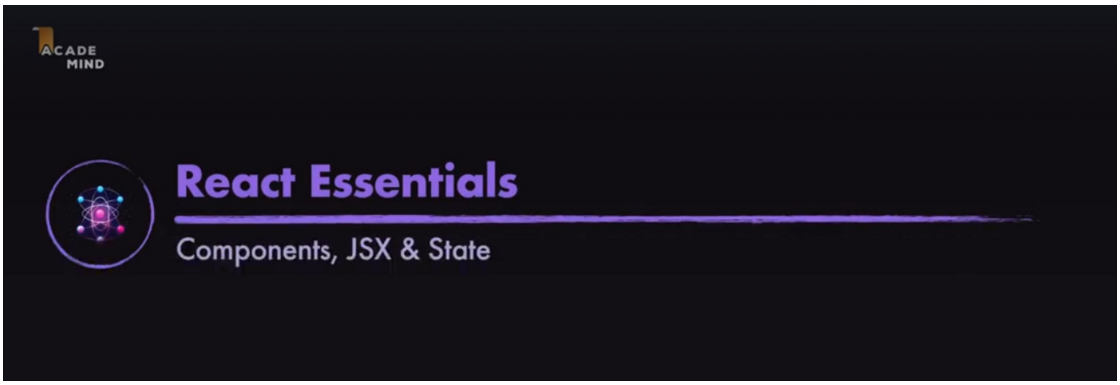
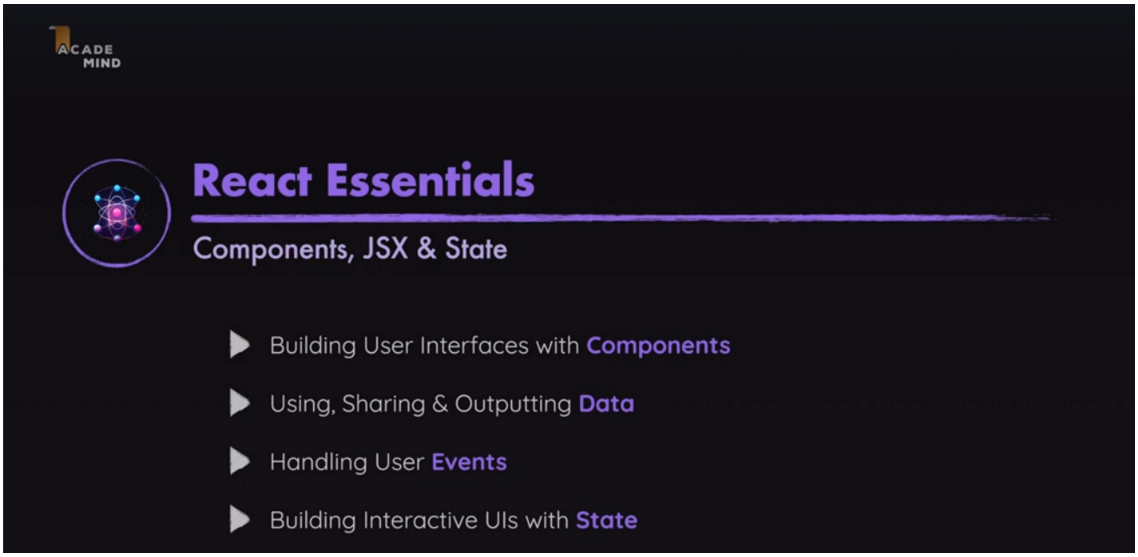
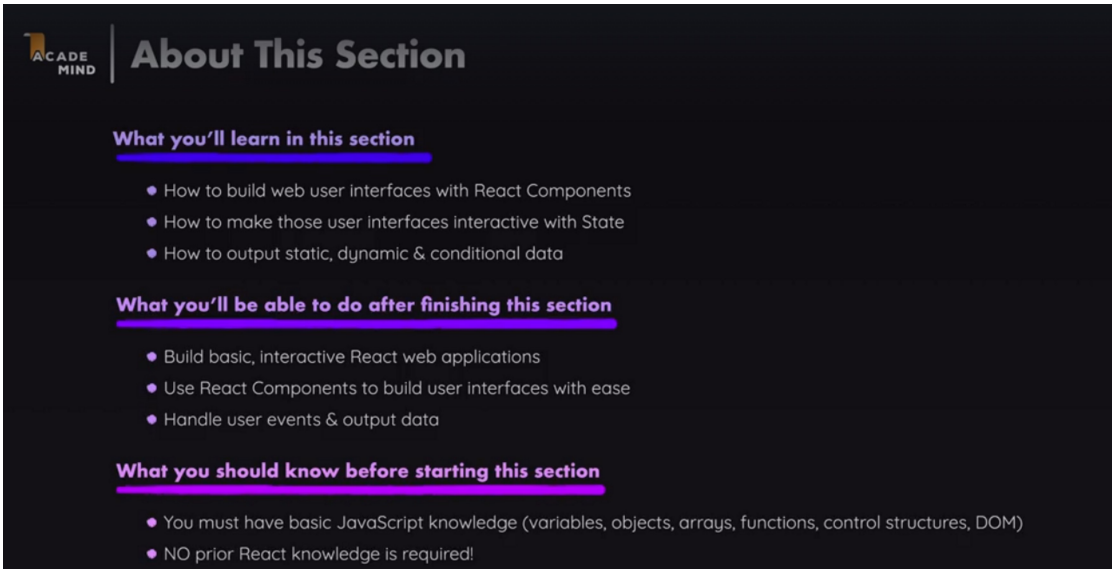
Section 3: React Essentials - Components, JSX, Props, State & More

16 September 2024 21:46

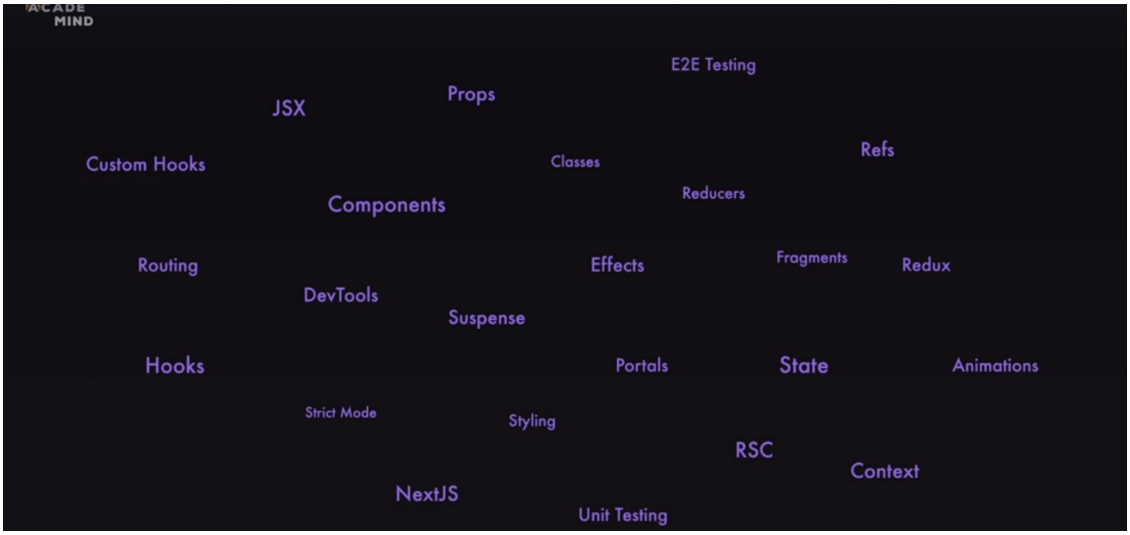




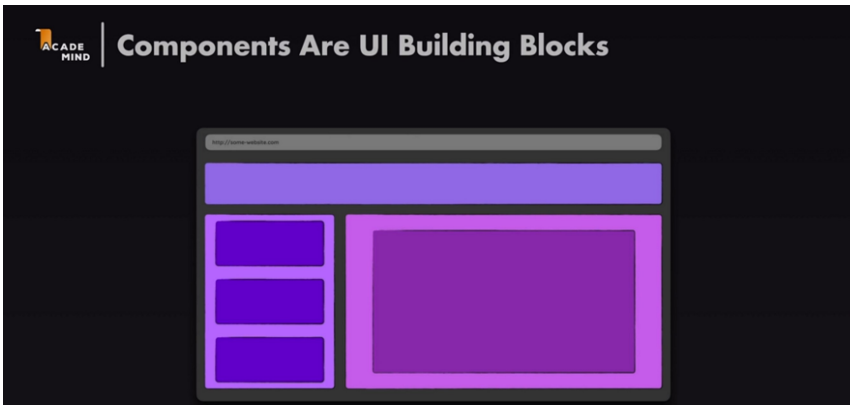


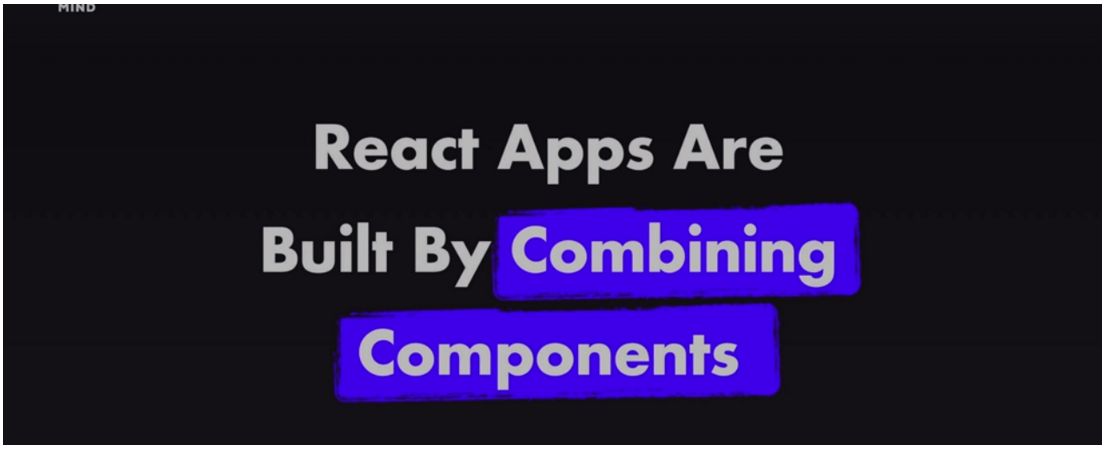
It's All About Component! (Core Concept)

React and its ecosystem provide dozens of useful and important features and concepts,
which, of course, are all going to be covered in depth throughout this course. But if you were forced to identify only one core concept which you will need for all React apps,



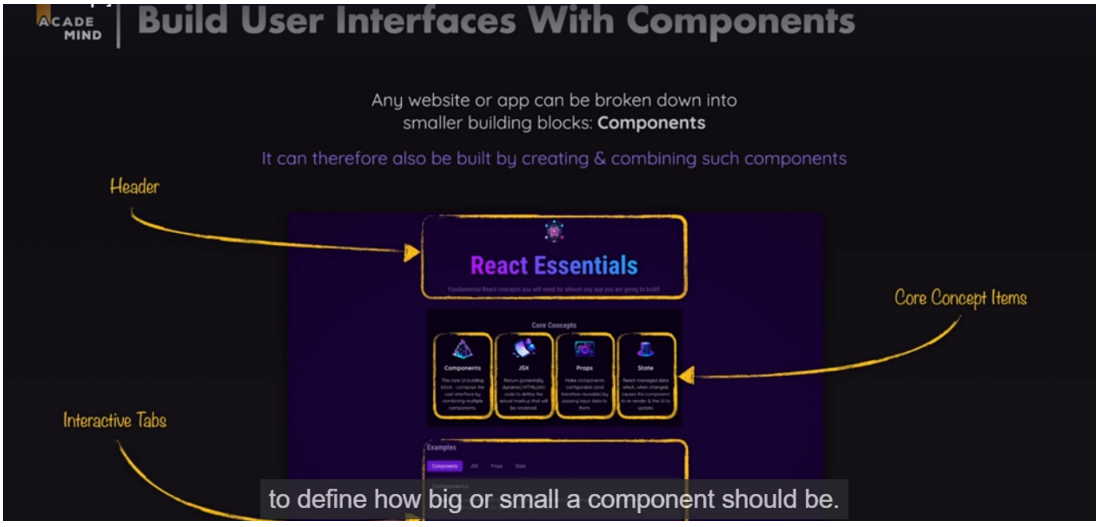
But if you were forced to identify only one core concept which you will need for all React apps,
it would be the concept of working with Components.

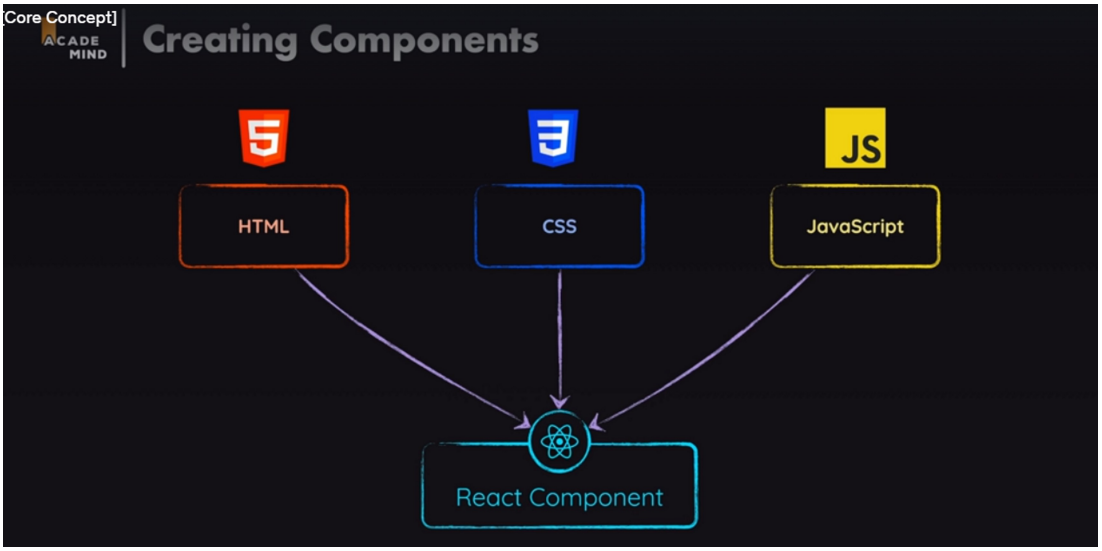




Because Components are potentially reusable building blocks which you, as a React developer, can create, and which you then combine to build the overall user interface.

So React apps are in the end built by combining Components. And whilst a couple of additional features and concepts would be required to create truly interactive user interfaces, Components are the one concept every React app, no matter its complexity, will use.

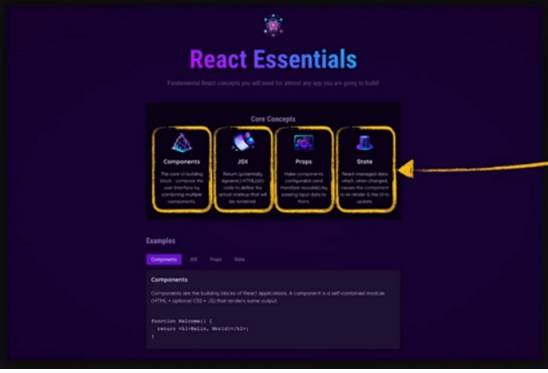




But the idea behind those Components really just is that they wrap HTML and possibly also CSS code, as well as any related JavaScript logic that might be needed. And together, those languages and code pieces
then define and control a part of the UI. This allows you, the developer, to split up complex user interfaces
into smaller, more manageable parts, which, at least potentially,

may also be reused in different places of the UI.

Components Can Potentially Be Reused



Core Concept Items

The same component is used multiple times with different input data

Why Components?

Reusable Building Blocks

Create small building blocks & compose the UI from them

If needed: Reuse a building block in different parts of the UI (e.g., a reusable button)

Related Code Lives Together

Related HTML & JS (and possibly CSS) code is stored together

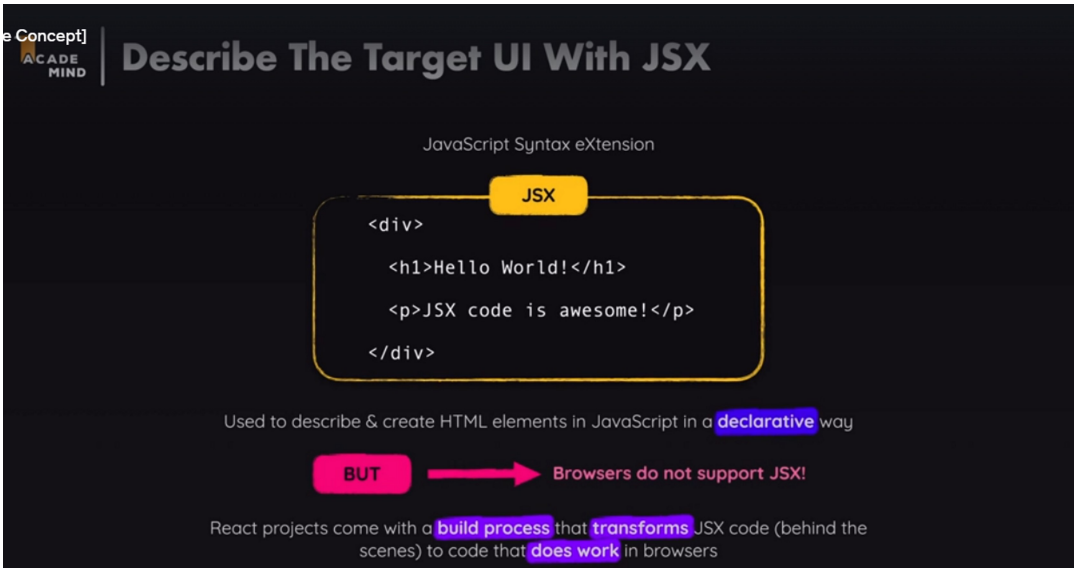
Since JS influences the output, storing JS + HTML together makes sense

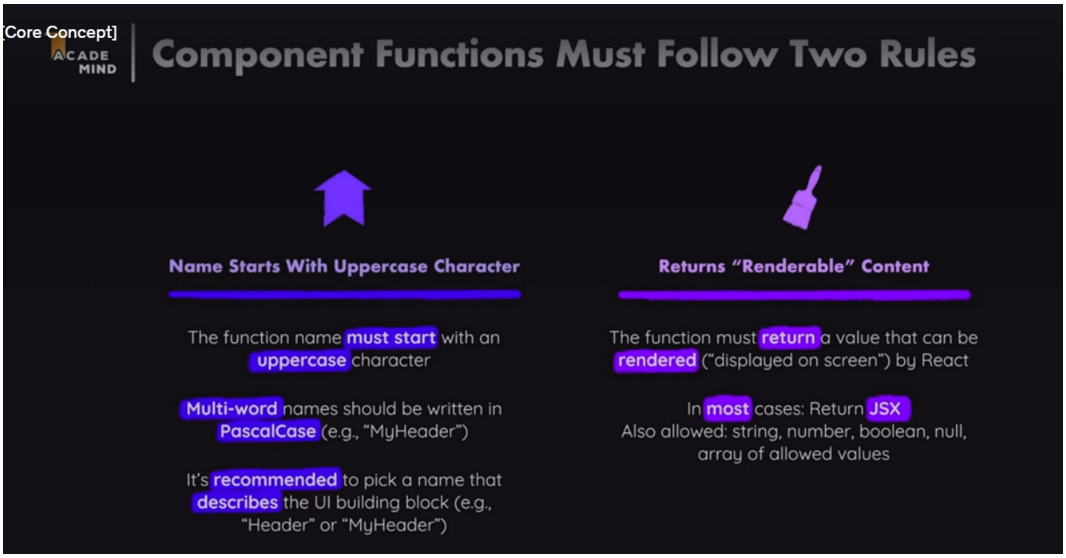
Separation of Concerns

Different components handle different data & logic

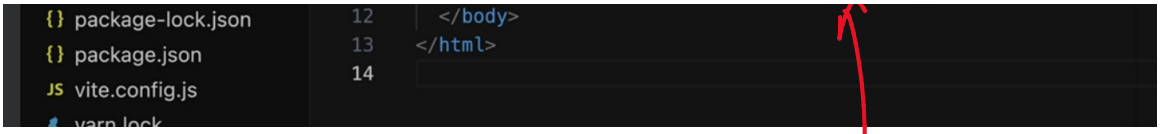
Vastly simplifies the process of working on complex apps

Components Are A Fundamental Concept

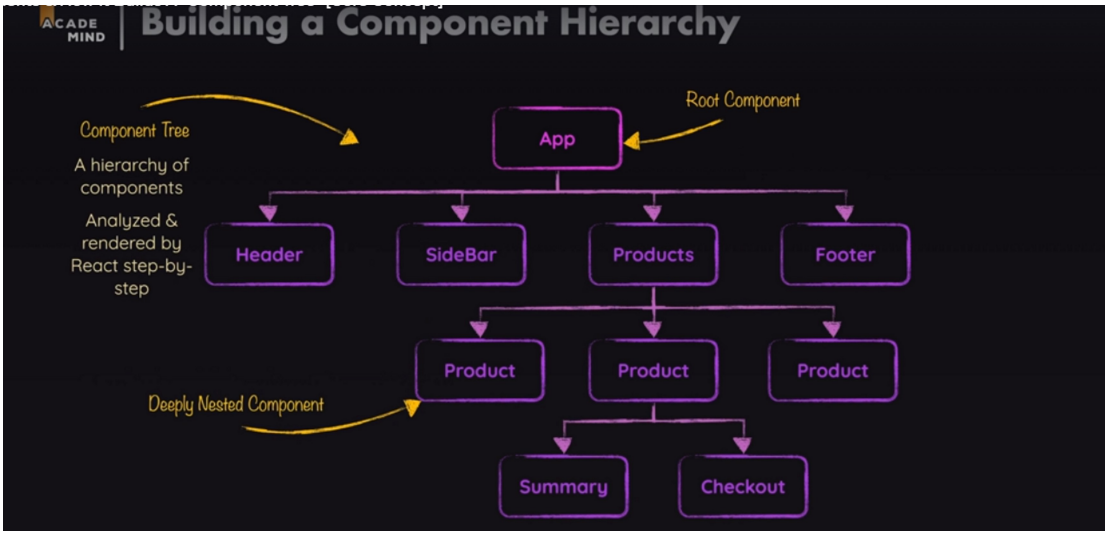




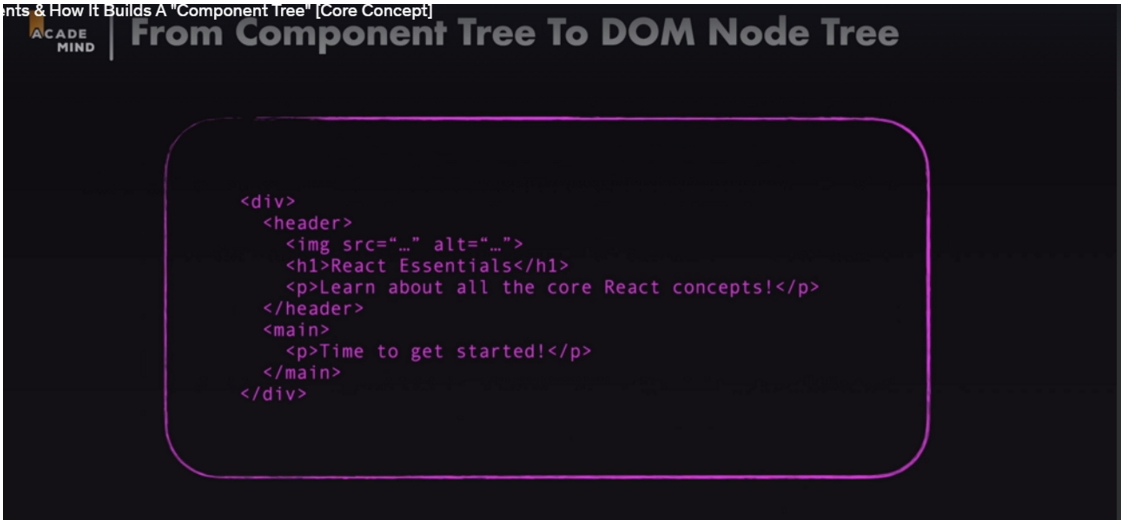
Components are just function in React



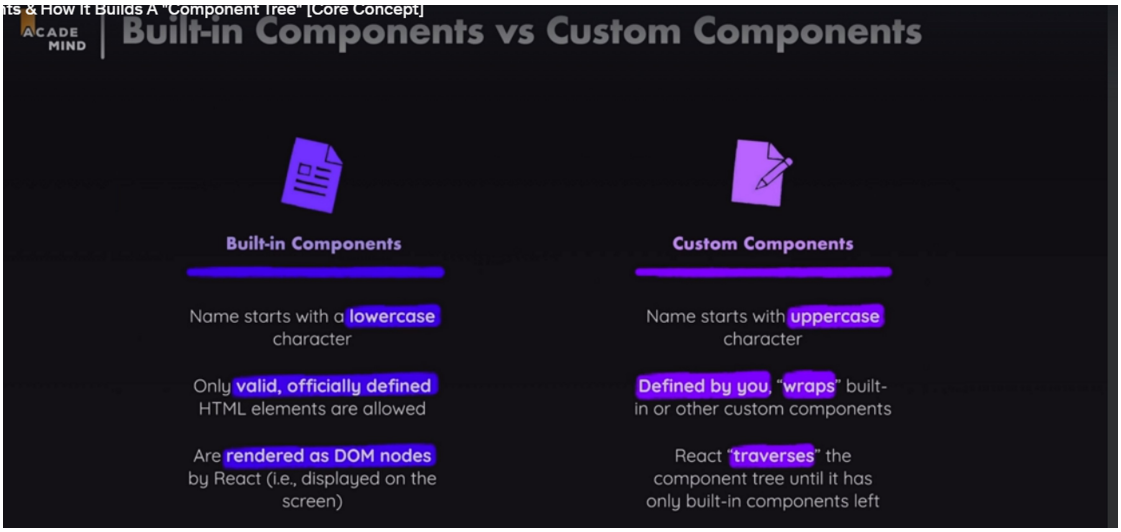

```jsx
import ReactDOM from "react-dom/client";

import App from "./App.jsx";
import "./index.css";

const entryPoint = document.getElementById("root");
ReactDOM.createRoot(entryPoint).render(<App />);

```



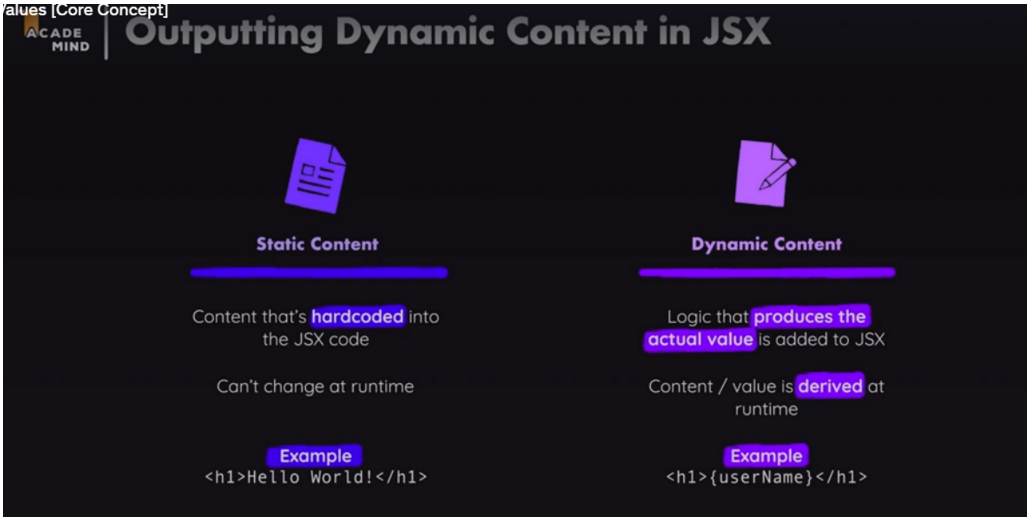
nts & How It Builds A "Component Tree" [Core Concept]

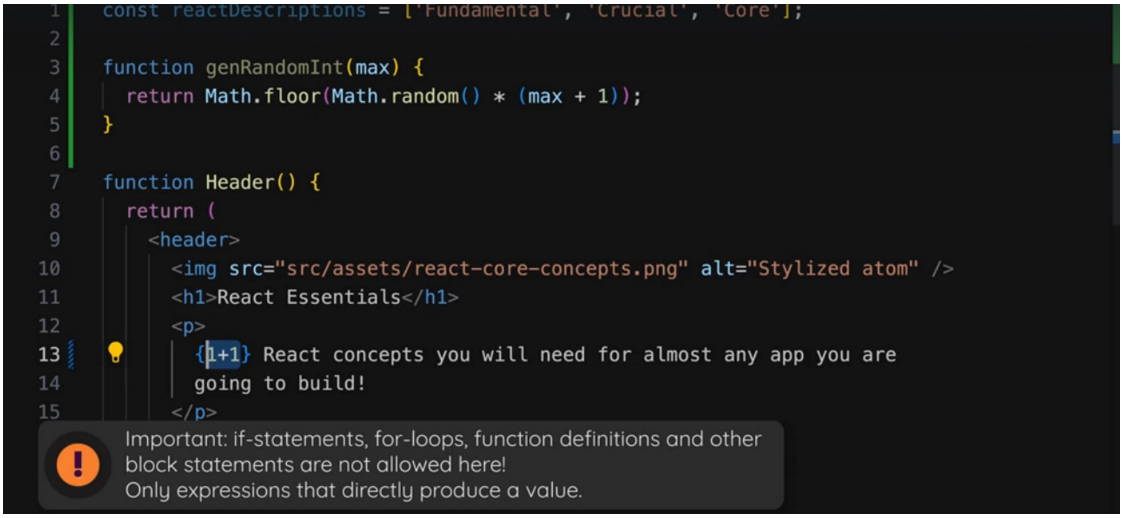


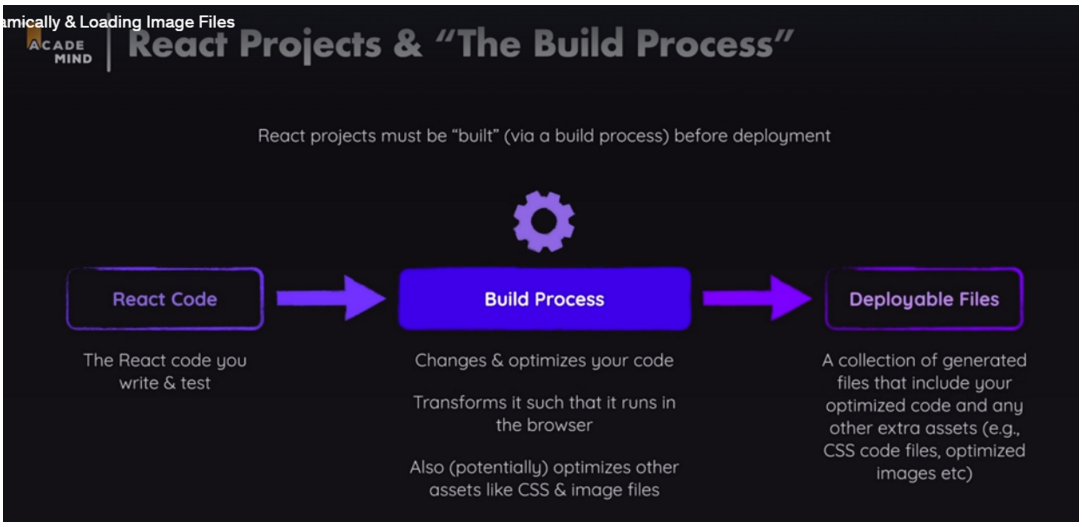
nts & How It Builds A "Component Tree" [Core Concept]



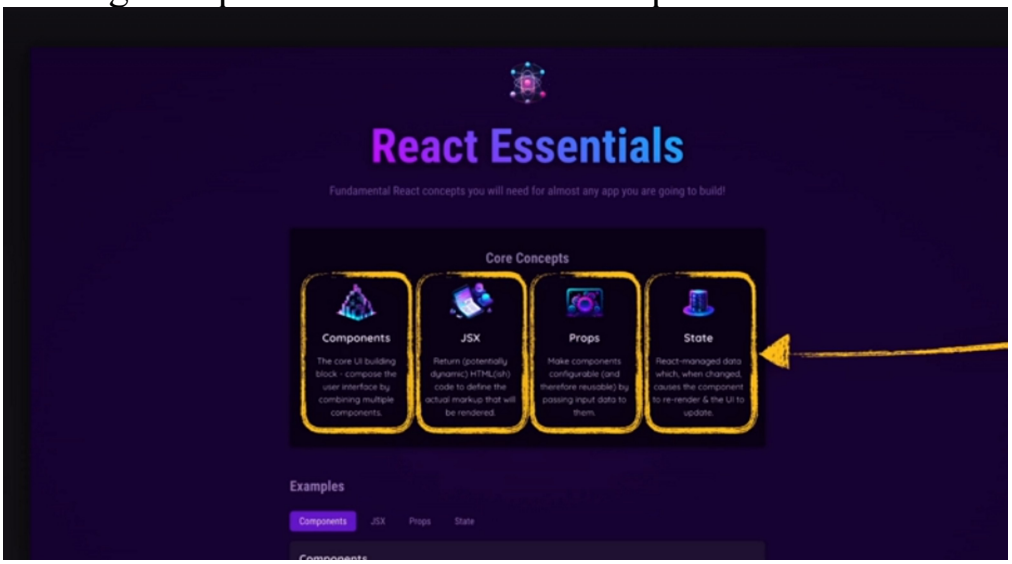
alues [Core Concept]

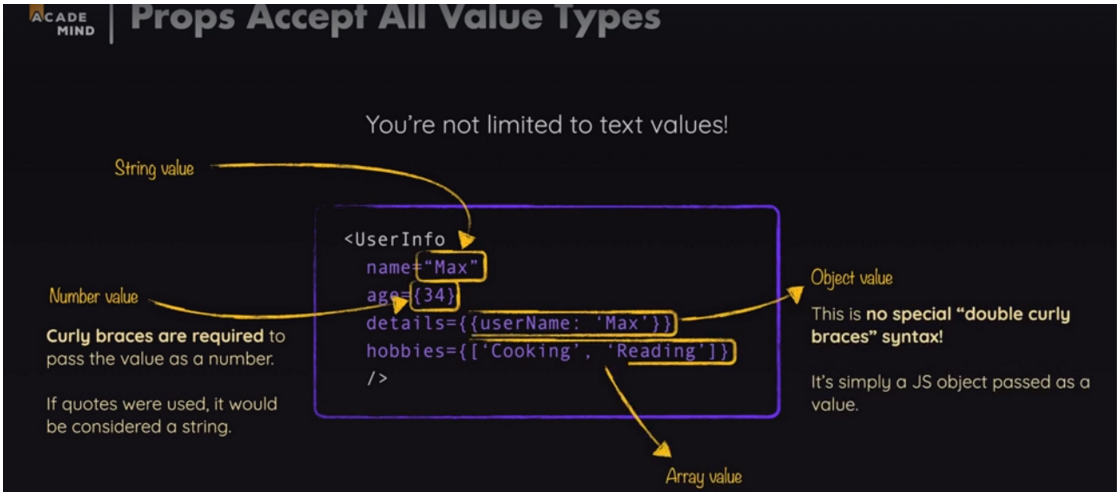


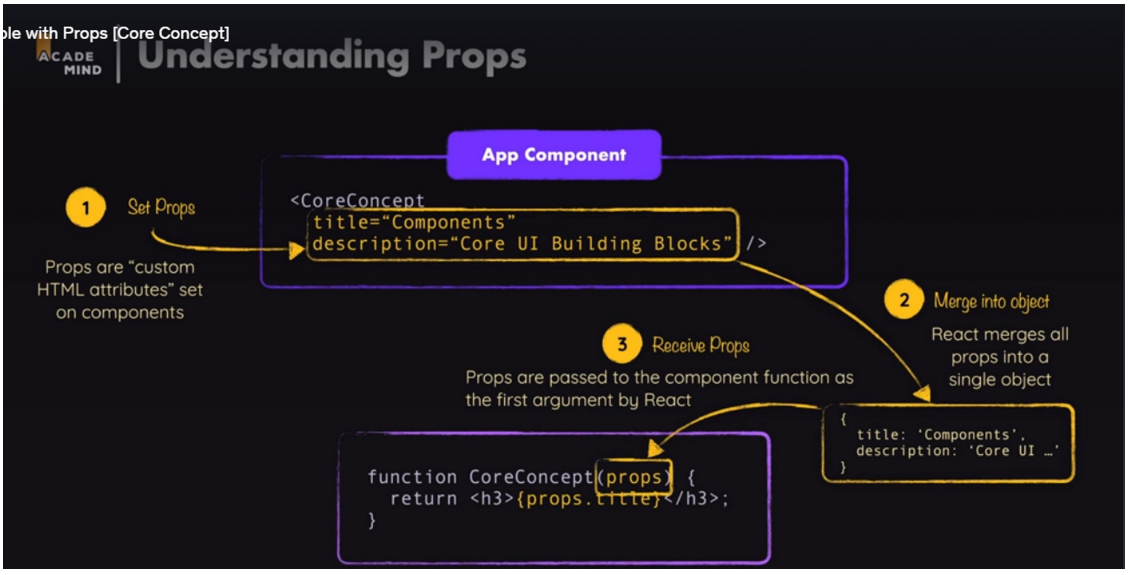


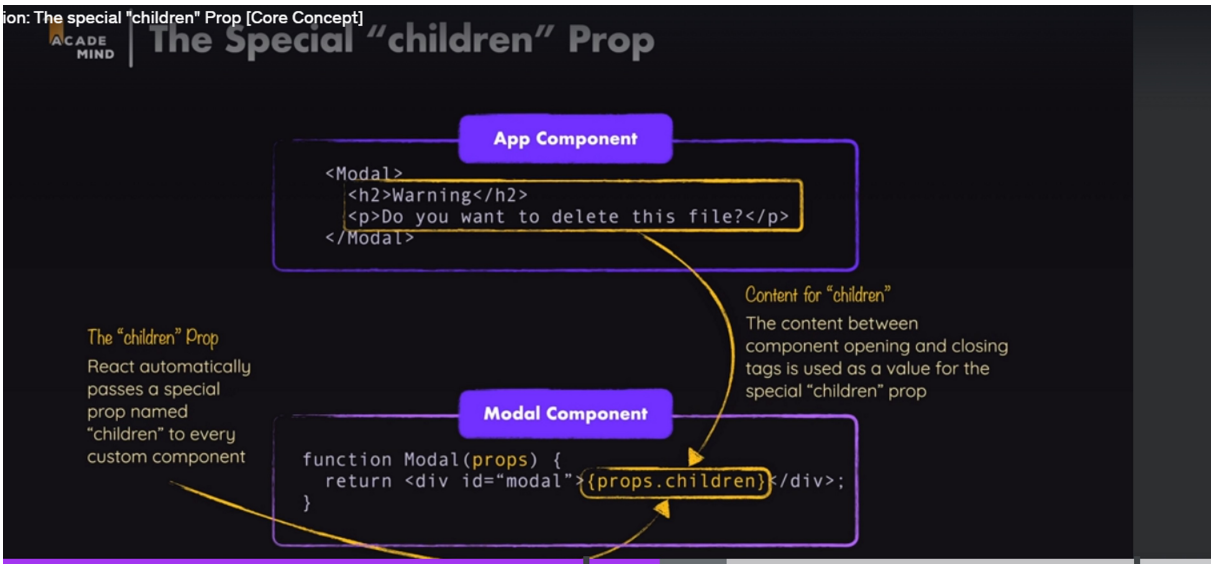


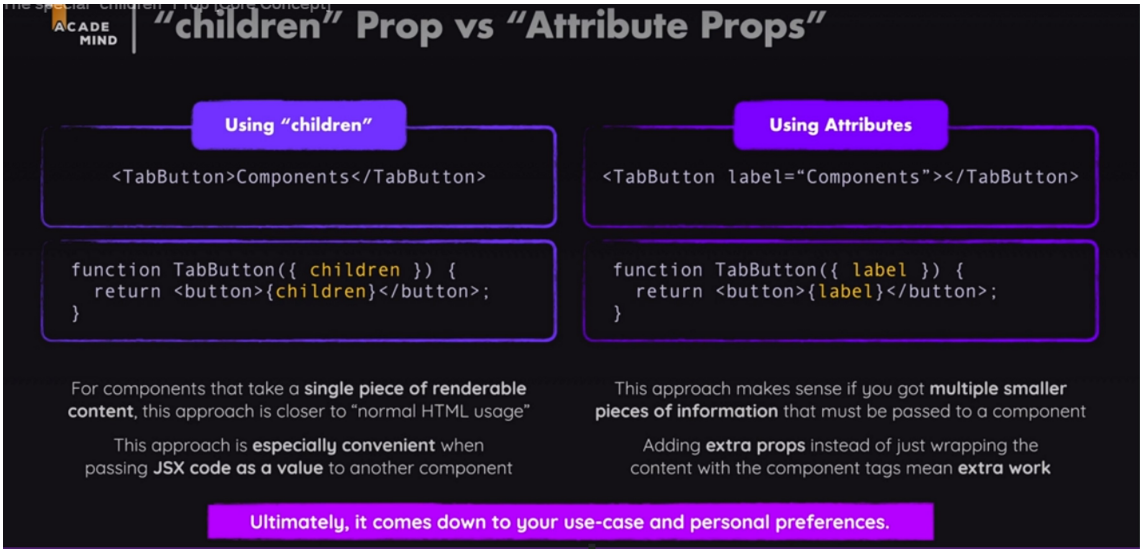
Making Components Reusable with Props




ACADEMIND
Props Accept All Value Types
You're not limited to text values!
String value
Number value
Curly braces are required to pass the value as a number.
If quotes were used, it would be considered a string.
<UserInfo
name="Max"
age={34}
details={{userName: 'Max'}}
hobbies={['Cooking', 'Reading']}
/>
Array value
Object value
This is no special "double curly braces" syntax!
It's simply a JS object passed as a value.


ACADEMIND
Understanding Props
App Component
1 Set Props
Props are "custom HTML attributes" set on components
<CoreConcept
title="Components"
description="Core UI Building Blocks" />
2 Merge into object
React merges all props into a single object
{
title: 'Components',
description: 'Core UI …'
}
3 Receive Props
Props are passed to the component function as the first argument by React
function CoreConcept(props) {
return <h3>{props.title}</h3>;
}

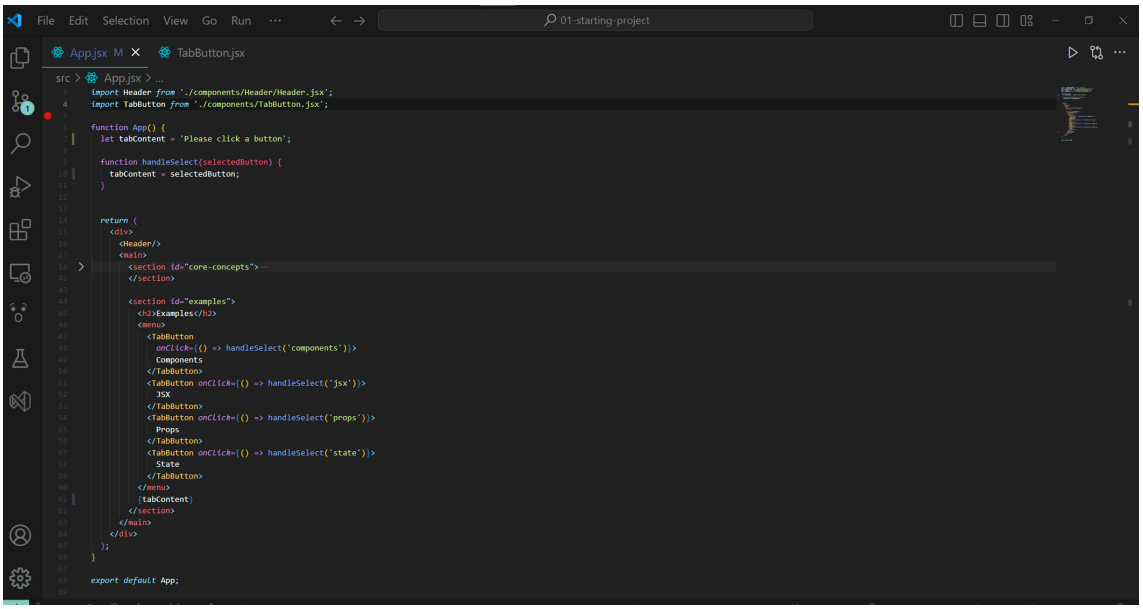

ACADEMIND
The Special "children" Prop
App Component
<Modal>
<h2>Warning</h2>
Do you want to delete this file?
</Modal>
Content for "children"
The content between component opening and closing tags is used as a value for the special "children" prop
The "children" Prop
React automatically passes a special prop named "children" to every custom component
Modal Component
function Modal(props) {
return {props.children} ;
}


ACADEMIND
"children" Prop vs "Attribute Props"
Using "children"
<TabButton>Components</TabButton>
function TabButton({ children }) {
return <button>{children}</button>;
}
Using Attributes
<TabButton label="Components"></TabButton>
function TabButton({ label }) {
return <button>{label}</button>;
}
For components that take a single piece of renderable content, this approach is closer to "normal HTML usage"
This approach is especially convenient when passing JSX code as a value to another component
This approach makes sense if you got multiple smaller pieces of information that must be passed to a component
Adding extra props instead of just wrapping the content with the component tags mean extra work
Ultimately, it comes down to your use-case and personal preferences.


export default function TabButton({ children }) {
function
return (

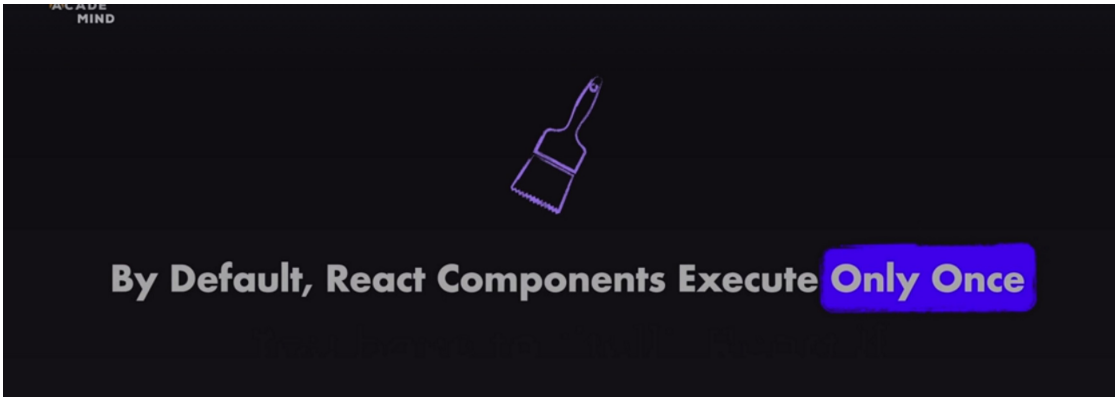
<button onClick={}>{children}</button>

);
}
The advantage of defining these event handler functions inside the component function is that they then have access to the component's props and state (covered later).

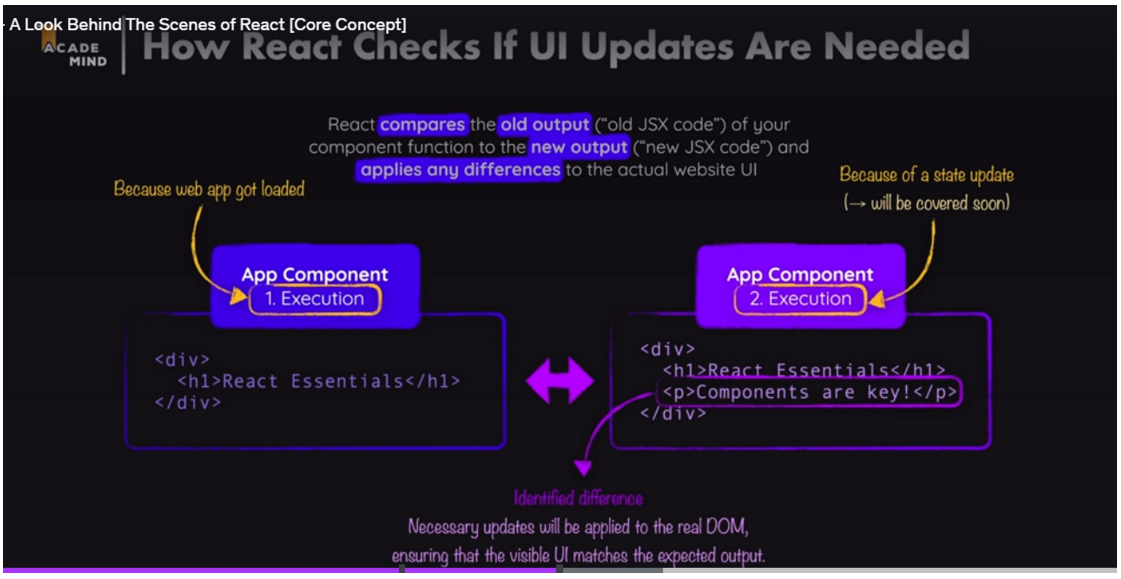

File Edit Selection View Go Run
01-starting-project
App.jsx M
TabButton.jsx
src > App.jsx > ...
import Header from './components/Header/Header.jsx';
import TabButton from './components/TabButton.jsx';
function App() {
let tabContent = 'Please click a button';
function handleSelect(selectedButton) {
tabContent = selectedButton;
}
return (

<Header/>
<main>
<section id="core-concepts">
</section>
<section id="examples">
<h2>Examples</h2>
<menu>
<TabButton
onClick={() => handleSelect('components')}>
Components
</TabButton>
<TabButton onClick={() => handleSelect('jsx')}>
JSX
</TabButton>
<TabButton onClick={() => handleSelect('props')}>
Props
</TabButton>
<TabButton onClick={() => handleSelect('state')}>
State
</TabButton>
</menu>
{tabContent}
</section>
</main>

);
}
export default App;


By Default, React Components Execute Only Once


By Default, React Components Execute Only Once
You have to "tell" React If A Component Should be Executed Again

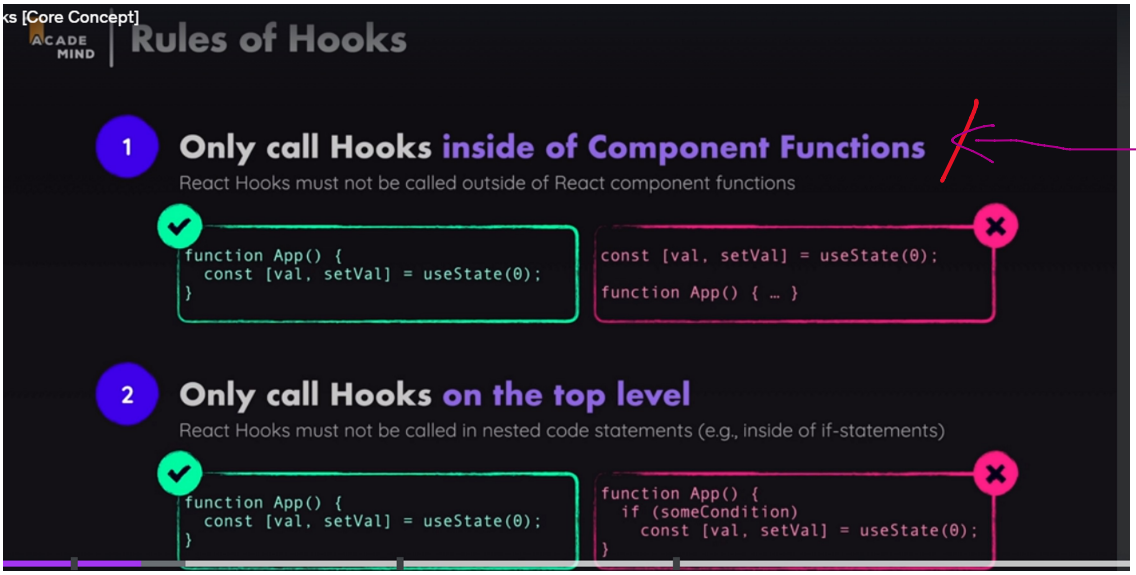

A Look Behind The Scenes of React [Core Concept]
How React Checks If UI Updates Are Needed
React compares the old output ("old JSX code") of your component function to the new output ("new JSX code") and applies any differences to the actual website UI
Because web app got loaded
Because of a state update (→ will be covered soon)
App Component
1. Execution
App Component
2. Execution

<h1>React Essentials</h1>

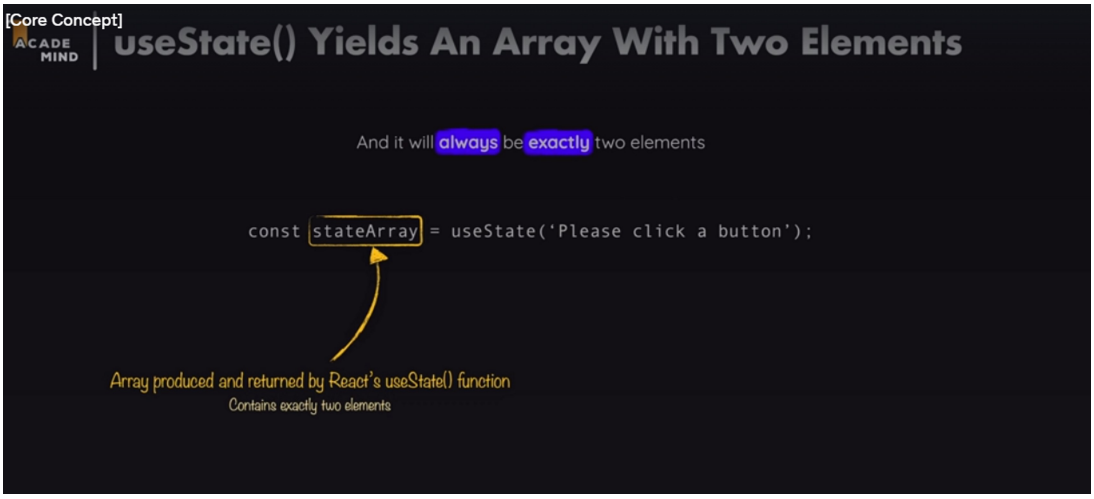
<h1>React Essentials</h1>
Components are key!

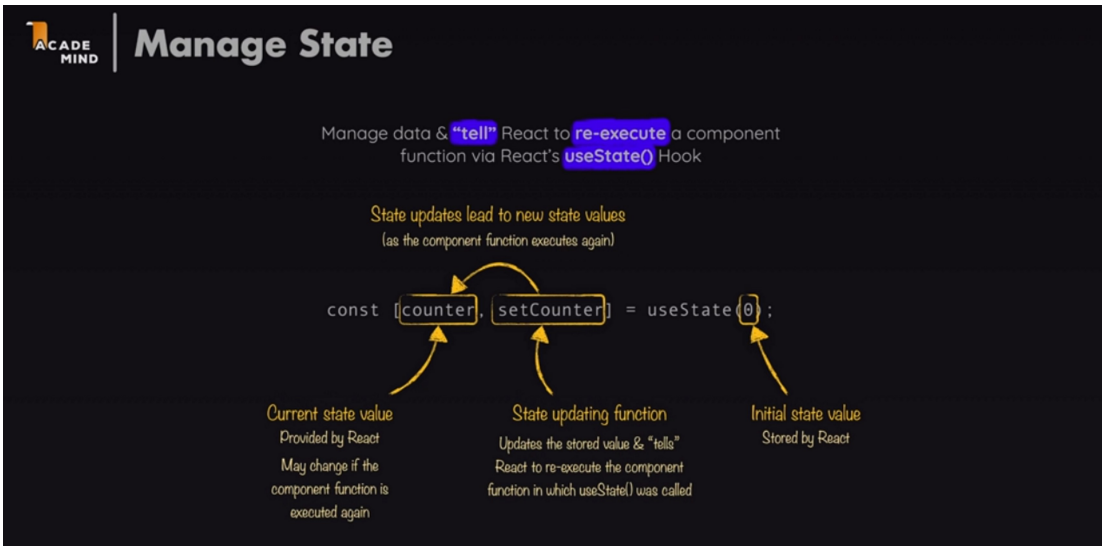
Identified difference
Necessary updates will be applied to the real DOM, ensuring that the visible UI matches the expected output.


ks [Core Concept]
Rules of Hooks


ks [Core Concept]
ACADEMIND
Rules of Hooks
1
Only call Hooks inside of Component Functions
React Hooks must not be called outside of React component functions
function App() {
const [val, setVal] = useState(0);
}
const [val, setVal] = useState(0);
function App() { … }
2
Only call Hooks on the top level
React Hooks must not be called in nested code statements (e.g., inside of if-statements)
function App() {
const [val, setVal] = useState(0);
}
function App() {
if (someCondition)
const [val, setVal] = useState(0);
}

Custom Hook

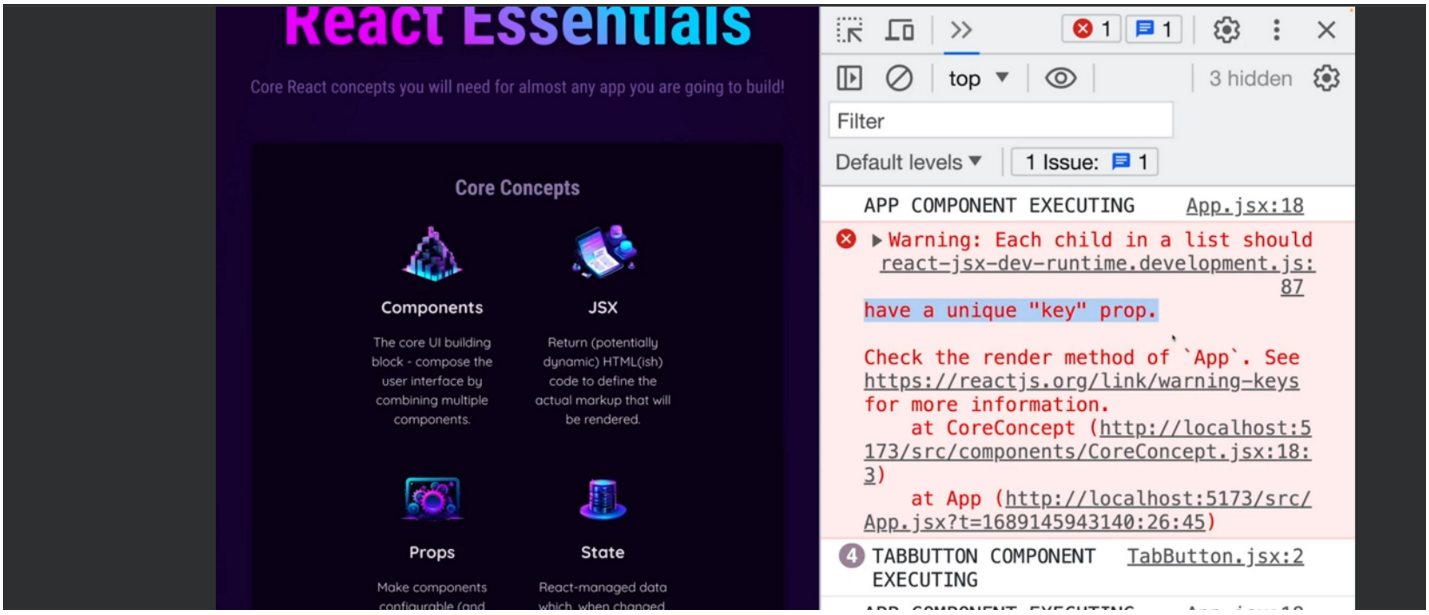

[Core Concept]
ACADEMIND
useState() Yields An Array With Two Elements
And it will always be exactly two elements
const stateArray = useState('Please click a button');
Array produced and returned by React's useState() function
Contains exactly two elements


ACADEMIND
Manage State
Manage data & "tell" React to re-execute a component function via React's useState() Hook
State updates lead to new state values
(as the component function executes again)
const [counter, setCounter] = useState(0);
Current state value
Provided by React
May change if the component function is executed again
State updating function
Updates the stored value & "tells" React to re-execute the component function in which useState() was called
Initial state value
Stored by React

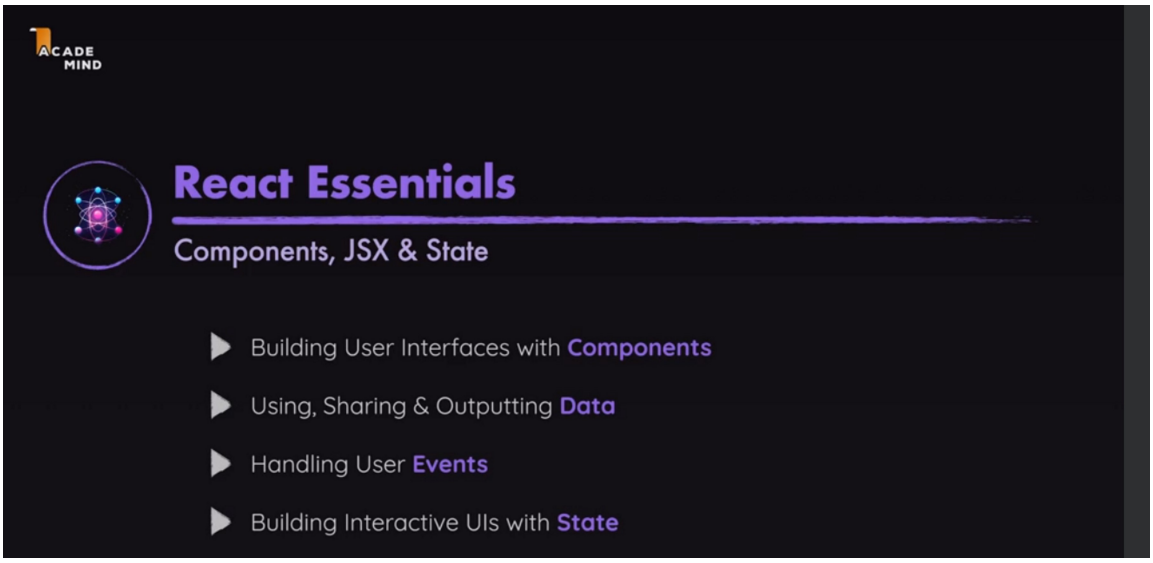
```
function App() {
  const [ selectedTopic, setSelectedTopic ] = useState('Please click a but

  function handleSelect(selectedButton) {
    // selectedButton => 'components', 'jsx', 'props', 'state'
    setSelectedTopic(selectedButton);
    console.log(selectedTopic);
  }
```

When you call state updating function here,
React in the end schedules, this state update and it then re-executes this component function.
So therefore the updated value will only be available after this app component function executed again.

Outputting List Data Dynamically



Module Summary


ACADEMIND
React Essentials
Components, JSX & State
Building User Interfaces with Components
Using, Sharing & Outputting Data
Handling User Events
Building Interactive UIs with State