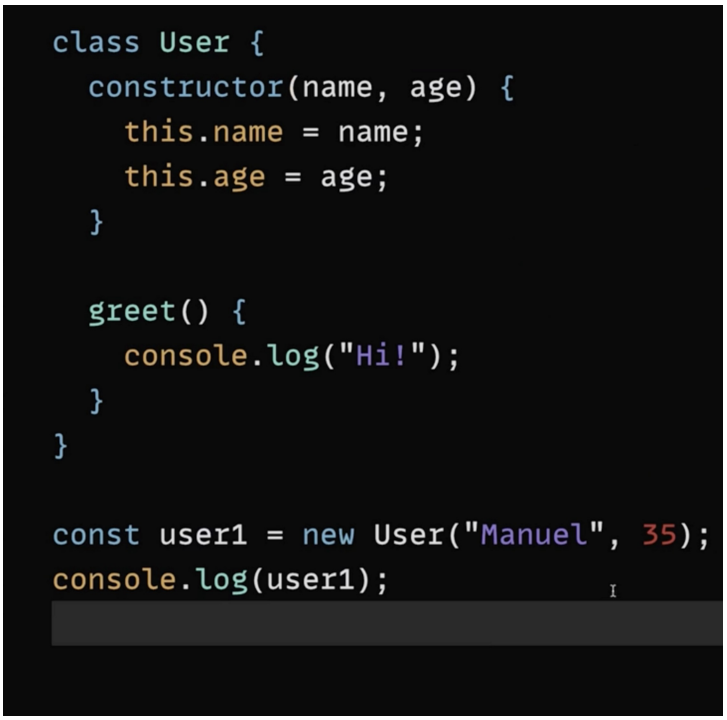
JS

Monday, April 29, 2024 12:20 PM

Objects :

```
//   return userName + messag
// };

const user = {
  name: "Max",
  age: 34,
  greet() {
    console.log("Hello!");
    console.log(this.age);
  }
};

console.log(user.name);
user.greet();
```

Classes:



```
class User {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  greet() {
    console.log("Hi!");
  }
}

const user1 = new User("Manuel", 35);
console.log(user1);
```

Array:

This type of function is known as arrow function:

```
const index = hobbies.findIndex((item) => item === "Sports");

console.log(index);
```

Short form of the above code., getting rid of the { and return

Map Method : Allows u to transform every item in the array to a another item. It wont change the default array will make a new array with the mentioned changes

```
const editedHobbies = hobbies.map((item) => item + "!");
console.log(editedHobbies);
```

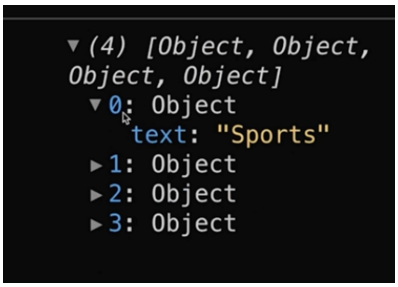
this makes a new array by adding ! To all the items in the array.

O. To make object from the array we can also use map

```
const editedHobbies = hobbies.map((item) => ({text: item}));
console.log(editedHobbies);
```

Here the ({}) parenthesis is used outside the curly bracket because then if not used then it would be treated as the parenthesis for the function body and not as a new javascript object returned by the arrow function

Console:



Destructing Array:

```
const [firstName, lastName] = ["Max", "Schwarzmüller"];

// const firstName = userNameData[0];
// const lastName = userNameData[1];

console.log(firstName);
console.log(lastName);
```

Destructing Object:

```
const { name: userName, age } = {
  name: "Max",
  age: 34
};

console.log(userName);
console.log(age);

// const name = user.name;
// const age = user.age;
```

Destruction in function Parameter

For example, if a function accepts a parameter that will **contain an object** it can be DE structured to *"pull out"* the object properties and make them available as **locally scoped variables** (i.e., variables only available inside the function body).

Here's an example:

```
function storeOrder(order) {
  localStorage.setItem('id', order.id);
  localStorage.setItem('currency', order.currency);
}
```

Instead of accessing the `order` *properties via the "dot notation" inside the* `storeOrder` *function body, you could use destructuring like this:*

```
function storeOrder({id, currency}) { // destructuring
  localStorage.setItem('id', id);
  localStorage.setItem('currency', currency);
}
```

it's very important to understand, that `storeOrder` ***still only takes one parameter*** *in this example! It does* ***not*** *accept two parameters. Instead, it's one single parameter - an* ***object*** *which then just is destructured internally.*

The function would still be called like this:

```
storeOrder({id: 5, currency: 'USD', amount: 15.99}); // one argument / value!
```

Spread operator: used to merge arrays :

Spread operation is also available in objects:

For Loop:

Time out function:

For Time out function we need two attributes, 1st the function to execute and the 2nd the time in milli second.

Here to pass a function in the Timeout function we need to pass it without the parenthesis, then it will be passed as a value and will be executed at the timer.

Else if passed with the parenthesis then it will execute at that moment and the value returned from the function will be go as a input argument for the Time out function.

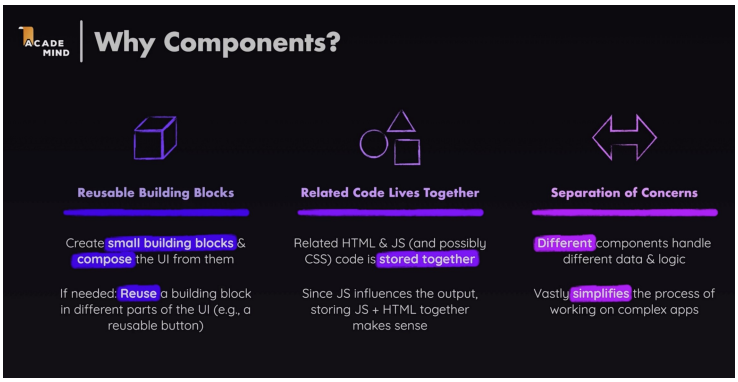
Now there are three method to make the function, mentioned below.

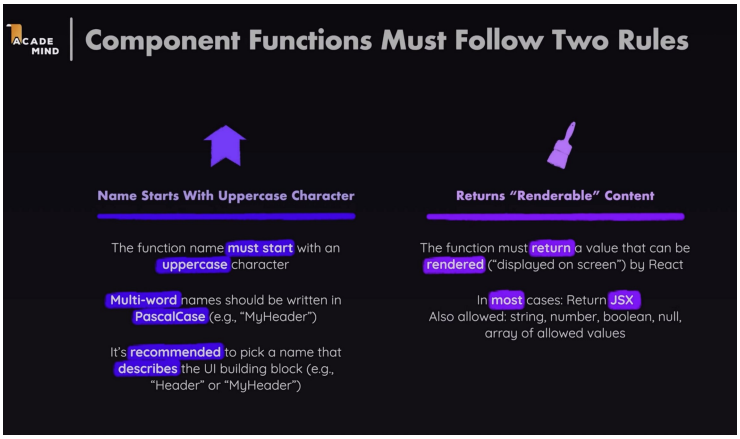
In last example the function is been defined and will be executed after the timer.

React Basic

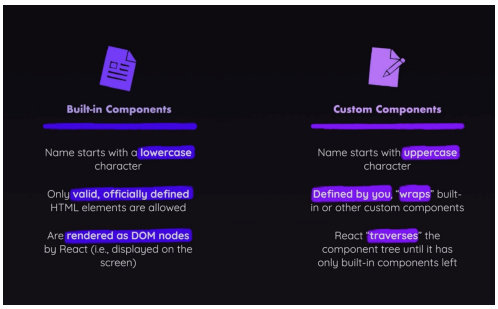
29 April 2024 12:28

Components





Custom components:




```jsx
function App() {
  return (
    <div>
      <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>

      <main>
        <h2>Time to get started!</h2>
      </main>
    </div>
  );
}

export default App;
```

Here we can get the header section in a function and call that function the main code.

This is know as custom components.

```jsx
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>
  );
}
```

```jsx
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>
      <main>
        <h2>Time to get started!</h2>
      </main>
```

Import image and save in a variable in the code

```jsx
import reactImg from './assets/react-core-concepts.png';
```

DYNAMIC IMAGE USING THE VARIABLE

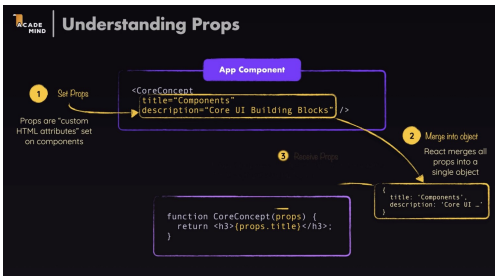
```jsx
function Header() {
  const description = reactDescriptions[genRandomInt(2)];

  return (
    <header>
      <img src={reactImg} alt="Stylized atom" />
      <h1>React Essentials</h1>
      <p>
        {description} React concepts you will need for almost any app you are
        going to build!
      </p>
    </header>
  );
}
```

Note : don't add quotes like "{reactImg}"

Dynamic value:

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

function genRandomInt(max) {
  return Math.floor(Math.random() * (max + 1));
}

function Header() {
  const description = reactDescriptions[genRandomInt(2)];

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

Props concept :



The function coreConcept is the base function which is used several time using component in app, and in this we only need to pass props as argument in coreConcept, not all the arguments, and then in the main body, we can change the details according to the need.

Note : when calling the function in the body and saving the data in the variable, then we should have the same variable name in the main function..

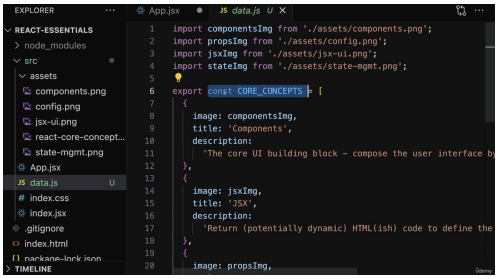
```jsx
function CoreConcept(props) {
  return (
    <li>
      <img src={props.image} alt={props.title} />
      <h3>{props.title}</h3>
      <p>{props.description}</p>
    </li>
  );
}

function App() {
  return (
    <div>
      <Header />
      <main>
        <section id="core-concepts">
          <h2>Core Concepts</h2>
```

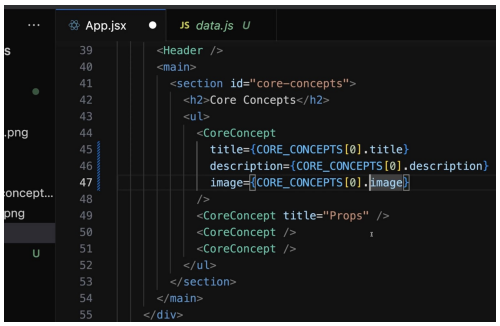
```jsx
function App() {
  return (
    <div>
      <Header />
      <main>
        <section id="core-concepts">
          <h2>Core Concepts</h2>
          <ul>
            <CoreConcept
              title="Components"
              description="The core UI building block."
              image={componentsImg}
            />
            <CoreConcept />
            <CoreConcept />
            <CoreConcept />
          </ul>
```

```jsx
import reactImg from './assets/react-core-concepts.png';
import componentsImg from './assets/components.png';
```

Saving image in a variable to use that variable later.

Reduce code in Prop :






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

This can be used to fetch all the variable from the array, when the variable name is the same as in the main function.

This reduces the code

Also we can use this syntax to not write props.title etc in each line.

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

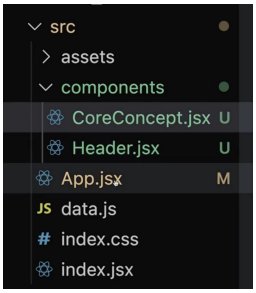
```jsx
function CoreConcept({image, title, description}) {
  return (
    <li>
      <img src={image} alt={title} />
      <h3>{title}</h3>
      <p>{description}</p>
    </li>
  );
}
```

Cleaning code:

Previously in the App.Jsx file all the components was there, we need to separate all the components into

its own file

Process :




```
function App() {
  return (
    <div>
      <Header />
      <main>
        <section id="core-concepts">
          <h2>Core Concepts</h2>
```

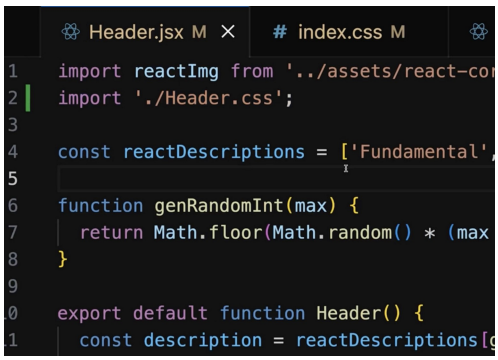
We put the header components in a different file and need to use export deault in front of the function and in the main App.jsx file we need to import the header component.

```
import Header from './components/Header.jsx';
```

Also in the header fucntion we are using image which is asset folder outside component folder, so we need to use ../ to get to that folder

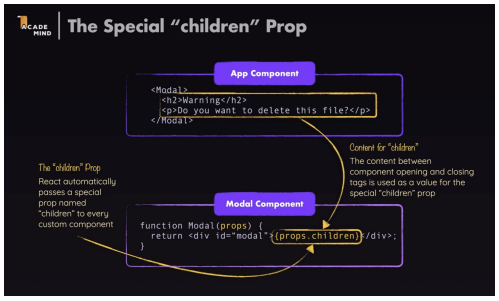
```
import reactImg from '../assets/react-core-concepts.png';
```

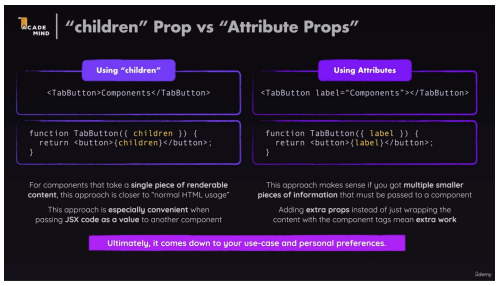
NOTE : the name of the file which we are creating should be the same as of the component name. 😇



Here we are making a header.css file and storing the css related to the header in the file and storing it in component folder.

Note : the css wont work until it is imported, so here in the header.jsx file we are importing header.css






```
</section>
<section id="examples">
  <h2>Examples</h2>
  <menu>
    <TabButton>Components</TabButton>
    <TabButton>JSX</TabButton>
    <TabButton>Props</TabButton>
    <TabButton>State</TabButton>

  </menu>
</section>
```

This is known as object destructuring.

The children prop is a special prop that's automatically provided to every component function. It contains the wrapped content as a value.
So the children prop's value for this code:

1. <Card>
2. Hi there
3. </Card>

would be Hi there .

Events:

```jsx
export default function TabButton({ children }) {
  function handleClick() {
    console.log('Hello World!');
  }

  return (
    <li>
      <button onClick={handleClick}>{children}</button>
    </li>
  );
}
```

Modifying to generate dynamic output:

App.jsx | CoreConcept.jsx | TabButton.jsx

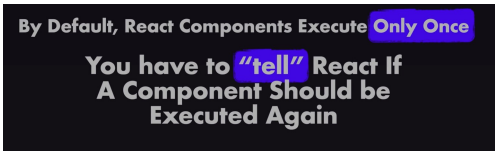
```jsx
import Header from './components/Header/Header.js
import CoreConcept from './components/CoreConcept
import TabButton from './components/TabButton.jsx

function App() {
  function handleSelect() {
    console.log('Hello World - selected!');
  }
```

```jsx
<section id="examples">
  <h2>Examples</h2>
  <menu>
    <TabButton onSelect={handleSelect}>Components</TabButton>
    <TabButton onSelect={handleSelect}>JSX</TabButton>
    <TabButton onSelect={handleSelect}>Props</TabButton>
    <TabButton onSelect={handleSelect}>State</TabButton>
  </menu>
```

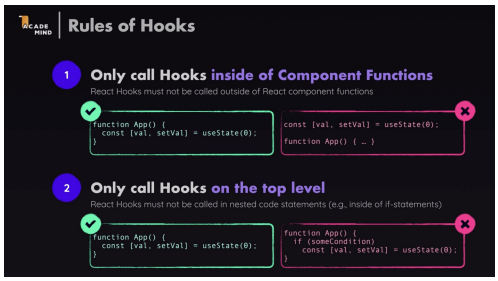
Now to output different things for different button:-

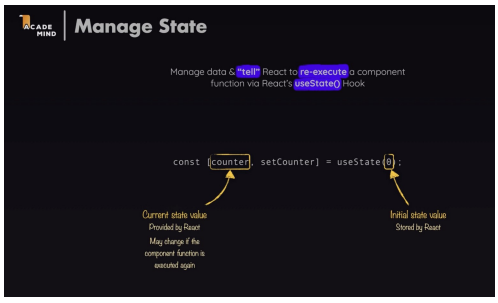
```jsx
function App() {
  function handleSelect(selectedButton) {
    // selectedButton => 'components', 'jsx', 'props', 'state'
    console.log(selectedButton);
  }
```

```jsx
<menu>
  <TabButton onSelect={() => handleSelect('components')}>
    Components
  </TabButton>
  <TabButton onSelect={() => handleSelect('jsx')}>JSX</TabButton>
  <TabButton onSelect={() => handleSelect('props')}>Props</TabButton>
  <TabButton onSelect={() => handleSelect('state')}>State</TabButton>
</menu>
```



Managing State & Using Hooks






```jsx
import { useState } from 'react';
```

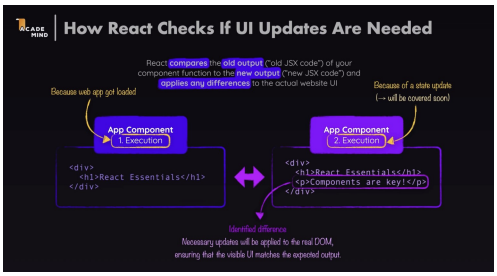
App.jsx | CoreConcept.jsx | TabButton.jsx

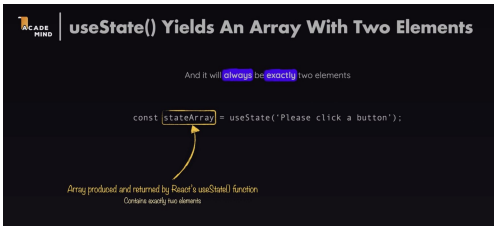
```jsx
export default function TabButton({ children, onSelect }) {
  return (
    <li>
      <button onClick={onSelect}>{children}</button>
    </li>
  );
}
```

Explanation:

onSelect goes to tabButton -> onClick triggers onSelect -> handleSelect function runs.

() => handleselect('components')} is used to avoid the code to Trigger when its executed, so the fake function gets executed and the handleSelect gets executed when the button is clicked.





Function starting with "use" are react hook

Note: when there is a change in the useState (hook function)
Then it will trigger the App component to re-run the component.

useState need a starting value which will show in the page,

component function is executed again

```
import { useState } from 'react';
```

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

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

```
  </menu>
  {selectedTopic}
</section>
```

Putting dynamic value, store in data.js file

```
  </menu>
  <div id="tab-content">
    <h3>{EXAMPLES[selectedTopic].title}</h3>
    <p>{EXAMPLES[selectedTopic].description}</p>
    <pre>
      <code>{EXAMPLES[selectedTopic].code}</code>
    </pre>
  </div>
</section>
```

If you want to show the content after the button is clicked then :

```
  {!selectedTopic ? (
    <p>Please select a topic.</p>
  ) : (
    <div id="tab-content">
      <h3>{EXAMPLES[selectedTopic].title}</h3>
      <p>{EXAMPLES[selectedTopic].description}</p>
      <pre>
        <code>{EXAMPLES[selectedTopic].code}</code>
      </pre>
    </div>
  )}
```

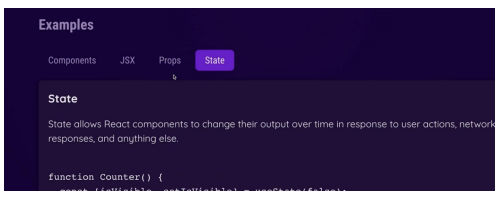
2nd approch

```
  {!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>
  )}
</section>
```

3rd approach

```
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>
  );
}
```

```
  {tabContent}
</section>
```



Note: when there is a change in the useState (hook function)

Then it will trigger the App component to re-run the component.

useState need a starting value which will show in the page,

And then when triggered, it will return a array with two values

Here we can use array destructuring to get the two value in two different variables.

Here the first value of the array will be the value with the useState is currently storing, it will return that value to the first value of the array.

The 2nd value will be a function provided by react that can be executed to update this state.(i.e it will make the app component to run again.

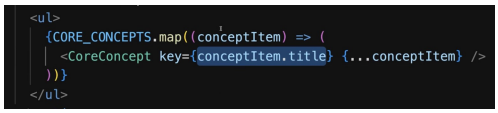
Therefore we will pass the first array value to get outputed.

To get this styling where its shows which button is selected

```
<menu>
  <TabButton
    isSelected={selectedTopic === 'components'}
    onSelect={() => handleSelect('components')}
  >
    Components
  </TabButton>
  <TabButton  onSelect={() => handleSelect('jsx')}>JSX</Ta
  <TabButton onSelect={() => handleSelect('props')}>Props<
```

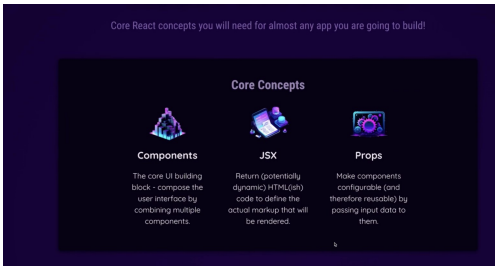
We take another attribute in TabButton function and when isSelected is true the class with the css styling gets active.

In the app component, we added a compare to push the isSelected button as true to the tabButton function.



Using key to uniquely identify a value.

```
import React from 'react';

import Todo from './Todo';
import './styles.css';

export const DUMMY_TODOS = [
    'Learn React',
    'Practice React',
    'Profit!'
];

// don't change the Component name "App"
export default function App() {
    return (
        <ul>
          {DUMMY_TODOS.map(todo => <Todo text={todo} />)}
        </ul>
    );
}
```



React adv

29 April 2024 12:29

General syntax : where a div is used to wrap all the sibling, but this adds an extra div, which can be avoided using this two methods:

1. Old syntax: fragment is used, which needs to be imported from react.
2. Here only <> is used. No import needed(Modern syntax)
3.

```
return (
  <div>
    <Header />
    <main>…
    </main>
  </div>
);
```

```
return (
  <Fragment>
    <Header />
    <main>…
    </main>
  </Fragment>
);
```

```
return (
  <>
    <Header />
    <main>…
    </main>
  </>
);
```

Instead of having all the components in the main app component, we should make individual file every component.

Final app component:

```
import Header from './components/Header/Header.jsx';
import CoreConcepts from './components/CoreConcepts.jsx';
import Examples from './components/Examples.jsx';

function App() {
  return (
    <>
      <Header />
      <main>
        <CoreConcepts />
        <Examples />
      </main>
    </>
  );
}

export default App;
```

Component:

```
import CoreConcept from './CoreConcept.jsx';
import { CORE_CONCEPTS } from '../data.js';

export default function CoreConcepts() {
  return (
    <section id="core-concepts">
      <h2>Core Concepts</h2>
      <ul>
        {CORE_CONCEPTS.map((conceptItem) => (
          <CoreConcept key={conceptItem.title} {...conceptItem} />
        ))}
      </ul>
    </section>
  );
}
```

```
import { useState } from 'react';

import TabButton from './TabButton.jsx';
import { EXAMPLES } from '../data.js';

export default function Examples() {
  const [selectedTopic, setSelectedTopic] = useState();

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

  let tabContent = <p>Please select a topic.</p>;
```

Here the section element is used several

Time, so creating a component for section:

```
export default function Section({ title, children }) {
  return (
    <section>
      <h2>{title}</h2>
      {children}
    </section>
  );
}
```

Import that in App.jsx

```
import Section from './Section.jsx';
```

Modify the section element into the section component :

```
return (
  <Section title="Examples" id="examples">
    <menu>
      <TabButton
        isSelected={selectedTopic === 'components'}
        onSelect={() => handleSelect('components')}
      >
        Components
      </TabButton>
      <TabButton
```

Forwarding Props:

Now to get all the classes in the component in that we need to use the prop:

```
export default function Section({ title, children, ...props }) {
  return (
    <section {...props}>
      <h2>{title}</h2>
      {children}
    </section>
  );
}
```

Using this for any number of classes, ID etc. will be called.

After updating:

```
export default function TabButton({ children, isSelected, ...props }) {
  console.log('TABBUTTON COMPONENT EXECUTING');
  return (
    <li>
      <button className={isSelected ? 'active' : undefined} {...props}>
        {children}
      </button>
    </li>
  );
}

<menu>
  <TabButton
    isSelected={selectedTopic === 'components'}
    onClick={() => handleSelect('components')}
  >
    Components
```

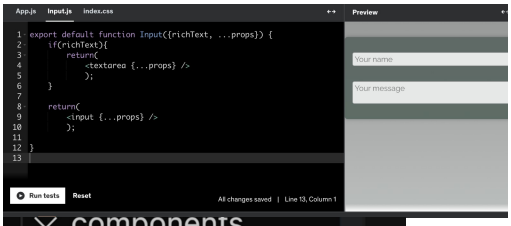
```
    >
      Components
    </TabButton>
    <TabButton
```

```
    isSelected={selectedTopic === 'components'}
    onClick={() => handleSelect('components')}
  >
    Components
```

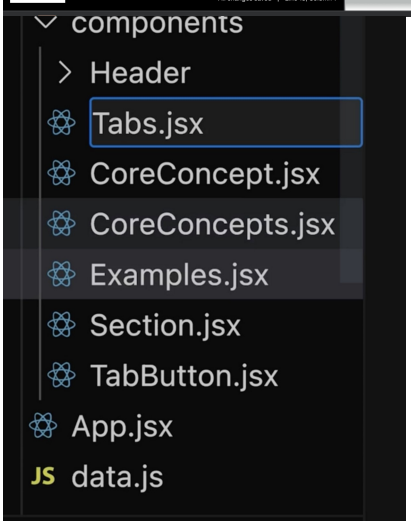
replace for all of them.

Same Forwarding props can be used in TabButton, so we don't have to use the onSelelct.

```
Examples.jsx M | Section.jsx U | TabButton.jsx ×
1  export default function TabButton({ children, onSelect, isSelected }) {
2    console.log('TABBUTTON COMPONENT EXECUTING');
3    return (
4      <li>
5        <button className={isSelected ? 'active' : undefined} onClick={onSe
6          {children}
7        </button>
8      </li>
9    );
10 }
11
```



JSX Slots




```
Examples.jsx M | Tabs.jsx U × | Section.jsx | TabButton.jsx
1  export default function Tabs({ children, buttons }) {
2    return (
3      <>
4        <menu>{buttons}</menu>
5        {children}
6      </>
7    );
8  }
```

Here we are making a common component tabs.jsx which can be reuse

```
Examples.jsx ● | Tabs.jsx U ● | Section.jsx | TabButton.jsx
30
31    return (
32      <Section title="Examples" id="examples">
33        <Tabs
34          buttons={
35            <>
36              <TabButton
37                isSelected={selectedTopic === 'components'}
38                onClick={() => handleSelect('components')}
39              >
40                Components
41              </TabButton>
42              <TabButton
43                isSelected={selectedTopic === 'jsx'}
44                onClick={() => handleSelect('jsx')}
45              >
```

```
              >
                Props
              </TabButton>
              <TabButton
                isSelected={selectedTopic === 'state'}
                onClick={() => handleSelect('state')}
              >
                State
              </TabButton>
            </>
        }
      >
        {tabContent}
      </Tabs>
      <menu></menu>
    </Section>
```

Using a extra prop to customize the component:

buttonContainer

```
Examples.jsx M ● | Tabs.jsx M ● | Section.jsx | TabButton.jsx
31    return (
32      <Section title="Examples" id="examples">
33        <Tabs
34          buttonsContainer={Section}
35          buttons={
36            <>
37              <TabButton
38                isSelected={selectedTopic === 'components'}
39                onClick={() => handleSelect('components')}
40              >
41                Components
42              </TabButton>
43              <TabButton
44                isSelected={selectedTopic === 'jsx'}
45                onClick={() => handleSelect('jsx')}
46              >
```

```
Examples.jsx M × | Tabs.jsx U | Section.jsx | TabButton.jsx
55                isSelected={selectedTopic === 'state'}
56                onClick={() => handleSelect('state')}
57              >
58                State
59              </TabButton>
60            </>
```

```
Components
</TabButton>
<TabButton
  isSelected={selectedTopic === 'jsx'}
  onClick={() => handleSelect('jsx')}
>
```

```
Examples.jsx M ● Tabs.jsx M ● Section.jsx TabButton.jsx
export default function Tabs({ children, buttons, buttonsContainer }) {
  return (
    <>
      <menu>{buttons}</menu>
      {children}
    </>
  );
}
```

```
Examples.jsx M ● Tabs.jsx M ● Section.jsx TabButton.jsx
export default function Tabs({ children, buttons, buttonsContainer }) {
  const ButtonsContainer = buttonsContainer;
  return (
    <>
      <ButtonsContainer>{buttons}</ButtonsContainer>
      {children}
    </>
  );
}
```

Default props values:

```
Examples.jsx ● Tabs.jsx ● Section.jsx TabButton.jsx
default function Tabs({ children, buttons, ButtonsContainer = 'menu' }) {
nst ButtonsContainer = buttonsContainer;
n (

ButtonsContainer>{buttons}</ButtonsContainer>
children}
```

```
            isSelected={selectedTopic === 'state'}
            onClick={() => handleSelect('state')}
          >
            State
          </TabButton>
        </>
      }
    >
      {tabContent}
    </Tabs>
  </Section>
);
}
```

```
<Section title="Examples" id="examples">
  <Tabs
    buttonsContainer="div"
    buttons={
      <>
```

The buttonContainer dynamicaly

If we want to call another component in the buttonContainer the we need to use {} and If we are using div or menu something like that then we have to use "".

Here in the Tabs.jsx component the buttonContainer wont get reconised as a component, because the system will try to fina a default component with the same name because it is starting with small letter

There are two solution for it:

1. Make a variable which start from the capital letter and assign that variable to that component, then call that variable as the component
2. Or else start the component name with capital letter.

New note:

App.jsx × Players.jsx

src > App.jsx > App

```
import Player from "./components/Players"
function App() {

  return (
    <main>
      <div id="game-container">
        <ol id="players">
          <Player intialName="player1" symbol="X" />
          <Player intialName="player2" symbol="O" />
        </ol>
        GAME BOARD
      </div>
      LOG
    </main>
  );
}

export default App
```

#calling the component player in the main app component.

App.jsx | Players.jsx

src > components > Players.jsx > Player > handleClick

```jsx
import { useState } from "react";

export default function Player({intialName, symbol}){
    const [playerName, setplayerName] = useState(intialName);
    const [isEditing, setIsEditing] = useState(false);
    function handleClick(){
        setIsEditing((editing) => !isEditing);
    }
    function handleChangeName(event){
        setplayerName(event.target.value);
    }
    let player = <span className="player-name">{playerName}</span>;
    let clicked="edit";
    if(isEditing) {
        clicked = "save";
        player= <input type="text" required value={playerName} onChange={handleChangeName} />;
    }
    return(
    <li>
    <span className="player">
    {player}
    <span className="player-symbol">{symbol}</span>
    </span>
    <button onClick={handleClick} >{clicked}</button>
    </li>
    );
}
```