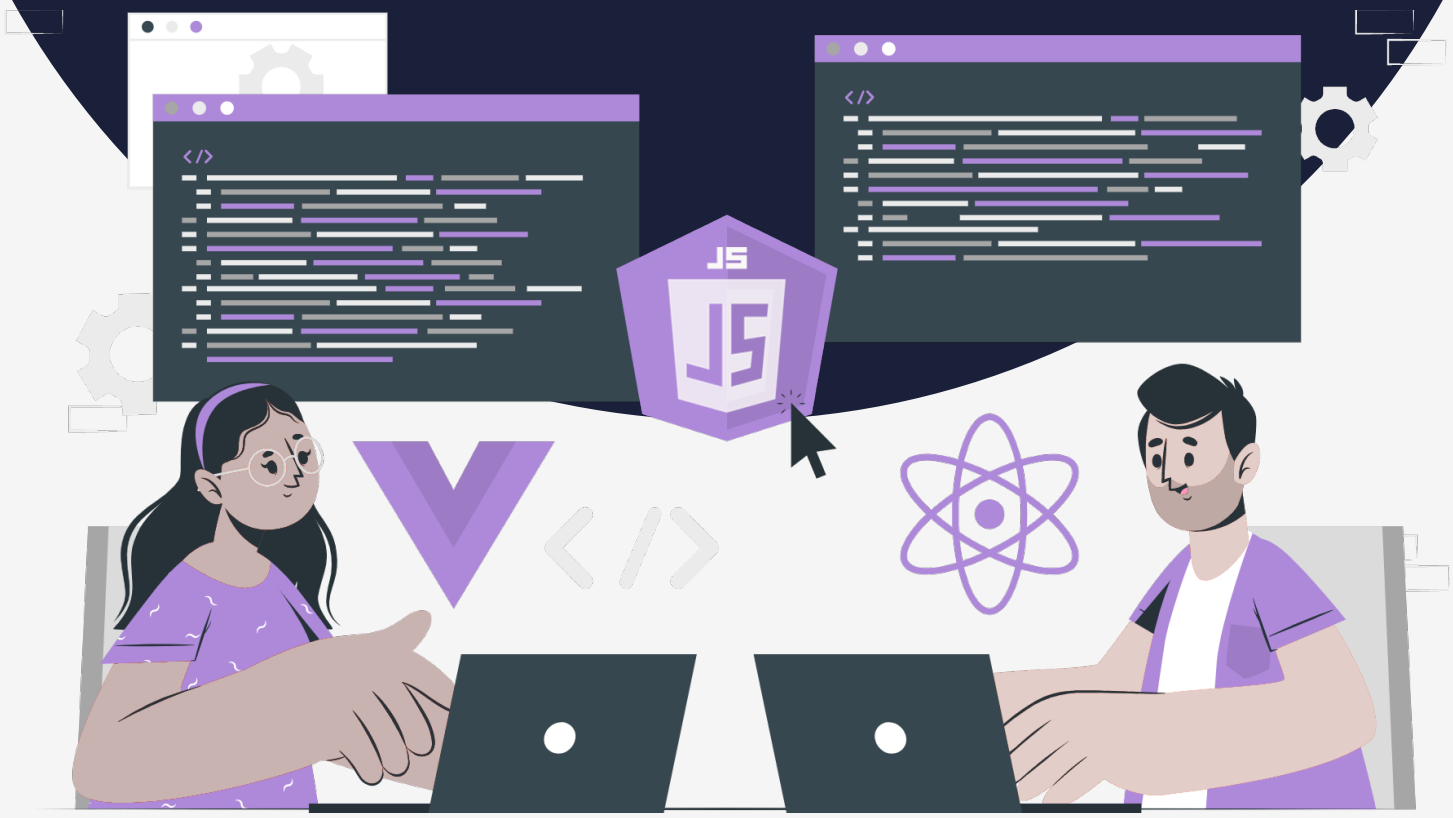
Lesson:

Methods of Conditional rendering



PW SKILLS

Topics Covered:

1. What is conditional rendering?
2. Methods of Conditional rendering in react
    a.Using if-else conditional operator
    b.rendering with null
    c.rendering with switch statement
3. Using Logical And(&&) and OR(||) operators.
4. Using Ternary Operators
5. How “&&” works.
6. Why to avoid “&&”.

What is conditional rendering ?
Conditional rendering is a technique in programming where a certain component or elements in a UI will be rendered (displayed) conditionally based on certain criteria such as props, state, or a particular condition. This allows for dynamic and flexible rendering of components, making it possible to show or hide content based on user interactions, or data from APIs or any other data source.

It is very important to render React components since it can make our app faster or slower depending on our approach for rendering components, re-rendering components based on condition or prop change.

Methods of Conditional rendering in React

1. **Using if-else conditional operator:**

Conditional rendering in React works similarly to the if-else statement in Javascript and each functional component returns a JSX value that is rendered. The following example shows how the conditional rendering occurs using if-else.

Let’s create a component named Welcome.jsx that contains the Jsx that will be rendered when the user LoggedIn.

```javascript
import React from 'react'

const Welcome = (props) => {
  return (
    <div>
        <h1>Welcome {props.name} to PW Skill</h1>
        <h1>Learn Anything..</h1>
    </div>
  )
}

export default Welcome
```

Create another component **NotLoggedIn.jsx** that will be rendered when the user is not LoggedIn.

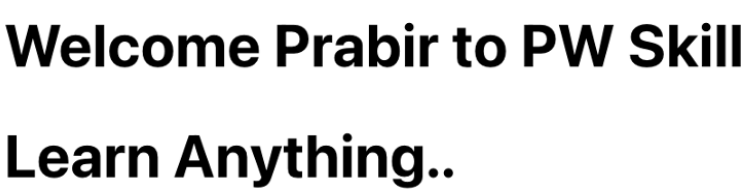
```JavaScript
import React from 'react'

const NotLoggedIn = () => {
  return (
    <div>
      <h1>Please login again</h1>
    </div>
  )
}

export default NotLoggedIn
```

Then import the two components into **App.js** file and add an **if-else** condition before the return statement that will assign either of the components to the **{data}** depending on whether the isLoggedIn state is true or false.

```JavaScript
import Welcome from './components/Welcome';
import NotLoggedIn from './components/NotLoggedIn';
import  { useState } from 'react';

      function App() {
  const[isLoggedIn, setIsLoggedIn]= useState(true)
  let data;

  if(isLoggedIn){
    data = <Welcome name="Prabir"/>
  }
    else{
      data = <NotLoggedIn/>
    }

  return (
      <div>{data}</div>
 );
}

export default App;
```

When the user logged in the state is set to true and the welcome message from Welcome component is displayed to the user as follows:

Output:



If the user is not logged in, the state is set to false and the following message is displayed.

Output:

2. Rendering with null:

When rendering JSX, conditional rendering in React gives you the option of not rendering a specific piece of content or anything to your users.
To implement such functionality in React, use 'null' as the rendered template. Using 'null' will result in nothing being rendered and will also prevent errors due to no template being returned.

Let's create a component named **Welcome.jsx** that contains the Jsx that will be rendered when the user LoggedIn

```javascript
import React from 'react'

const Welcome = (props) => {
  return (
    <div>
        <h1>Welcome {props.name} to PW Skill</h1>
        <h1>Learn Anything..</h1>
    </div>
  )
}
export default Welcome
```

Then import the component into App.js.

```
JavaScript
import Welcome from './components/Welcome';
import  { useState } from 'react';

      function App() {
  const[isLoggedIn, setIsLoggedIn]= useState(true)
  let data;

  if(isLoggedIn){
    data = <Welcome name="Prabir"/>
  }
    else{
      data = null;
    }

  return (
<div>
{data}


</div>
 );
}

export default App;
```

If the state is true, then the Welcome component is rendered as below:

Welcome Prabir to PW Skill

Learn Anything..

If the isLoggedIn state variable is false, then UI is rendered as nothing.

3. Conditional rendering with switch statement:

Sometimes we want to show a different UI to users based on the state of the application.
switch statements are not included in the syntax of JSX.So they can't be used directly within React.We can, however, use the Switch statements in a subcomponent before using the component in the main component.

Let's take an example, We will use switch statements to render different content based on a specific case. We will create an input that takes the user's value and renders the components the user requests.

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

 function Details(props){
  switch(props.route) {
    case "home":
      return <h1>You are in the Home Page</h1>;
    case "aboutUs":
      return <h1>You are in the About us page</h1>;
    case "contactUs":
      return <h1>You are in the contact page</h1>;
    default:
      return null;

  }
}

export default function App(){
  const [path, setPath] = useState("")
  return(
    <div className='App'>
      <input onChange={(e)=> setPath(e.target.value)}/>
      <Details route={path}/>
    </div>
  )
}
```

The component will render based on what the user enters

Output:

You are in the Home Page

3. Using Logical AND(&&) (Short Circuit Evaluation)

When the left-hand expression returns false, the right-hand expression is evaluated and returned. If the left-hand expression is false, nothing is returned.

For example, suppose We have a store application; when the store is open, it should return true and render a specific message to users; when the store is closed, it should return false and render nothing.

```javascript
export default function App() {

  const isOpen = true;
  return (
    <div>
      {isOpen && (<h1>Hello, How I can help you?</h1>)}
    </div>
  );
}
```

4. Using Ternary operators

The ternary operator is synonymous with the 'if-else' operator.

It has the following syntax:

```javascript
condition ? expression1 : expression2
```

The condition is evaluated. If it's truthy, expression1 is returned; otherwise, expression2 is returned.
Let's take an example using ternary operators

```javascript
function Display({isVisible}) {
  return (
    <div>
      {isVisible ? (
        <p>This element is visible.</p>
      ) : (
        <p>This element is hidden.</p>
      )}
    </div>
  );
}
```

In this example, the component Display takes a prop isVisible which is used to determine whether to show or hide an element. The ternary operator ? : checks the value of isVisible and returns one of two elements accordingly.

How "&&" works:

```javascript
function MyComponent({ condition }) {
  return (
    <div>
      <h1>Title</h1>
      {condition && <ConditionalComponent />}
    </div>
  );
}
```

- if condition is a truthy value, **<ConditionalComponent /> is rendered.**
- if condition is a falsy value, **<ConditionalComponent /> is not rendered.**

if the first operand **(condition)** is falsy, the AND operator (&&) stops and it does not evaluate the second operand **(<ConditionalComponent/>)**.

```javascript
// This will display an alert box.
true && alert('PW SKILL');
// This won't do anything.
false && alert('Nothing');
```

Why to avoid "&&"

In the above example, if the condition is true or false, We get what we would expect - **<ConditionalComponent />** is or is not rendered respectively.However, if condition doesn't evaluate to a boolean, it may cause trouble.

- if condition is 0, 0 is displayed in the UI.
- if condition is undefined, you'll get an error: "Uncaught Error: Error(...): Nothing was returned from render. This usually means a return statement is missing. Or, to render nothing, return null."

Note: To prevent avoidable UI bugs, we can use the javascript ternary operator for conditional rendering of React components instead of the logical AND operator.