When an element type starts with a lowercase letter, it refers to a built-in elements like <div> or <span> and results in a string 'div' or 'span' passed to React.createElement
- Must return one parent item. Not more than one.

Embedding Expressions in JSX

Use curly bracket to refer a variable or call a function.

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

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Returning Multiple Elements

Wrap components and other HTML elements in a div

```
function App() {
 return (
    <div className="App">
        <p>
          Hello World. This is my first React App.
        </p>
        <p>
          It is fun !!!
        </p>
    </div>
 );
}
```

Returning Multiple Elements

I always use empty opening and closing tags for Fragment (<></>)

```
function App() {
 return (
   <Fragment>
      <p>
         Hello World. This is my first React App.
       </p>
       <p>
         It is fun !!!
       </p>
   </Fragment>
 );
}
```

Fragment motivation

Fragments let you group a list of children without adding extra nodes to the DOM.

```
class Table extends React.Component {
    render() {
      return (
        <table>
          <tr>
            <Columns />
          </tr>
        </table>
      );
    }
  }
```

Error without Fragment

```
class Columns extends React.Component {
    render() {
      return (
        <div>
          <td>Hello</td>
          <td>World</td>
        </div>
      );
    }
  }
```

```
<!-- result -->
<table>
  <tr>
    <div>
      <td>Hello</td>
      <td>World</td>
    </div>
  </tr>
</table>
```

Solution with Fragment

```
render() {
    return (
      <>
        <td>Hello</td>
        <td>World</td>
      </>
    );
  }
}
```

```
<!-- result -->
<table>
  <tr>
    <td>Hello</td>
    <td>World</td>
  </tr>
</table>
```

React Components

- React separates concerns with loosely coupled units called “components” that contain both the markup (HTML) and logic (JS).
- Components let you split the UI into independent, **reusable** pieces.
- Components are “made of” elements.
- There are 2 types of components:
  - Functional – Stateless, dumb, presentational. Preferred.
  - Class – Stateful, smart, containers. Should override render() method
- Component is just like a function.
  - Single responsibility
  - If code is repeated, you make it a function and call from many places.
  - If code gets bigger, you decompose it into smaller functions. So it more readable and maintainable. That is exactly how you design apps in React. I never ask or force students to read links in the slide. But this is the only link you must read, Thinking in React.

Component is nothing but a function that returns JSX

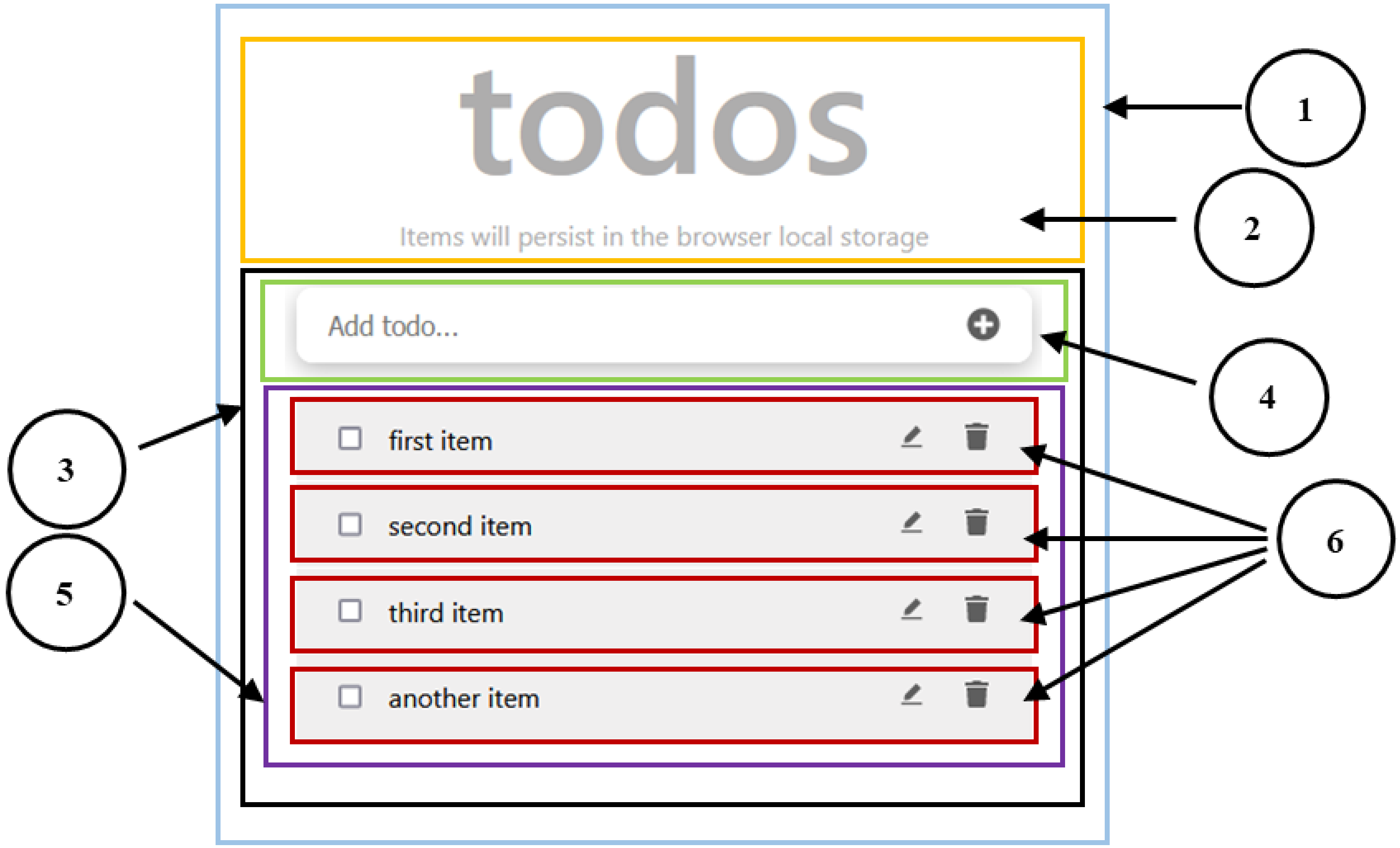
Describing the UI

Components are reusable and nestable. To reuse them, you need to export and import the components (or functions).

From web sites to phone apps, everything on the screen can be broken down into components.

For more: Describing the UI

Another example of designing components



https://ibaslogic.com/react-tutorial-for-beginners/

Functional Components

- 90% cleaner code than class components.
- Class components are verbose.
- Class components get compiled. The compiled code could be messy.
- More **consistent** and easier to test.

Functional and Class Components

```
function Welcome() {
  return <h1>Hello world!</h1>;
}
```

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello world!</h1>;
  }
}
```

Extracting Components

Don’t be afraid to split components into smaller components!

Split if the code gets too big or is used frequently.

```
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <img className="Avatar"
          src={props.author.avatarUrl}
          alt={props.author.name}
        />
        <div className="UserInfo-name">
          {props.author.name}
        </div>
      ...
```

Creating an Avatar component

```
function Avatar(props) {
    return (
      <img className="Avatar"
        src={props.user.avatarUrl}
        alt={props.user.name}
      />
    );
  }
```

Including the Avatar component

```
function Comment(props) {
  return (
   <div className="Comment">
     <div className="UserInfo">
       <Avatar user={props.author} />
       <div className="UserInfo-name">
         {props.author.name}
       </div>
       …
```

Props

- React component is just a function that has only one parameter “prop”. Prop is an object that can have many properties. Developers directly destructure it.
- React takes advantage of the JS language, functions are the first-class citizen (functions are objects). You can include anything in the prop such as another component (function), or event handler.
- Props (properties) are arguments passed to components.
- Can be used to pass data from the parent component to child components. One way, parent to child.
- They are passed via HTML properties.
- Props are read-only!
- “Props are parameters from the parent.”

Component is a function that returns JSX with one parameter, props.

Props are just a parameter or an object that can have many attributes.

Props in Functional Component

advancedHello.js

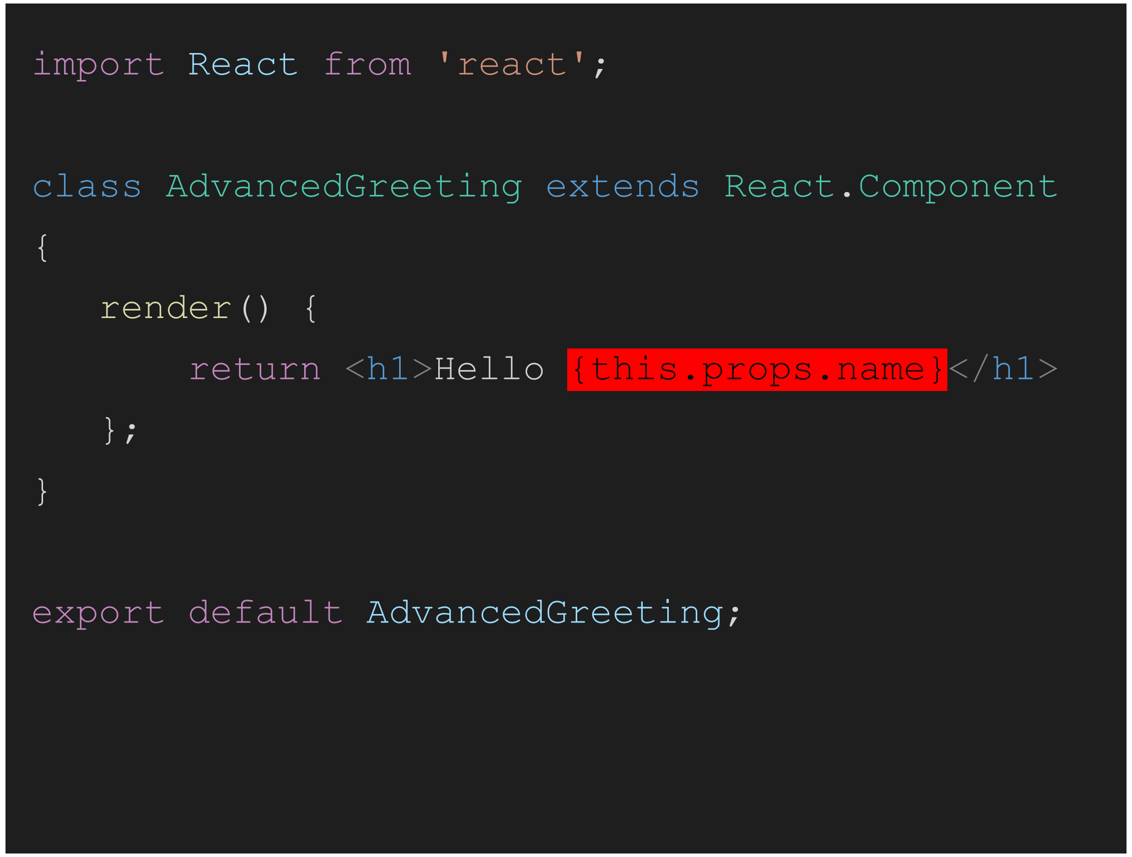
```
function AdvancedHello(props) {
   return (<h1>Hello {props.name} </h1>);
}

export default AdvancedHello;
```

App.js

```
function App() {
 return (
   <div className="App">
    <AdvancedHello name='Umur'></AdvancedHello>
    <AdvancedHello name='Unubold'></AdvancedHello>
   </div>
 );
}
```

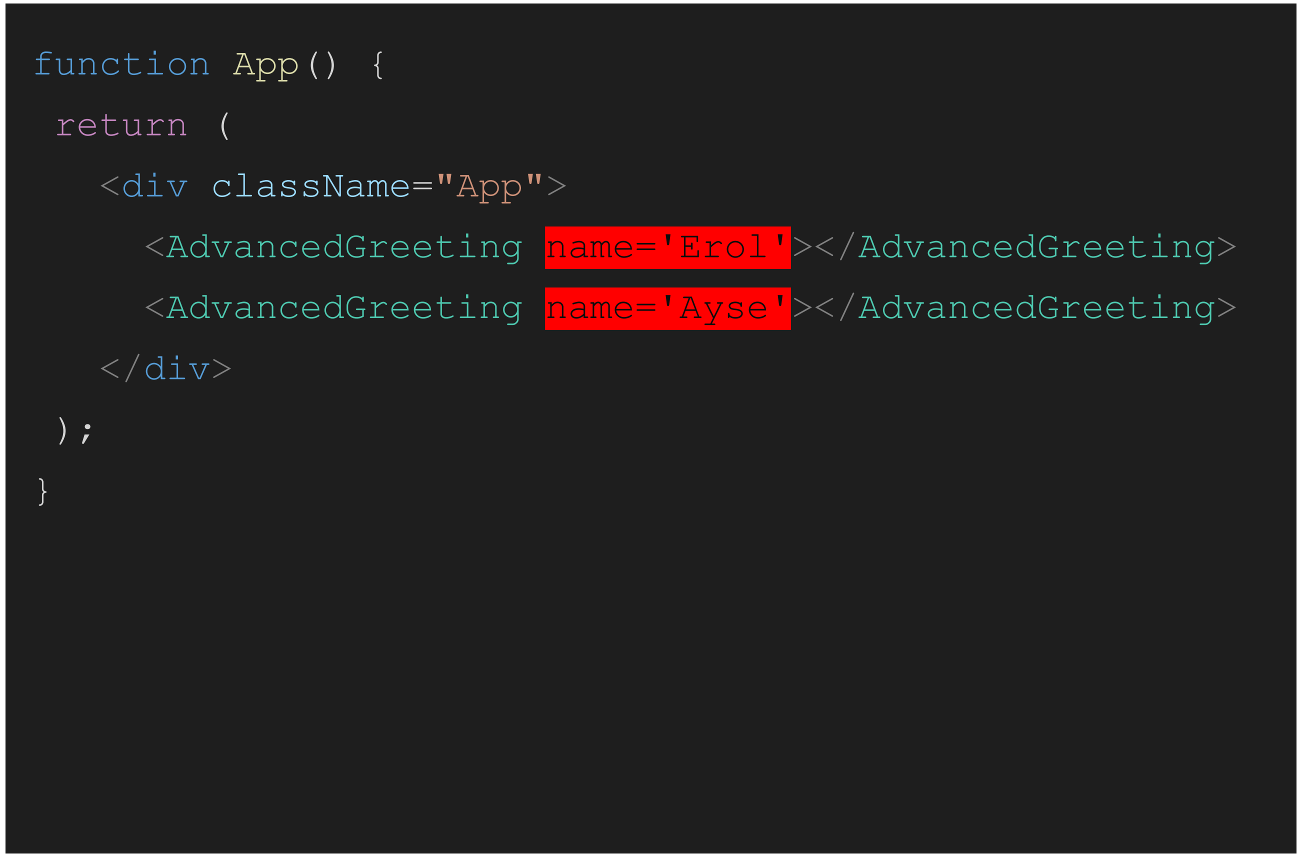
Props in Class-Based Component

advancedGreeting.js



```
import React from 'react';

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

export default AdvancedGreeting;
```

App.js



```
function App() {
 return (
   <div className="App">
     <AdvancedGreeting name='Erol'></AdvancedGreeting>
     <AdvancedGreeting name='Ayse'></AdvancedGreeting>
   </div>
 );
}
```

Children Prop

Children is a special property of React components which contains any child elements defined within the component. You can achieve **composition** (advanced technique to reuse code) using the children prop. We will see the composition at the end of block again.

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

export default AdvancedGreeting;
```

Children Prop

```
function App() {
 return (
   <div className="App">
     <AdvancedGreeting name='Erol'>How are you?</AdvancedGreeting>
     <AdvancedGreeting name='Ayse'>We all miss you</AdvancedGreeting>
   </div>
 );
}
```

Advantages of one-way data flow

- **Debugging** – One-way data flow makes debugging much easier. The developer knows where the data is coming from and where it is going.
- Less prone to errors
- More efficient

Is it possible to change the data in the parent from the child component?

Yes, pass down the function in the parent that changes the data as a prop.

Summary

- Component functions are named with PascalCase.
- You can read JSX variables by putting them between curly braces, like {so}.
- Some JSX attributes are different than HTML attributes so that they don't conflict with JavaScript reserved words. For example, class in HTML translates to className in JSX. Note that multi-word attributes are in camelCase.