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.

```
function Header() {
  return(
    <header>
      <img src="src/assets/react-core-concepts.png" alt="Stylized atom" />
      <h1>React Essentials</h1>
      <p>
        Fundamental React concepts you will need for almost any app you are
        going to build!
      </p>
    </header>

  )
}
function App() {
  return (
    <div>
      <Header></Header>
      {/* or <Header /> */}
      <main>
        <h2>Time to get started!</h2>
      </main>
    </div>
  );
}

export default App;
```

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 />**

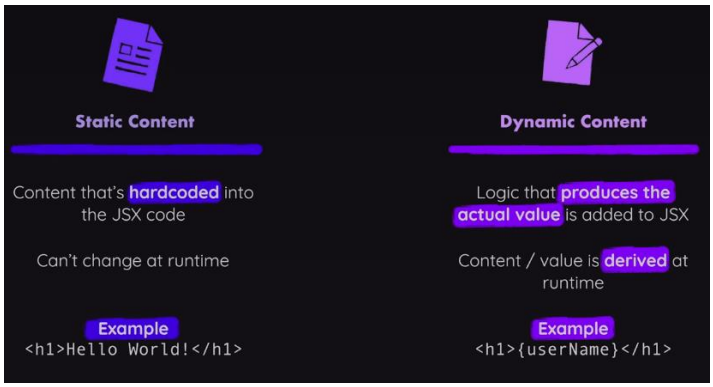
In addition, you'll also find projects that require the **file extension as part of file imports** (e.g., `import App from './App.jsx'`) and you'll find other projects that don't require this (i.e., there, you could just use `import App from './App'`).

This, again, has nothing to do with the browser or *"standard JavaScript"* - instead it simply depends on the requirements of the code build process that's part of the project setup you chose.

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';

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

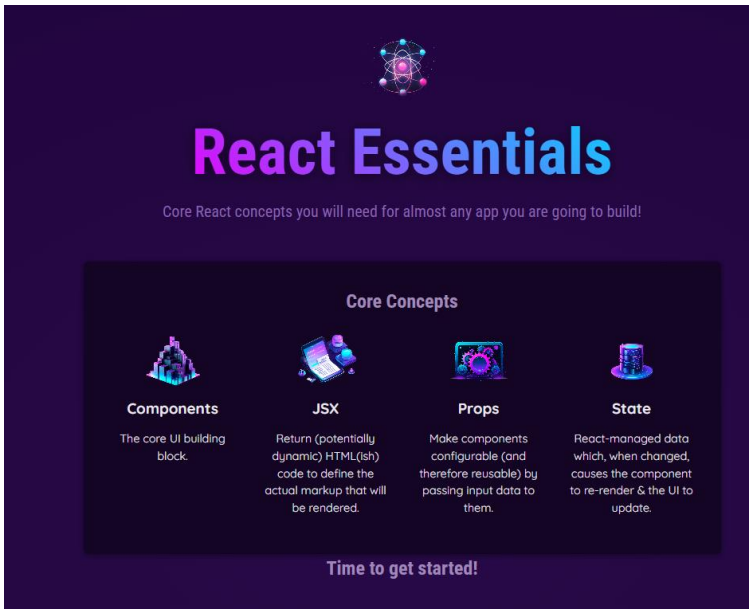
It is like a read only properties that were shared between components. A Parent component can send data to a child component. Like key:value, for any integers we need to mention like {18} other than string

```
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]}/>
```

Passing a Single Prop Object

If you got data that's already organized as a JavaScript object, you can pass that object as a single prop value instead of splitting it across multiple props.

I.e., instead of

```
<CoreConcept
  title={CORE_CONCEPTS[0].title}
  description={CORE_CONCEPTS[0].description}
  image={CORE_CONCEPTS[0].image} />
```

or

```
<CoreConcept
  {...CORE_CONCEPTS[0]} />
```

you could also pass a single `concept` (or any name of your choice) prop to the `CoreConcept` component:

```
<CoreConcept
  concept={CORE_CONCEPTS[0]} />
```

In the `CoreConcept` component, you would then get that one single prop:

```
export default function CoreConcept({ concept }) {
  // Use concept.title, concept.description etc.
  // Or destructure the concept object: const { title, description, image } = concept;
}
```

It is entirely up to you which syntax & approach you prefer.

Default Prop Values

Sometimes, you'll build components that may receive an optional prop. For example, a custom `Button` component may receive a `type` prop.

So the Button component should be usable either with a type being set:

```
<Button type="submit" caption="My Button" />
```

Or without it:

```
<Button caption="My Button" />
```

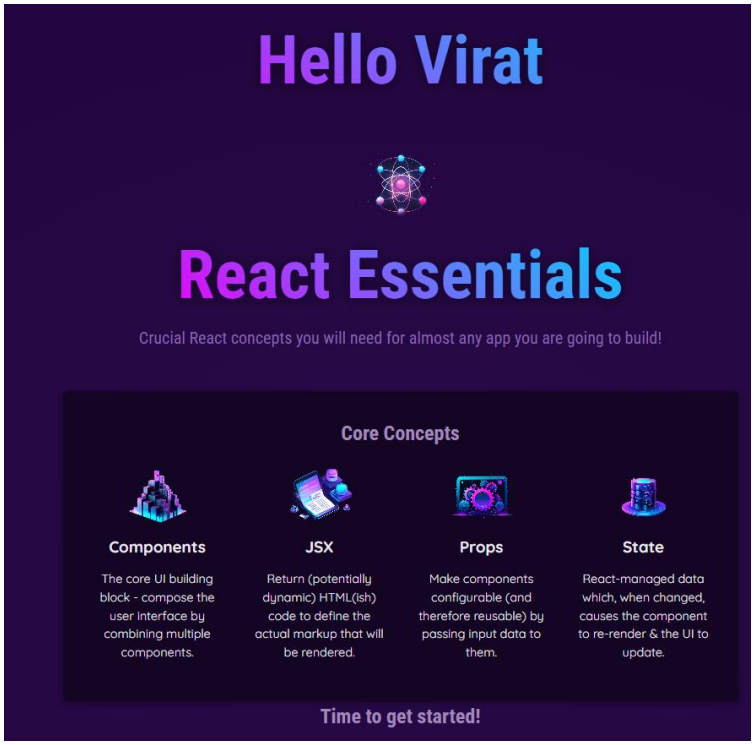
To make this component work, you might want to set a default value for the `type` prop - in case it's not passed.

This can easily be achieved since JavaScript supports default values when using object destructuring:

```
export default function Button({ caption, type = "submit" }) {
  // caption has no default value, type has a default value of "submit"
}
```

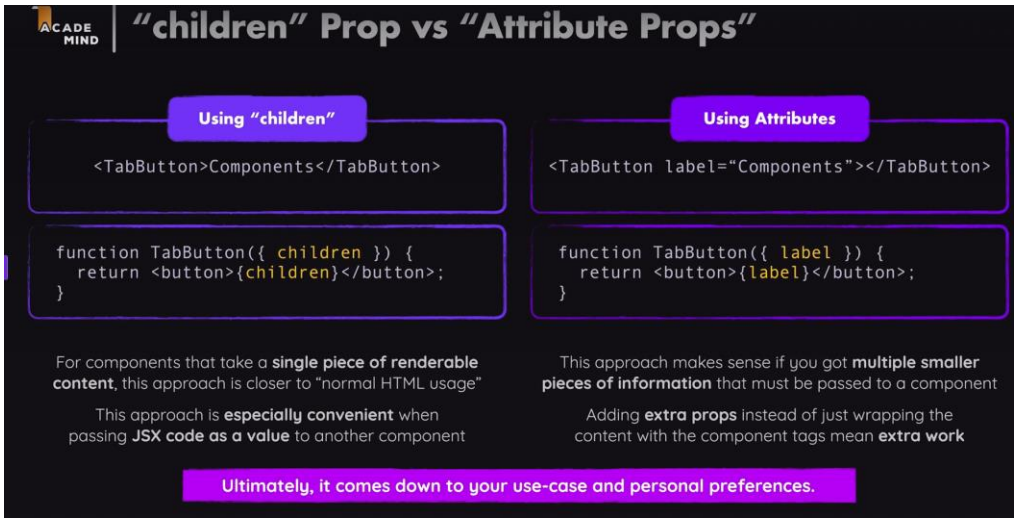
There are some limitations if you store all the header elements in a Header.jsx file and the header style properties in a separate Header.css file.

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:

```
<Card name="Maria Miles">
  <p>
    Maria is a professor of Computer Science at the University of Illinois.
  </p>
  <p>
    <a href="mailto:blake@example.com">Email Maria</a>
  </p>
</Card>
```

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

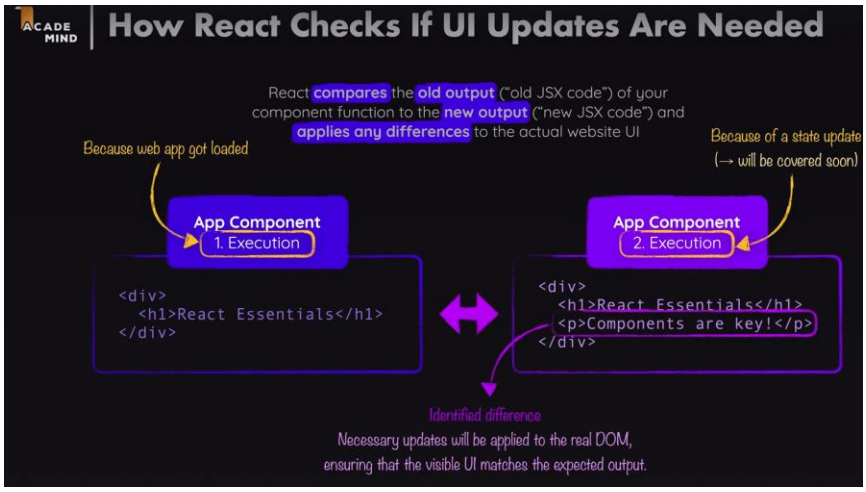
Code:

```
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:

App.jsx M | index.jsx | TabButton.jsx

```
function App() {
  let tabContent = 'Please click a button';

  function handleSelect(selectedButton) {
    // selectedButton => 'components', 'jsx', 'p
    tabContent = selectedButton;
    console.log(tabContent);
  }

  console.log('APP COMPONENT EXECUTING');
```

App.jsx M | index.jsx | TabButton.jsx

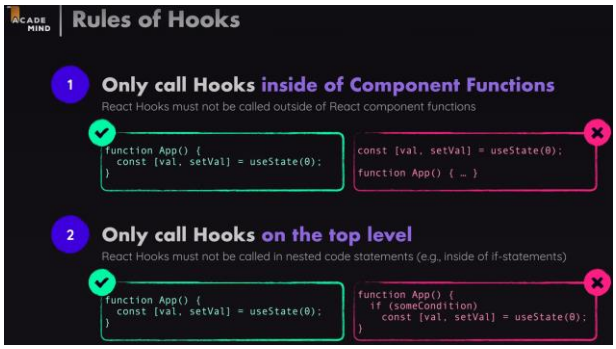
```
export default function TabButton({ children, onSelect }) {
  console.log('TABBUTTON COMPONENT EXECUTING');
  return (
    <li>
      <button onClick={onSelect}>{children}</button>
    </li>
  );
}
```

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

```
APP COMPONENT EXECUTING   App.jsx:15
4 TABBU TTON COMPONENT EXECUTING   TabButton.jsx:2
```

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).

2. **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.

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

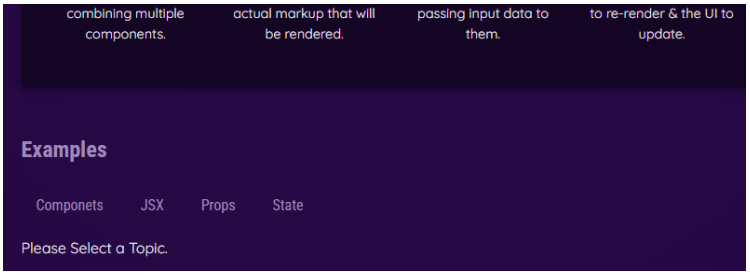
```
{!selectedTopic && <p>Please Select  a Topic.</p>}
          {selectedTopic && (<div id ="tab-content">
            <h3>{EXAMPLES[selectedTopic].title}</h3>
            <p>{EXAMPLES[selectedTopic].description}</p>
            <pre>
              <code>
                {EXAMPLES[selectedTopic].code}
         </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

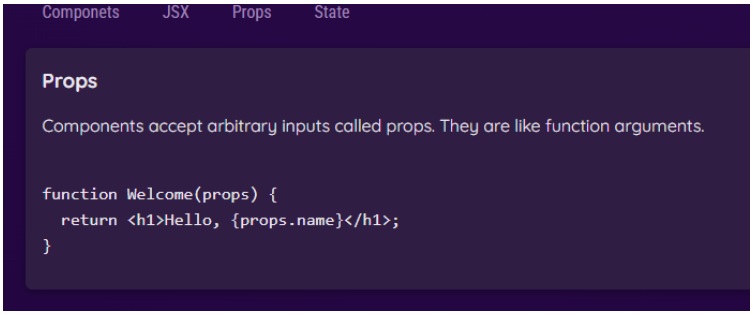
- Moreover EXAMPLES is declared as object like key value pairs
-

```
react > 01-starting-project > src > data.js > EXAMPLES > state > code
33  export const EXAMPLES = {
34    components: {
35      title: 'Components',
36      description:
37        'Components are the building blocks of React applications. A component is a self-cont
38      code: `
39  function Welcome() {
40    return <h1>Hello, World!</h1>;
41  }`,
42    },
43    jsx: {
44      title: 'JSX',
45      description:
46        'JSX is a syntax extension to JavaScript. It is similar to a template language, but i
47      code: `
48  <div>
49    <h1>Welcome {userName}</h1>
50    <p>Time to learn React!</p>
51  </div>`,
52    },
53    props: {
54      title: 'Props',
55      description:
56        'Components accept arbitrary inputs called props. They are like function arguments.',
57      code: `
58  function Welcome(props) {
59    return <h1>Hello, {props.name}</h1>;
60  }`,
61    },
62    state: {
63      title: 'State',
64      description:
65        'State allows React components to change their output over time in response to user a
66      code: `
67  function Counter() {
68    const [isVisible, setIsVisible] = useState(false);
```

- 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>);
```

```
  }
```

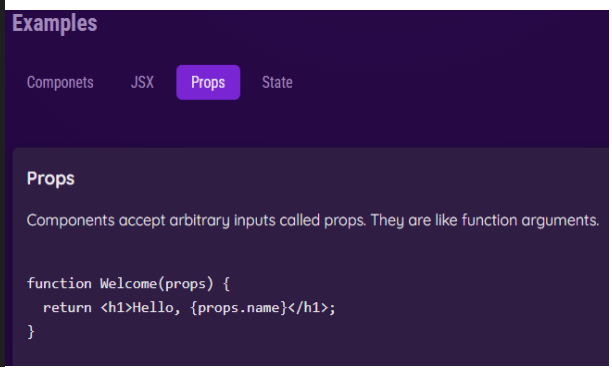
We can execute it by calling this variable by simply by using if statement and a variable. **{tabContent}** we will get same output as above

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

```
<section id ="examples">
  <h2>Examples</h2>
  <menu>
    <TabButton isSelected={selectedTopic ==='components'}
               onSelect={()=>handSelect('components')}
               >Componets
    </TabButton>
    <TabButton isSelected={selectedTopic ==='jsx'}
               onSelect={()=>handSelect('jsx')}
               >JSX
    </TabButton>
    <TabButton isSelected = {selectedTopic ==='props'}
               onSelect={()=>handSelect('props')}
               >Props
    </TabButton>
    <TabButton isSelected = {selectedTopic ==='state'}
               onSelect={()=>handSelect('state')}
               >State
    </TabButton>
```



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}/>
              )}
```