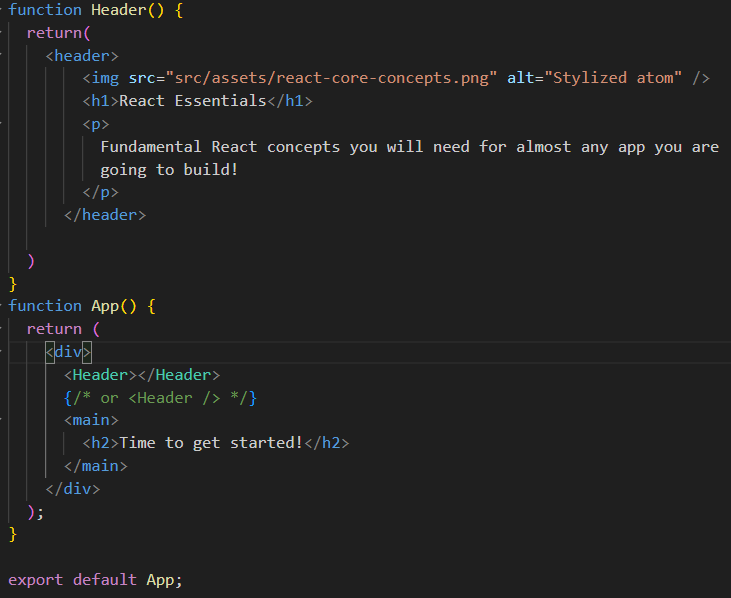
In React we will create components like a function declared in a java script

Later we will execute that function as

Declare it as a tag in default in component.



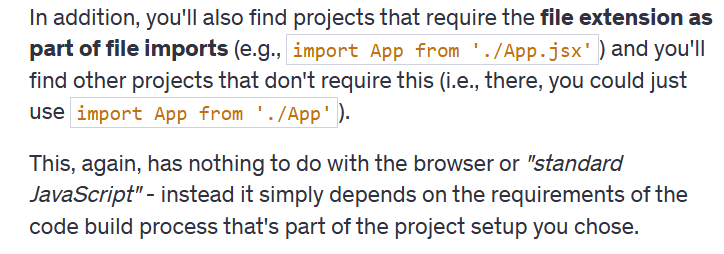
Here we created a new component as Header()

It declared in default component as a tag

<Header></Header>

      {/\* or <Header /> \*/}

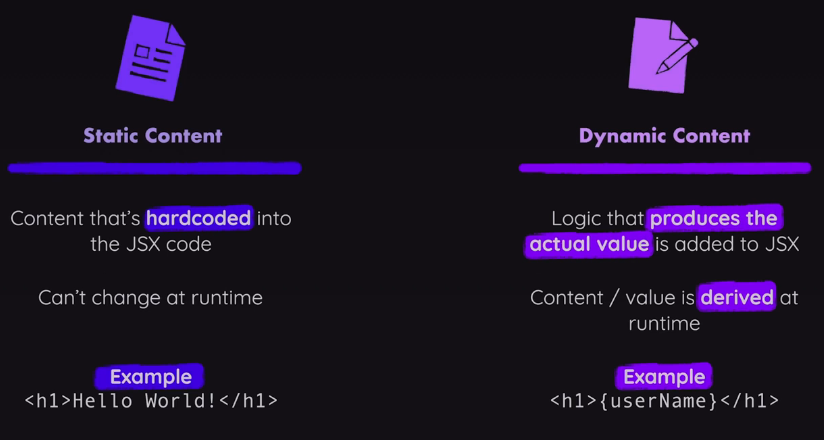
We can simply write like a forward slash as **<Header />**



There are 2 type of components:

* Built-In Components: These are nothing but html elements, it’ll start with lower case
* Custom Components: you need start with Uppercase

Using & Outputting dynamic values:



const reactDescriptions = ['Fundamental','Crucial','Core'];

function genRandomInt(max){

  return Math.floor(Math.random() \* (max+1));

}

function Header() {

  return(

    <header>

        <img src="src/assets/react-core-concepts.png" alt="Stylized atom" />

        <h1>React Essentials</h1>

        <p>

          {reactDescriptions[genRandomInt(2)]} React concepts you will need for almost any app you are

          going to build!

        </p>

      </header>

  )

}

Or we can write like

Const description = reactDescriptions(getRandomInt(2));

{description}

We can use similar concept to load images in different way as below

Import reactimg from ‘./assets/react-core-concepts.png’;

<img src ={reactimg} alt = “Stylized atom” />

React allows you to pass data to components via a concept called **Props**.

It is like a read only properties that were shared between components. A Parent component can send data to a child component. Like key:value, for any integers we need to mention like {18}other than string

import componentsimg from './assets/components.png'

import jsximg from './assets/jsx-ui.png'

import propimg from './assets/config.png'

import stateimg from './assets/state-mgmt.png'

function Coreconcepts(props){

  return(

    <li>

      <img src={props.image}  />

      <h3>{props.title}</h3>

      <p>{props.description}</p>

    </li>

  )

}

function App() {

  return (

    <div>

      <Header></Header>

      {/\* or <Header /> \*/}

      <main>

        <section id = "core-concepts">

          <h2>Core Concepts</h2>

          <ul>

            <Coreconcepts

            title="Components"

            description="The core UI building block."

            image={componentsimg}

            />

            <Coreconcepts

            title="JSX"

            description="Return (potentially dynamic) HTML(ish) code to define the actual markup that will be rendered."

            image={jsximg}

            />

            <Coreconcepts

            title="Props"

            description="Make components configurable (and therefore reusable) by passing input data to them."

            image={propimg}

            />

            <Coreconcepts

            title="State"

            description="React-managed data which, when changed, causes the component to re-render & the UI to update."

            image={stateimg}

            />

          </ul>

        </section>

        <h2>Time to get started!</h2>

      </main>

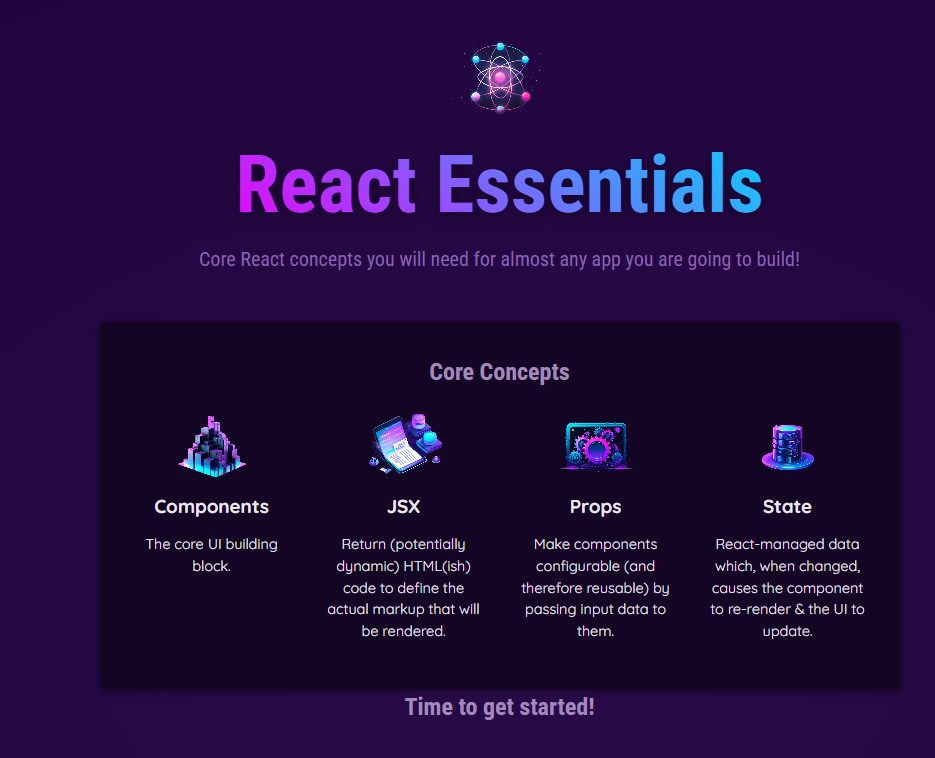
    </div>

  );

}

export default App;

o/p:

 by using this props concept we displayed 4 components.

We can write in other way we will create a new file as data.js

import componentsImg from './assets/components.png';

import propsImg from './assets/config.png';

import jsxImg from './assets/jsx-ui.png';

import stateImg from './assets/state-mgmt.png';

export const CORE\_CONCEPTS = [

  {

    image: componentsImg,

    title: 'Components',

    description:

      'The core UI building block - compose the user interface by combining multiple components.',

  },

  {

    image: jsxImg,

    title: 'JSX',

    description:

      'Return (potentially dynamic) HTML(ish) code to define the actual markup that will be rendered.',

  },

  {

    image: propsImg,

    title: 'Props',

    description:

      'Make components configurable (and therefore reusable) by passing input data to them.',

  },

  {

    image: stateImg,

    title: 'State',

    description:

      'React-managed data which, when changed, causes the component to re-render & the UI to update.',

  },

];

import {CORE\_CONCEPTS} from './data.js'

function Coreconcepts(props){

  return(

    <li>

      <img src={props.image}  />

      <h3>{props.title}</h3>

      <p>{props.description}</p>

    </li>

  )

}

<section id= "core-concepts">

            <h2>Core Concepts</h2>

            <ul>

              <Coreconcepts

              title ={CORE\_CONCEPTS[0].title}

              description={CORE\_CONCEPTS[0].description}

              image={CORE\_CONCEPTS[0].image}

              />

              <Coreconcepts

              title ={CORE\_CONCEPTS[1].title}

              description={CORE\_CONCEPTS[1].description}

              image={CORE\_CONCEPTS[1].image}

              />

              <Coreconcepts

              title ={CORE\_CONCEPTS[2].title}

              description={CORE\_CONCEPTS[2].description}

              image={CORE\_CONCEPTS[2].image}

              />

              <Coreconcepts

              title ={CORE\_CONCEPTS[3].title}

              description={CORE\_CONCEPTS[3].description}

              image={CORE\_CONCEPTS[3].image}

              />

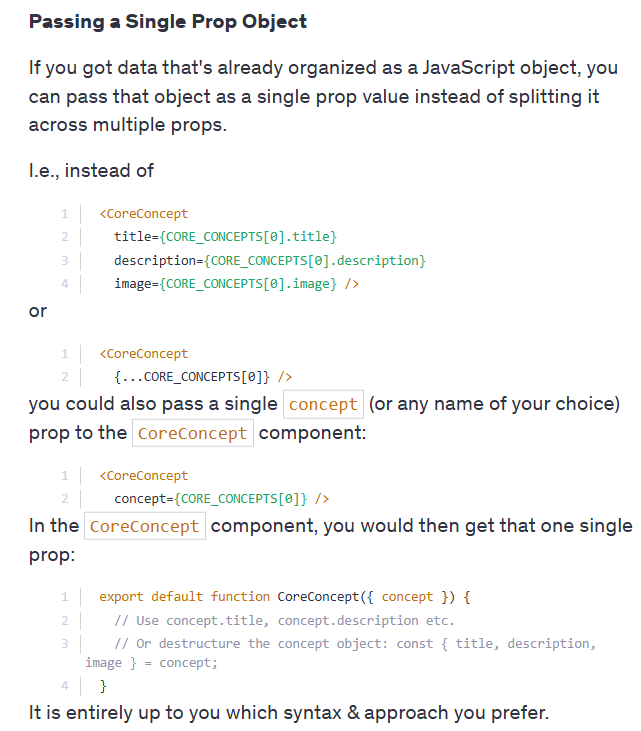
            </ul>

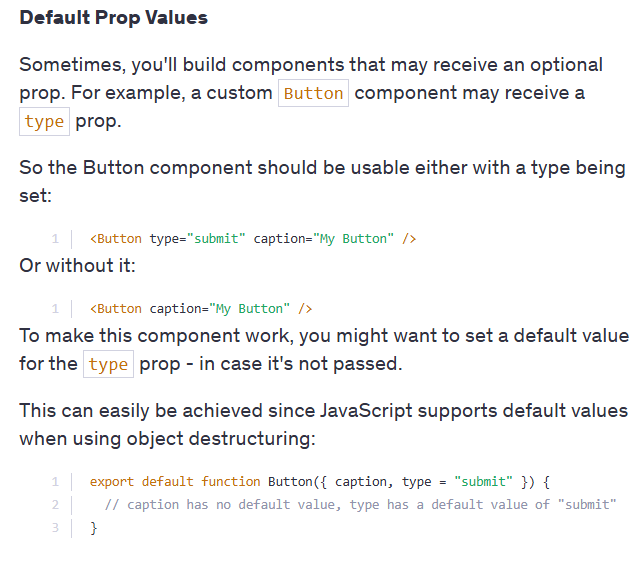
        </section>

Alternative way of approach.

Instead of writing whole component we can use directly spread operator.

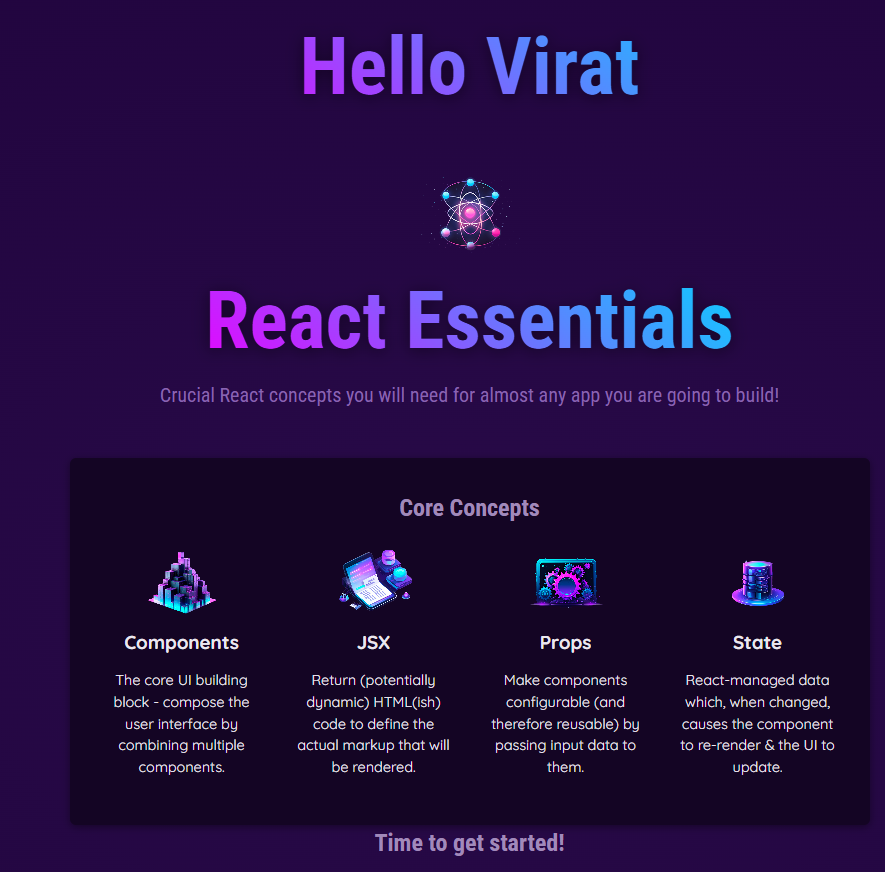
<Coreconcepts {...CORE\_CONCEPTS[3]}/>



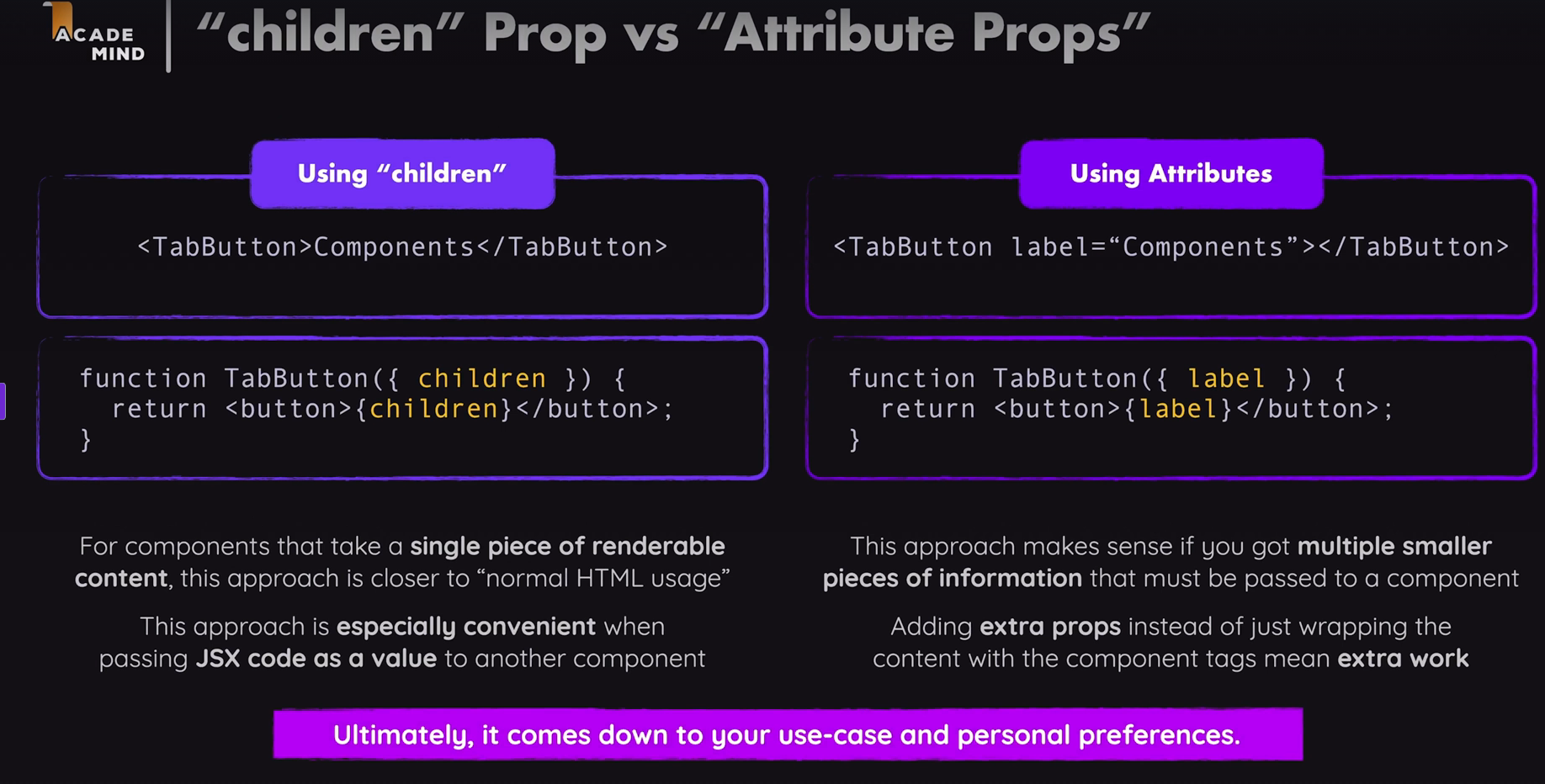


There are some limitations if you store all the header elements in a Header.jsx file and the header style properties in a separate Header.css file.

If you write any header element in the App.jsx file, its properties will also be applied to the header element in the App.jsx file

 like wise React Essential properties also applied to Hello Virat

Lets Create a new examples Using children props concept



For this question   
**Component Composition**

Your task is to create a reusable Card component that takes a name prop as an input and, in addition, can be wrapped around any JSX code.

Use the already existing Card.js file to create the Card component in there. You can add the card CSS class to the main wrapping element in that component for some styling.

The name prop should be output as a title inside the Card component, the wrapped JSX code should be output below that title.

For example, the final Card component, should be usable like this:

1. <Card name="Maria Miles">
2. <p>
3. Maria is a professor of Computer Science at the University of Illinois.
4. </p>
5. <p>
6. <a href="mailto:blake@example.com">Email Maria</a>
7. </p>
8. </Card>

This should yield the following visual **output**:

Code:

import './Card.css';

function Card({ name, children }) {

return (

<div className="card">

<h2>{name}</h2>

<p>{children}</p>

</div>

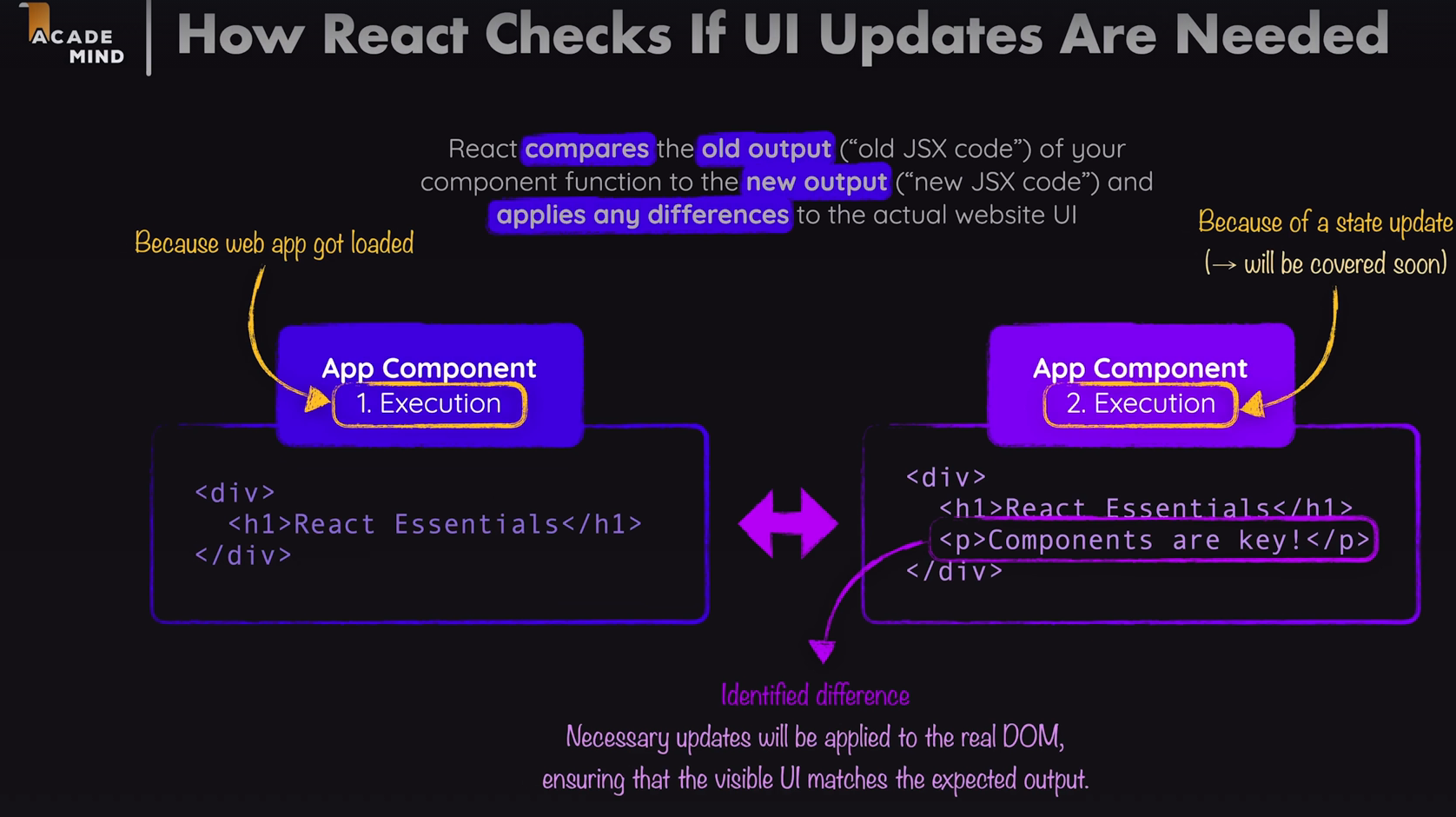
);

}

export default Card;

In next concept we can use functions as values, just like we use them in props.

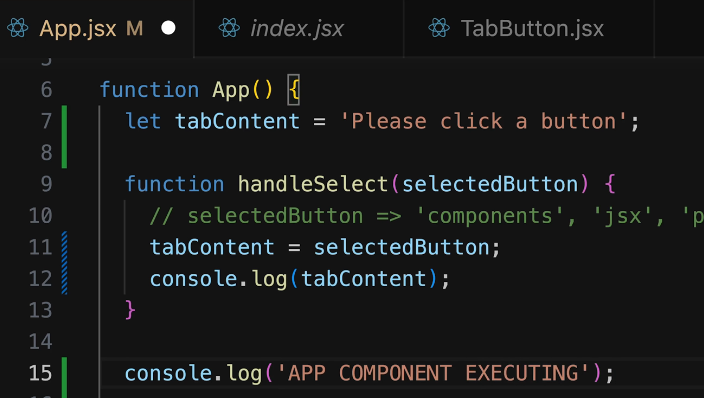
We need to pass only as a value not like a function



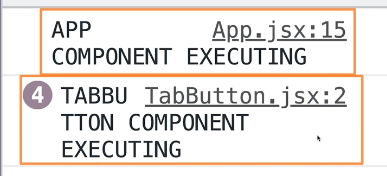
We need to remember one thing that react will only execute its component function only once.

In React, remember that each component function runs only once during its initial render. After that, React updates the component based on state or props changes without re-executing the function, ensuring efficient rendering and performance.

Eg:

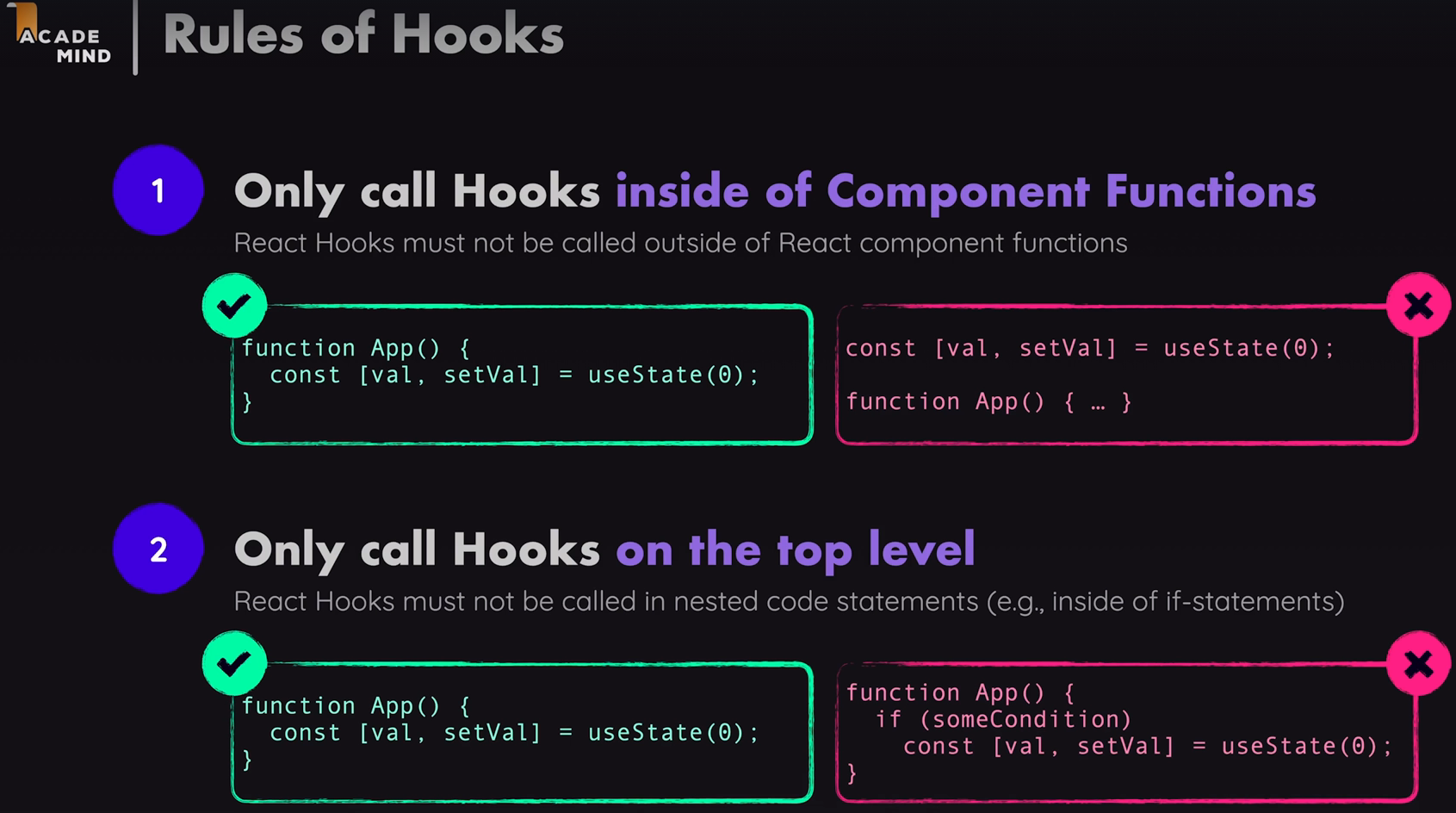
 

After running our project we will notice that app component will execute only 1 time where as tabButton component execute 4 times



**State Concept**: We will use **useState** it is usually called as **React Hooks** (all hooks starts with use**)**.

import {useState} from 'react';

 We are using this state concept to update the UI.

State is React’s way of managing dynamic data inside components. Unlike props (which are fixed once passed), state changes over time and makes components interactive.

**How our Code Uses State (useState)**

1. **Define state using useState hook:**

  const  [selectedTopic,setSelectedtopic] = useState();

* + selectedTopic is the **current state value**.
  + setSelectedTopic is the **function to update the state**.
  + The default value is undefined (no topic is selected initially).

1. **Updating state when a button is clicked:**

function handSelect(selectedButton){

    setSelectedtopic(selectedButton);

    console.log(selectedTopic);

  }

* + Clicking a **TabButton** updates the selectedTopic.
  + The console.log(selectedTopic) might not show the updated value immediately because **state updates are asynchronous** in React.

1. **Conditional Rendering Based on State:**

{!selectedTopic && <p>Please Select  a Topic.</p>}

          {selectedTopic && (<div id ="tab-content">

            <h3>{EXAMPLES[selectedTopic].title}</h3>

            <p>{EXAMPLES[selectedTopic].description}</p>

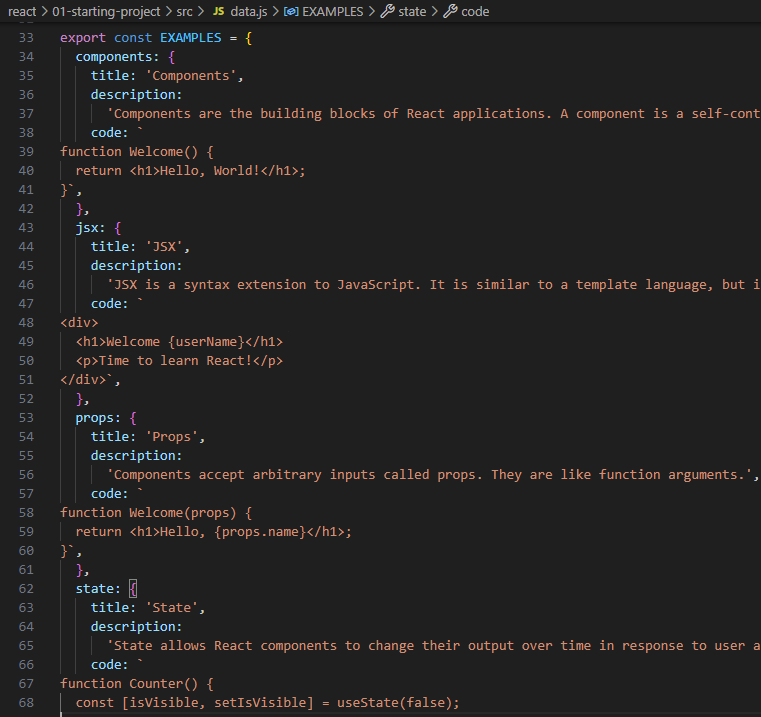
            <pre>

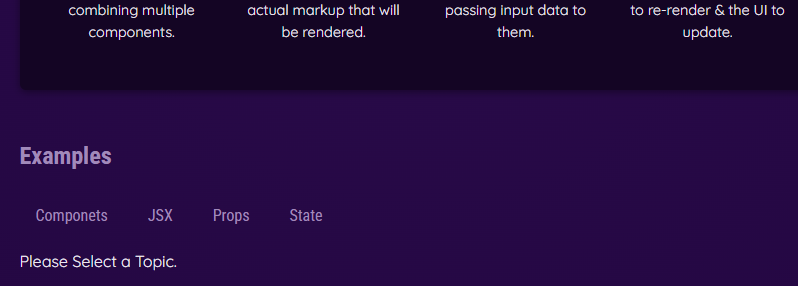
              <code>

                {EXAMPLES[selectedTopic].code}

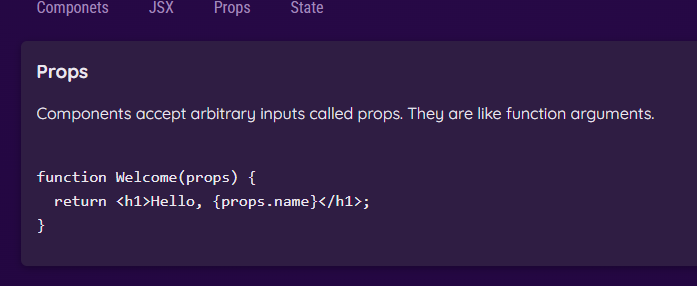
        </code>

            </pre>   </div>)}

* + If no topic is selected → Show "Please Select a Topic."
  + If a topic **is** selected → Show its title, description, and example code.
  + it means if we not selected any value it will show like below
  + Moreover EXAMPLES is declared as object like key value pairs
  + 
  + Output will occur as below



After selecting a topic



We can write in other way also Like Rendering Content Conditionally.

let tabContent = <p> Please Select A topic.</p>

  if(selectedTopic){

    tabContent = (<div id ="tab-content">

      <h3>{EXAMPLES[selectedTopic].title}</h3>

      <p>{EXAMPLES[selectedTopic].description}</p>

      <pre>

        <code>

          {EXAMPLES[selectedTopic].code}

        </code>

      </pre>

    </div>);

  }

We can execute it by calling this variable by         **{tabContent}** we will get same output as above simply by using if statement and a variable.

We can also set a conditional CSS class like we can highlight selected topic of example.

export default function TabButton({children, onSelect, isSelected}){

    return(

        <li>

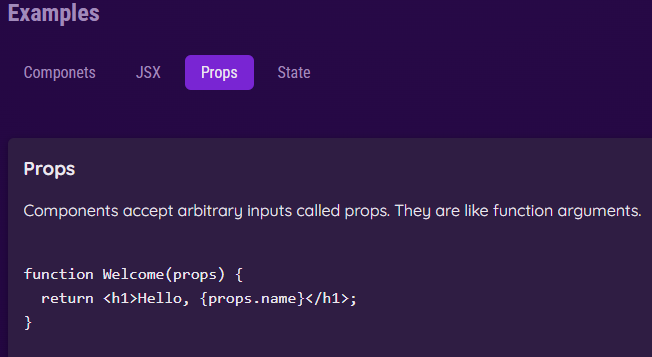
            <button className = {isSelected ? 'active': undefined} onClick={onSelect}>{children}</button>

        </li>

    )

}

Here we added new prop as isSelected



As we see props topic is highlighted this is how we can use dynamic styling through css.

In jsx class is written as **className**

**Outputting List Data Dynamically:**

We know in CORE\_CONCEPTS array has 4 elements, if any 1 element has no data then our output will not occur. If number of coreconcepts elements will derive by dynamically we will get this issue.

{CORE\_CONCEPTS.map((conceptItem)=>

                <Coreconcepts {...conceptItem}/>

              )}

You need to use a <div> (or any other enclosing tag) to wrap EXAMPLES data because React requires a single root element in a component’s return statement.

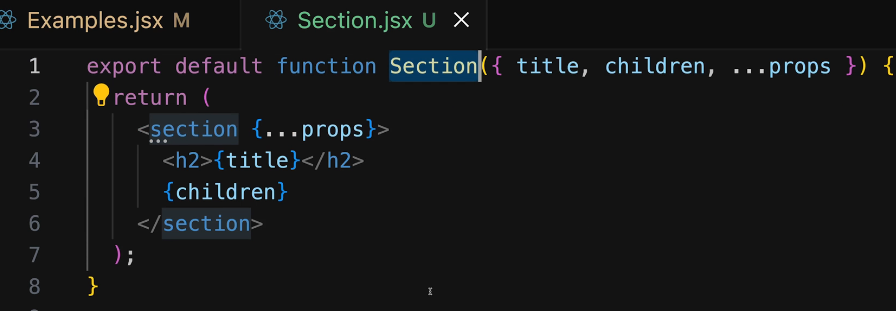
JSX doesn’t allow returning multiple sibling elements directly without a parent container.

 like this

Like this we can use a limitation for this Fragment(<> to </>)

Forwarding props to wrapped elements: we can use …props

The {...props} syntax is the **spread operator** used to pass down all additional properties (props) that the <Section> component receives to the underlying <Section> element.

****

Working with Multiple JSX slots:

When building reusable React components, we often need to pass different types of content into different sections of the component. In HTML, elements like <section> and <menu> naturally contain different types of content, but in React, we need a structured way to handle this.

To achieve this, we use **props forwarding** and **multiple JSX slots** in a reusable component like Tabs.jsx.

**Why Use Multiple JSX Slots?**

**Problem with a Single Children Prop**

React components typically use children to receive content, but when we need **multiple separate content areas**, such as:

* A tab menu (buttons)
* A tab content area

We need more than just children. If we only rely on children, we lose flexibility.

**Solution: Pass JSX Code as Props**

Instead of just using children, we define **additional named props** to pass JSX content to different areas.

**Implementing the Tabs Component**

**Creating Tabs.jsx**

We create a reusable Tabs component that enforces a structure:

* **A <menu> for buttons**
* **A section below for dynamic content**

**Using the Tabs Component in Examples.jsx**

**Before (Without Slots)**

**Originally, the tab buttons and content were in the same component, making it less reusable.**

**After (With Slots)**

**Now, we pass tab buttons separately while keeping content inside Tabs.**

export default function Tabs({children,buttons}){

    return(

    <>

      <menu>{buttons}</menu>

      {children}

    </>

    );

}



**Setting Components with Dynamic Types in React**

**1️ The Concept: Dynamic Component Wrappers**

* In React, you may want to dynamically set the wrapper element around certain UI elements.
* Instead of hardcoding an element (e.g., <menu> or <div>), we can allow the developer to choose which HTML element or custom component should be used.
* This is useful in larger applications where a component may be reused in different contexts.

**2️ How It Works**

* We pass a component identifier (either a built-in element like "menu" or "div" or a custom component) as a prop.
* React will determine whether to render a built-in HTML element or a custom React component based on the value of this prop.

Example:

export default function Tabs({children,buttons,buttonsContainer}){

    const ButtonsContainer = buttonsContainer;

    return(

    <>

      <ButtonsContainer>{buttons}</ButtonsContainer>

      {children}

    </>

    );

}

**<Tabs buttonsContainer="menu" buttons={...} />**

Like wise

For Custom Components: Custom components must be passed as a dynamic value (without quotes) because they are React functions.

**<Tabs buttonsContainer={Section} buttons={...} />**

**Imp points:**

* Built-in elements (menu, div, ul) must be passed as a string.
* Custom components must be passed as a variable (without quotes).
* The prop should be mapped to an uppercase-named variable (ButtonsContainer) so that React treats it as a component.

**Setting Default Prop Values:**

We can use built in elements manually without enter it dynamically.

**What Are Default Prop Values?**

* Default prop values allow a React component to have a fallback value **if a prop is not provided**.
* This makes the component more **flexible** and **user-friendly** by ensuring it still functions without explicitly passing all props.

**2️ How to Set Default Prop Values?**

* When using destructuring in function parameters, we can set a default value using =.

🔹 Example: Setting Default Prop Value for buttonsContainer

export default function Tabs({children,buttons,buttonsContainer='menu'}){

Defaulting it to a Custom Component

export default function Tabs({children,buttons,buttonsContainer=Section}){

In Tabsjsx we can’t need to pass buttonsContainer

Since buttonsContainer is not explicitly passed, it will default to 'menu' (or the Section component, depending on how it was set in Tabs)