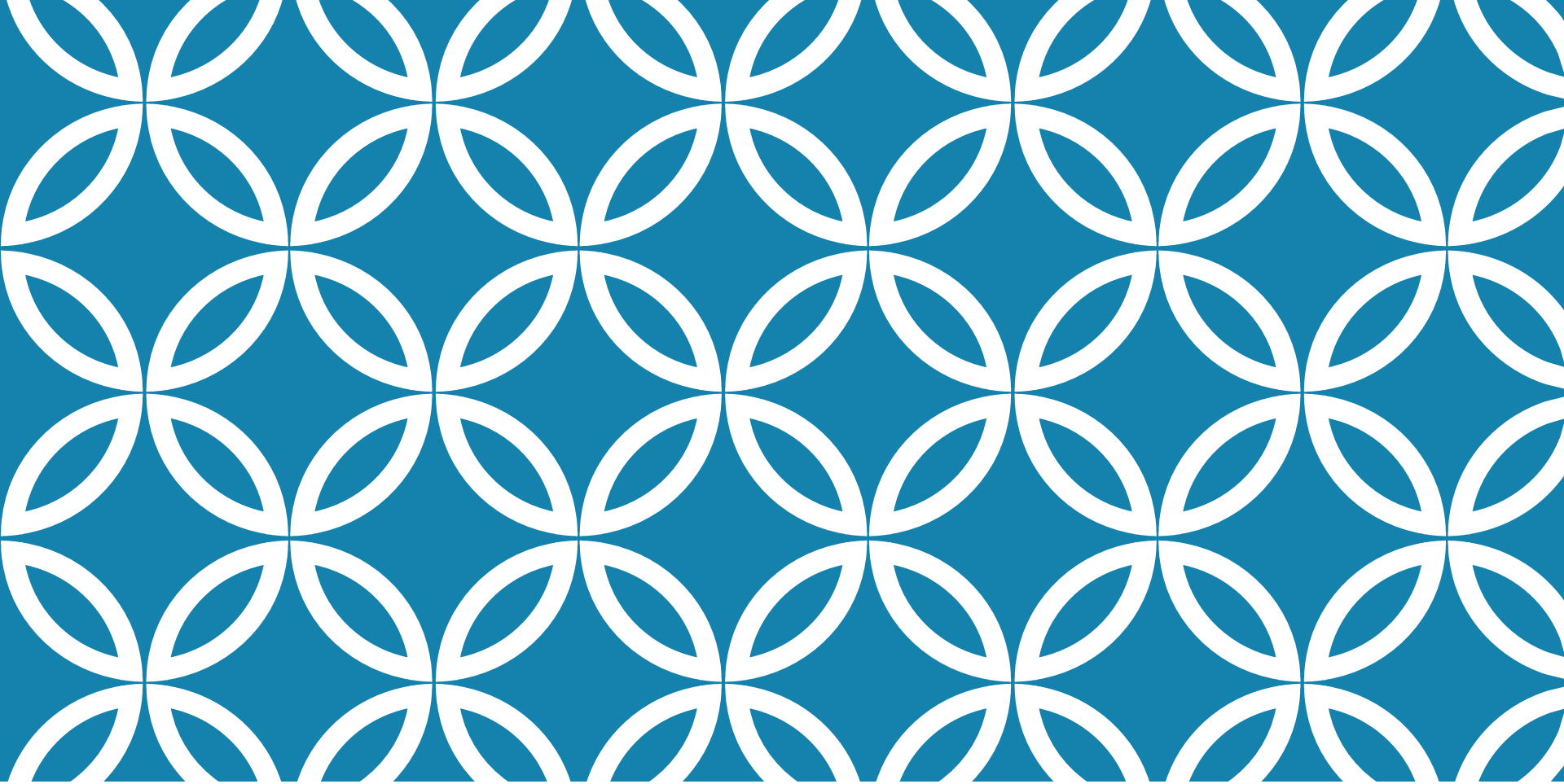

Components and Props

LEARNING OBJECTIVES

At the end of this module, the learner will be able to:

- Develop understanding on React components and props.

COMPONENTS

A component is an independent, reusable code block which divides the UI into smaller pieces.

COMPONENTS

Rather than building the whole UI under one single file, we can and we should divide all the sections (marked with red) into smaller independent pieces. In other words, these are **components**.

React has two types of components: **functional** and **class**.

FUNCTIONAL COMPONENTS

The simplest way to define a component is to write a JavaScript function:

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

This function is a valid React component because it accepts a single “props” (which stands for properties) object argument with data and returns a React element. We call such components “**function components**” because they are literally JavaScript functions.

CLASS COMPONENTS

Class components are ES6 classes that return JSX. Below, you see our same Welcome function, this time as a class component:

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

RENDERING A COMPONENT

When React sees an element representing a user-defined component, it passes JSX attributes and children to this component as a single object. We call this object "**props**".

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

const element = <Welcome name="Juan" />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

COMPONENTS

So a Functional Component in React:

- is a JavaScript/ES6 function
- must return a React element (JSX)
- always starts with a capital letter (naming convention)
- takes props as a parameter if necessary

RENDERING A COMPONENT

Note: Always start component names with a capital letter.

React treats components starting with lowercase letters as DOM tags. For example, `<div />` represents an HTML div tag, but `<Welcome />` represents a component and requires `Welcome` to be in scope.

COMPOSING COMPONENTS

Components can refer to other components in their output. This lets us use the same component abstraction for any level of detail.

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

function App() {
  return (
    <div>
      <Welcome name="EJ" />
      <Welcome name="Eliza" />
      <Welcome name="Edite" />
    </div>
  );
}

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

EXTRACTING COMPONENTS

Comment Component

```
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>
      </div>
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

Avatar Component

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

Simplified Comment Component

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

User Info Component

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

Simplified Comment Component

```
function Comment(props) {
  return (
    <div className="Comment">
      <UserInfo user={props.author} />
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

Extracting components might seem like grunt work at first, but having a palette of reusable components pays off in larger apps.

A **good rule of thumb** is that if a part of your UI is used several times (`Button`, `Panel`, `Avatar`), or is complex enough on its own (`App`, `FeedStory`, `Comment`), it is a good candidate to be extracted to a separate component.

PROPS ARE READ-ONLY

Whether you declare a component as a function or a class, it must never modify its own props.

```
function sum(a, b) {
  return a + b;
}
```

```
function withdraw(account, amount) {
  account.total -= amount;
}
```

React is pretty flexible but it has a single strict rule:
All React components must act like pure functions with respect to their props.

REFERENCE

SimpliLearn. (n.d.). ReactJS Tutorial: A Step-by-Step Guide To Learn React. https://www.simplilearn.com/tutorials/reactjs-tutorial

ReactJS Documentation: https://reactjs.org/docs/getting-started.html