REACT

component

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.

Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called props) and return React elements describing what should appear on the screen.

Functional and Class 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 object argument with data and returns a React element. We call such components 'functional' because they are literally JavaScript functions.

You can also use an ES6 class to define a component:

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

The above two components are equivalent from React's point of view.

Classes have some additional features that we will discuss in the next sections. Until then, we will use functional components for their conciseness.

Rendering a Component

Previously, we only encountered React elements that represent DOM tags:

```
const element = <div />;
```

However, elements can also represent user-defined components:

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

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

For example, this code renders 'Hello, Sara' on the page:

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

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

Try it on CodePen.

Let's recap what happens in this example:

We call ReactDOM.render() with the <Welcome name='Sara' /> element.
React calls the Welcome component with {name: 'Sara'} as the props.
Our Welcome component returns a <h1>Hello, Sara</h1> element as the result.
React DOM efficiently updates the DOM to match <h1>Hello, Sara</h1>.
Caveat:
Always start component names with a capital letter.
For example, represents a DOM 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. A button, a form, a dialog, a screen: in React apps, all those are commonly expressed as components.

times:

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

function App() {
  return (
    <div>
      <Welcome name='Sara' />
      <Welcome name='Cahal' />
      <Welcome name='Edite' />
    </div>
  );
}
```

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

Try it on CodePen.

Typically, new React apps have a single App component at the very top. However, if you integrate React into an existing app, you might start bottom-up with a small component like Button and gradually work your way to the top of the view hierarchy.

------***---***---***---***---***---***