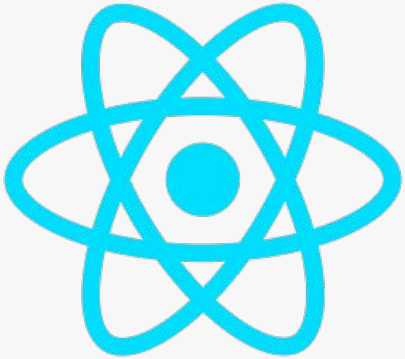
Newton School of Technology

The Ultimate React Course



@newtonschool

Lecture 14: JSX, Interpolation, and Rendering

- Narendra Kumar

Prerequisites

- Setting up React dev environment
- Understanding of project structure in react
- React Functional Components
- Basic Understanding of JSX
- Launching React Application in browser using terminal

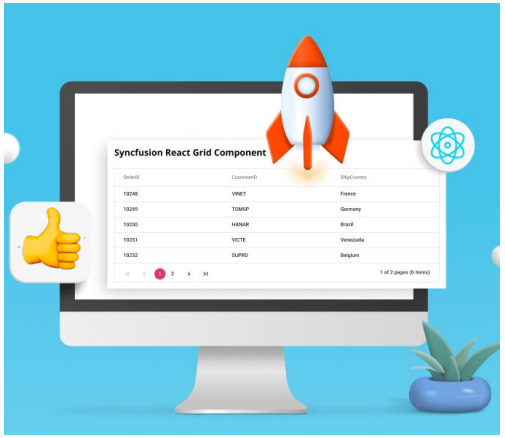
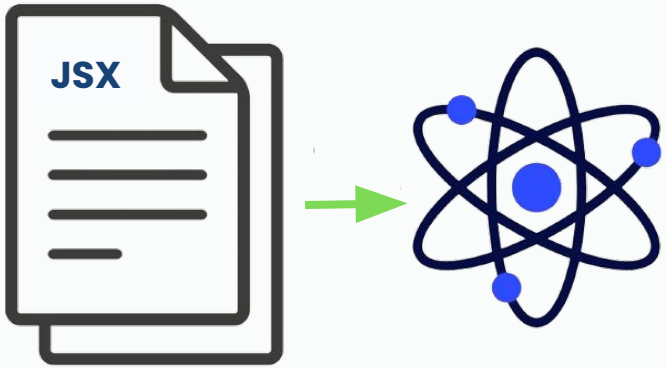
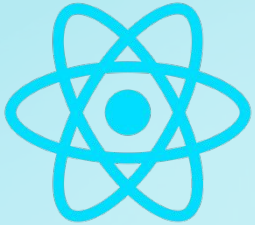


Table of Contents



Newton School of Technology



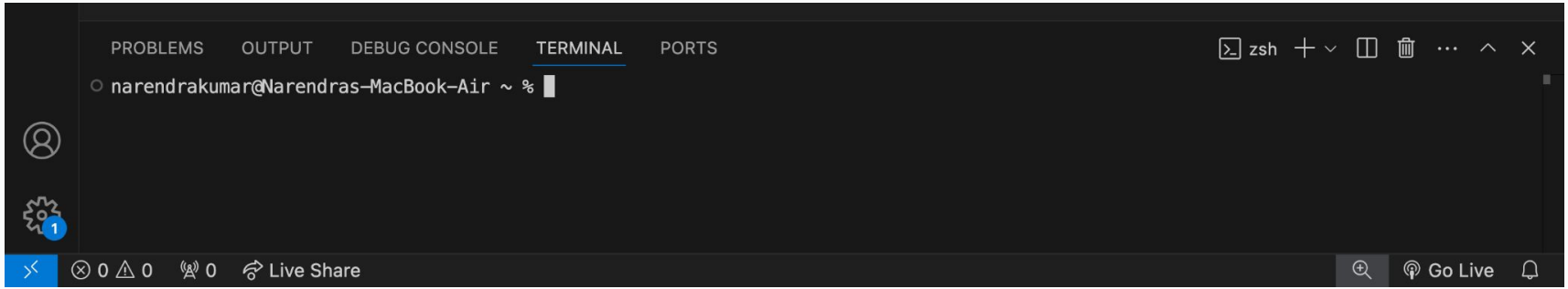
Quick Recap

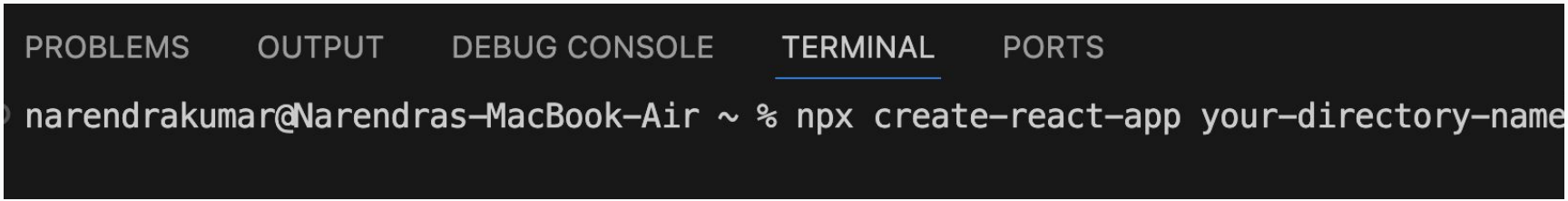
A brief Overview

Pre-discussed: Creating a React App

Newton School of Technology

Terminal would look like this:



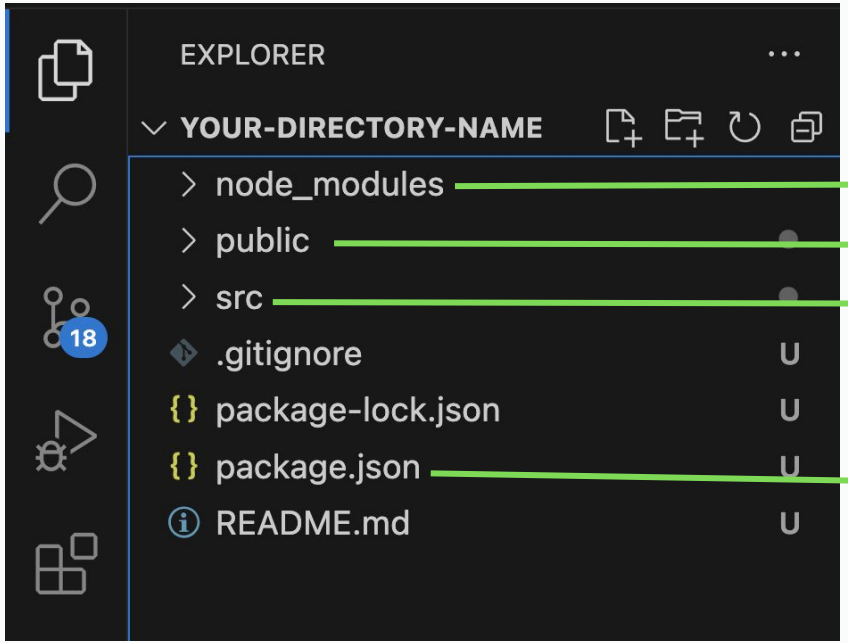


Run create-react-app command with directory name next to it

Pre-discussed: Project Structure

Newton School of Technology

In File Explorer, you can see multiple folders and files being created. Let's understand each one, one by one:



Contains all third-party dependencies installed via npm or yarn.

Contains static files directly served by the browser.

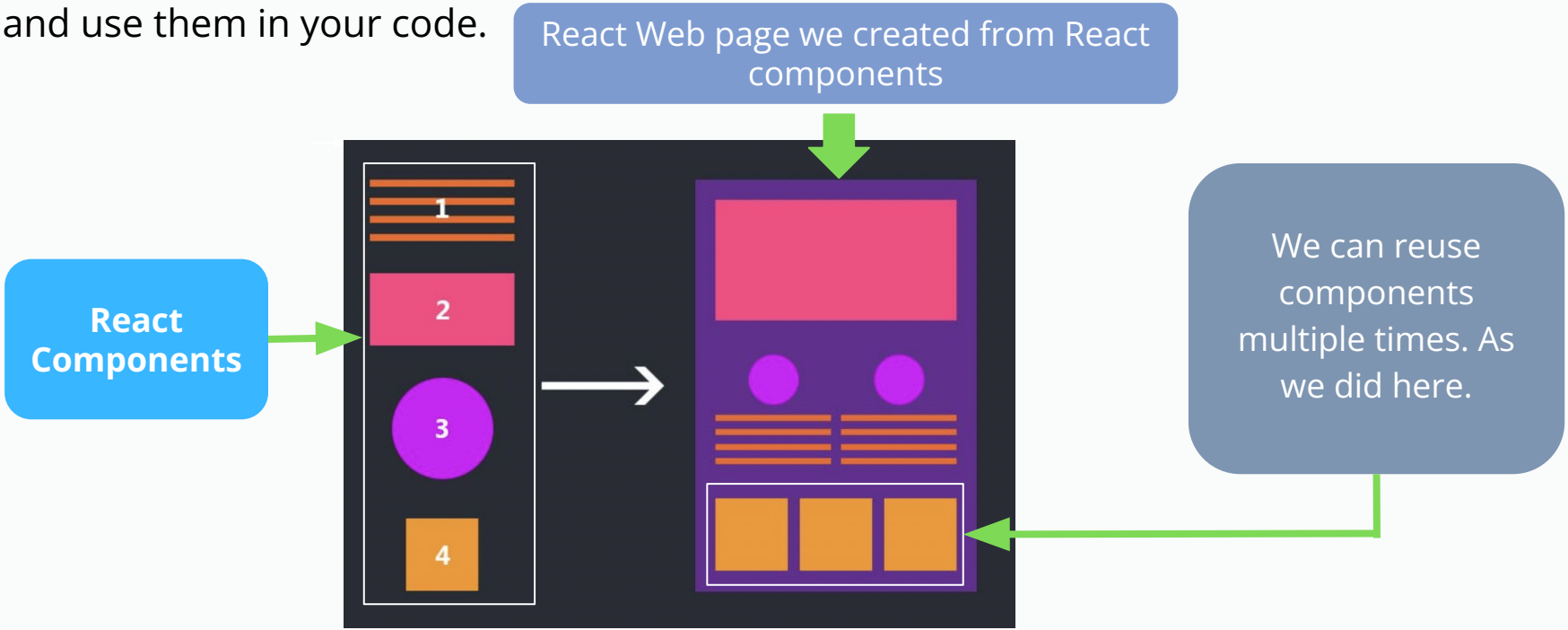
Contains all the application code

Lists dependencies, scripts, and project metadata.

Pre-discussed: Components

Newton School of Technology

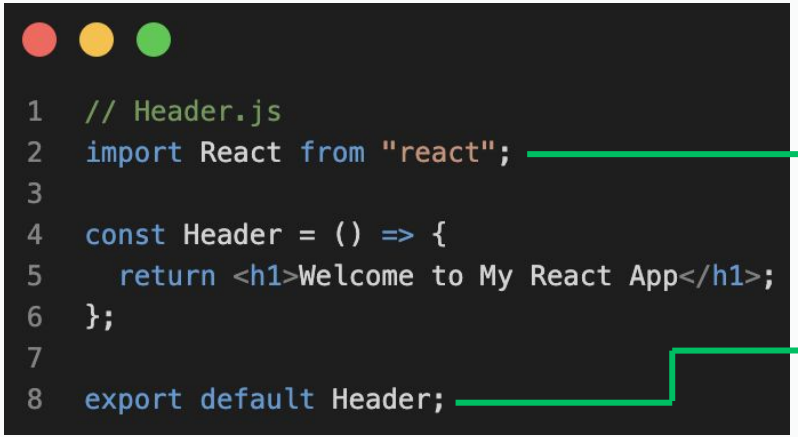
React follows a component-based structure, where you build the components once and use them in your code.



Pre-discussed: Creating Header

Newton School of Technology

Let’s create a component named Header.jsx:-



```jsx
// Header.js
import React from "react";

const Header = () => {
  return <h1>Welcome to My React App</h1>;
};

export default Header;
```

Although not explicitly used in the function, **React** is still required to import.

We need to export component to utilize it in some other place

Pre-discussed: Including Header in App.js

We can simply import it in App.js file and use it here:

```
1  // App.js
2  import React from "react";
3  import Header from "./Header";
4
5  const App = () => {
6    return (
7      <div>
8        <Header />
9        <p>This is the main content of the app.</p>
10     </div>
11   );
12 };
13
14 export default App;
```

While importing, we have used relative path with respect to location of App.js

We can include our components using custom JSX tags, similar to HTML elements.

Newton School of Technology

Introduction to JSX

What is JSX?

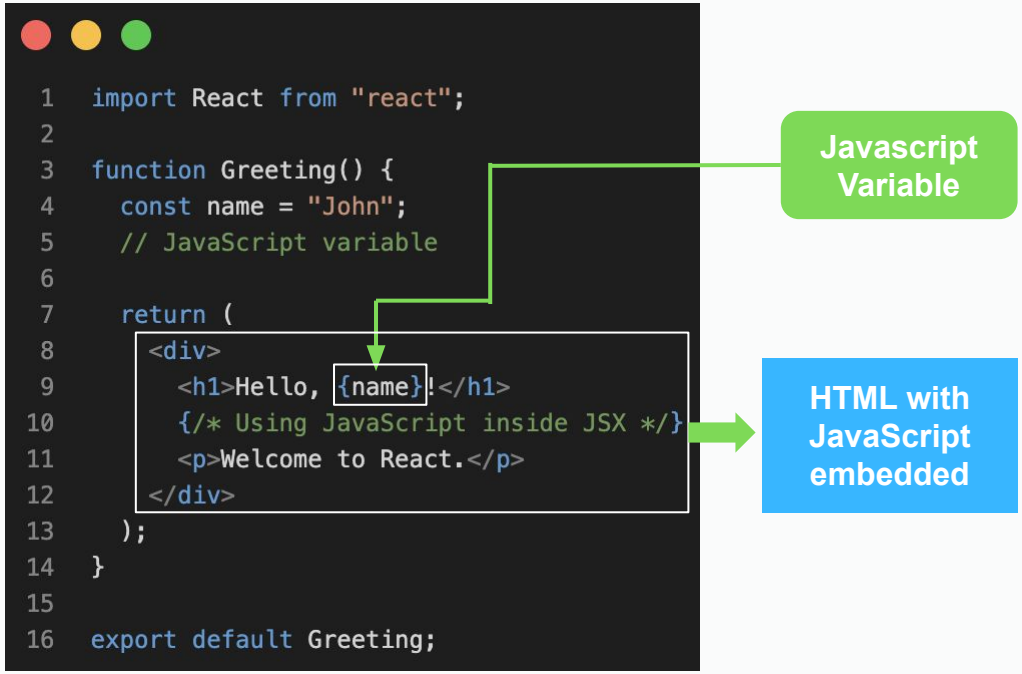
Newton School of Technology

JSX is a syntax extension for JavaScript that lets you write HTML-like markup inside a JavaScript file.

JSX is basically javascript embedded inside HTML code.

Let's Understand it with an example

Here in this code, the entire snippet inside the return statement is JSX.

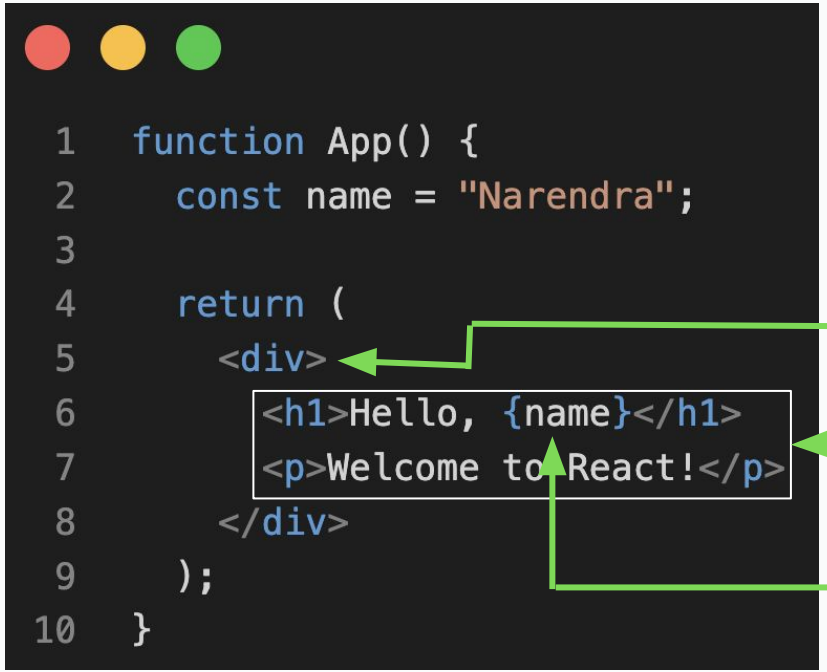


Anything that evaluates to a **primitive value, an array, or a function call returning a valid JSX-friendly value** can be placed inside JSX.

How do I write JSX code?

Newton School of Technology

We have simple rules for writing JSX:



```jsx
function App() {
  const name = "Narendra";

  return (
    <div>
      <h1>Hello, {name}</h1>
      <p>Welcome to React!</p>
    </div>
  );
}
```

Child elements inside the root element.

Rules for writing JSX

JSX must have a **single root element**.

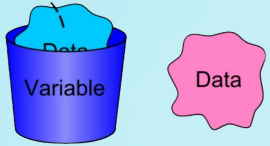
Use **curly braces {}** to embed Javascript.

Using fragment <>..</> instead of

Many a time we use fragment <>...</> instead of .. tag of HTML.

```
1  function App() {
2    const name = "Narendra";
3
4    return (
5      <>
6        <h1>Hello, {name}</h1>
7        <p>Welcome to React!</p>
8      </>
9    );
10 }
```

We use fragments instead of ... to avoid default HTML styling.



Variables in JSX

Why use variables inside JSX?

Using variables in JSX makes apps faster, keeps code organized, and updates data easily, like live scores or user info. 🚀

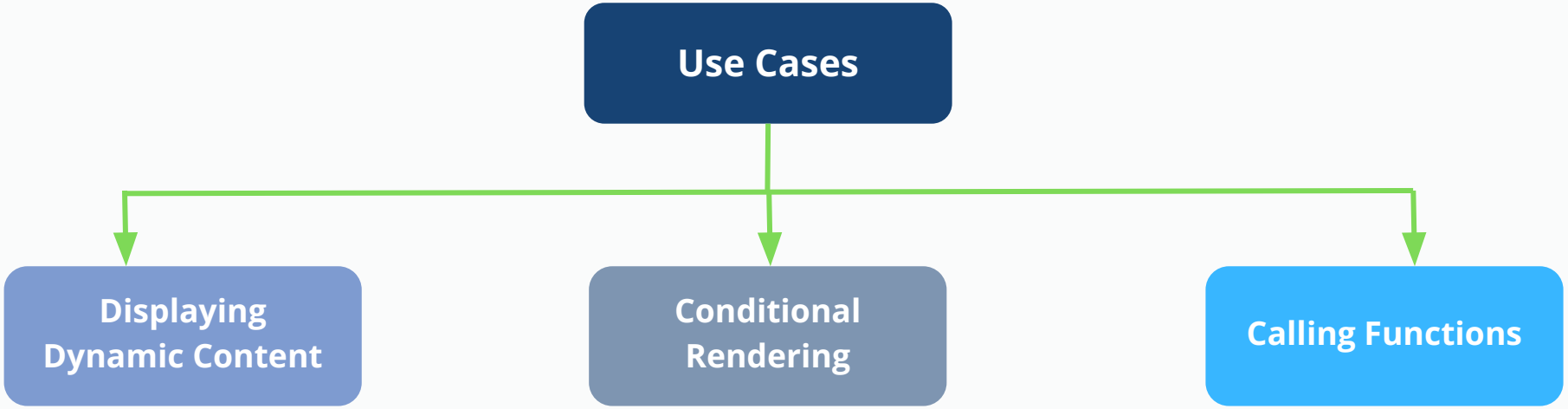
```jsx
function App() {
    const team_A_score = 45;
    const team_B_score = 38;

    return (
      <div>
        <h1>Team A Score: {team_A_score}</h1>
        <h1>Team B Score: {team_B_score}</h1>
      </div>
    );
}

export default App;
```

Here we are using variables to store values and using those values inside our JSX.

Use cases of variables in JSX

Newton School of Technology

JSX makes building UI easier and more efficient for variety of use cases.



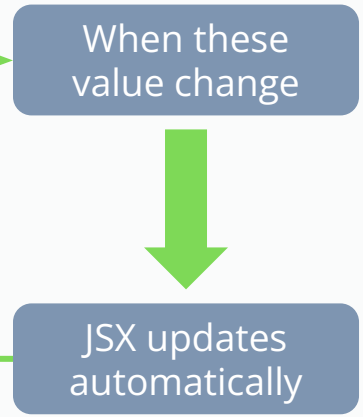
And many more...

JSX: Displaying Dynamic Content

Newton School of Technology

In the previous example, we stored values in variables and used them in JSX. When these values change, JSX updates automatically. 🚀

```jsx
function App() {
  const team_A_score = 45;
  const team_B_score = 38;

    return (
      <div>
        <h3>Team A Score: {team_A_score}</h3>
        <h3>Team B Score: {team_B_score}</h3>
      </div>
    );
}

export default App;
```



However, these updates are not guaranteed, so we use hooks.

Don’t worry, we will learn about hooks later.

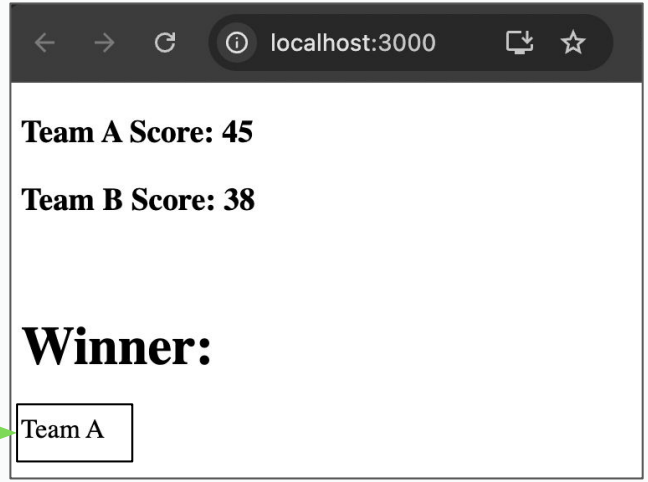
JSX: Conditional Statements

Newton School of Technology

Just like variables, we can also embed conditional statements inside JSX inside curly braces.

```jsx
function App() {
    const team_A_score = 45;
    const team_B_score = 38;

    return (
      <div>
        <h1>Team A Score: {team_A_score}</h1>
        <h1>Team B Score: {team_B_score}</h1>
        <br />
        <h1>
            Winner:
        </h1>
        <p>{team_A_score > team_B_score ? "Team A" : "Team B"}</p>
      </div>
    );
}

export default App;
```

Conditionally Rendering Winner



JSX: Calling Functions

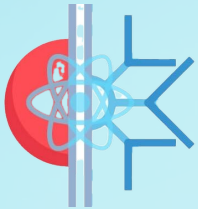
Instead of using conditional statements inside JSX, we can directly call a function.

```
function evaluateWinner(){
  return (
    team_A_sore > team_B_score ? "Team A" : "Team B"
  );
}
```

We may directly call a function inside JSX.

```
1  function App() {
2    const team_A_score = 45;
3    const team_B_score = 38;
4
5    return (
6        <div>
7          <h1>Team A Score: {team_A_score}</h1>
8          <h1>Team B Score: {team_B_score}</h1>
9          <br />
10         <h1>
11             Winner:
12         </h1>
13         <p>{evaluateWinner(team_A_score, team_B_score)}</p>
14       </div>
15     );
16 }
17
18 export default App;
```

Newton School of Technology

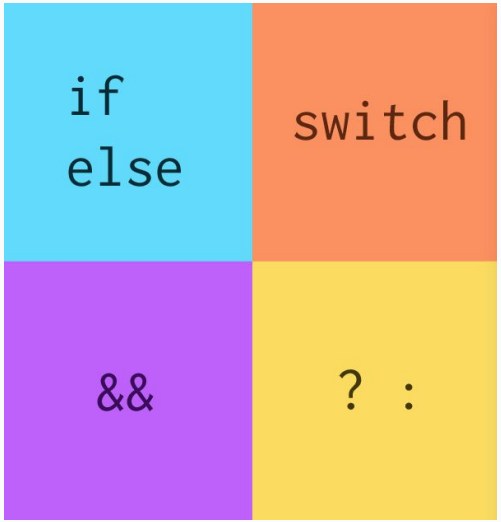


Conditional Rendering

What is conditional rendering?

Newton School of Technology

Conditional rendering in JSX controls what UI elements appear based on conditions like user login status or data availability, similar to if-else in JavaScript.



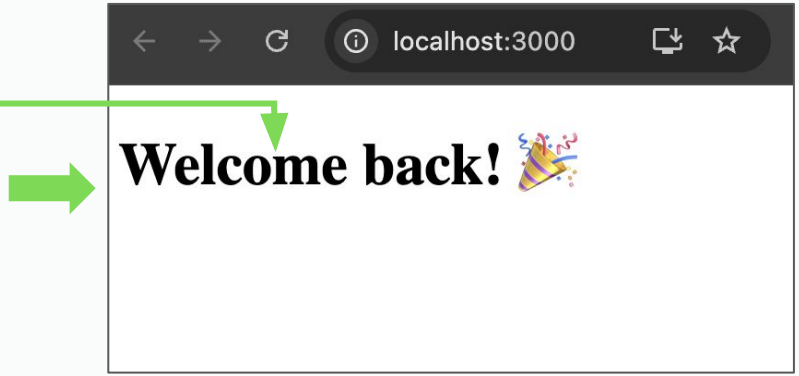
We can use the following structures to conditionally render in a ReactJS app.

Conditional Rendering: if...else

Newton School of Technology

In React, if-else decides what to render, useful for returning different UI elements based on conditions.

```jsx
function App() {
  const isLoggedIn = true;
  // For now we are setting logged status true

  if (isLoggedIn) {
    return <h1>Welcome back! 🎉</h1>;
  } else {
    return <h1>Please log in 🔑</h1>;
  }
}

export default App;
```



Conditional Rendering: if...else

Newton School of Technology

If isLoggedIn is set to false, a different message appears as the login status changes.

```
1  function App() {
2    const isLoggedIn = false;
3    // For now we are setting logged status true
4
5    if (isLoggedIn) {
6      return <h1>Welcome back! 🎉</h1>;
7    } else {
8      return <h1>Please log in 🔑</h1>;
9    }
10 }
11
12 export default App;
```

Conditional Rendering: switch

Newton School of Technology

The switch statement is a better choice when handling multiple conditions in JSX, as too many if-else statements can make the code messy and harder to read.

```jsx
function App() {
  const userRole = "admin";
  // It can also be "user" or "guest"

  let message;

  if (userRole === "admin") {
    message = <h1>Welcome, Admin! 👑</h1>;
  } else if (userRole === "user") {
    message = <h1>Welcome, User! 🚀</h1>;
  } else if (userRole === "guest") {
    message = <h1>Welcome, Guest! 🔑</h1>;
  } else {
    message = <h1>Unknown Role ❌</h1>;
  }

  return <div>{message}</div>;
}

export default App;
```

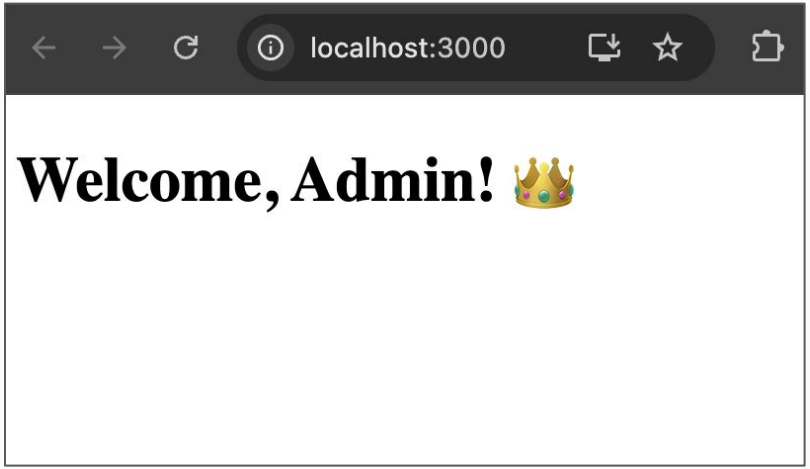
```jsx
function App() {
  const userRole = "admin";
  // It can also be "user" or "guest"

  function getRoleMessage(role) {
    switch (role) {
      case "admin":
        return <h1>Welcome, Admin! 👑</h1>;
      case "user":
        return <h1>Welcome, User! 🚀</h1>;
      case "guest":
        return <h1>Welcome, Guest! 🔑</h1>;
      default:
        return <h1>Unknown Role ❌</h1>;
    }
  }
  return <div>{getRoleMessage(userRole)}</div>;
}
export default App;
```

Which version of code looks cleaner to you?

Conditional Rendering: switch

Let’s have a look of it at the browser:

```jsx
function App() {
  const userRole = "admin";
  // It can also be "user" or "guest"

  function getRoleMessage(role) {
    switch (role) {
      case "admin":
        return <h1>Welcome, Admin! 👑</h1>;
      case "user":
        return <h1>Welcome, User! 🚀</h1>;
      case "guest":
        return <h1>Welcome, Guest! 🔑</h1>;
      default:
        return <h1>Unknown Role ❌</h1>;
    }
  }
  return <div>{getRoleMessage(userRole)}</div>;
}
export default App;
```



Conditional Rendering: switch

Newton School of Technology

Let’s try changing it to user this time:

```
function App() {
  const userRole = "user";
  // It can also be "user" or "guest"

  function getRoleMessage(role) {
    switch (role) {
      case "admin":
        return <h1>Welcome, Admin! 👑</h1>;
      case "user":
        return <h1>Welcome, User! 🚀</h1>;
      case "guest":
        return <h1>Welcome, Guest! 🔑</h1>;
      default:
        return <h1>Unknown Role ❌</h1>;
    }
  }
  return <div>{getRoleMessage(userRole)}</div>;
}
export default App;
```

Conditional Rendering: AND (&&)

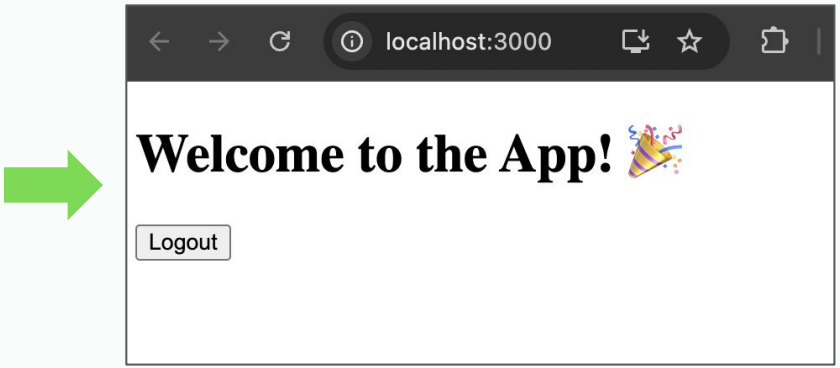
Newton School of Technology

The logical AND (&&) operator is a way to conditionally render elements in JSX. It works when you **only need to display something if a condition is true** without needing an `else`

```jsx
function App() {
  const isLoggedIn = true;
  // Change to false to hide the button

  return (
    <div>
      <h1>Welcome to the App! 🎉</h1>
      {isLoggedIn && <button>Logout</button>}
    </div>
  );
}

export default App;
```

isLoggedIn is true

Renderable only if



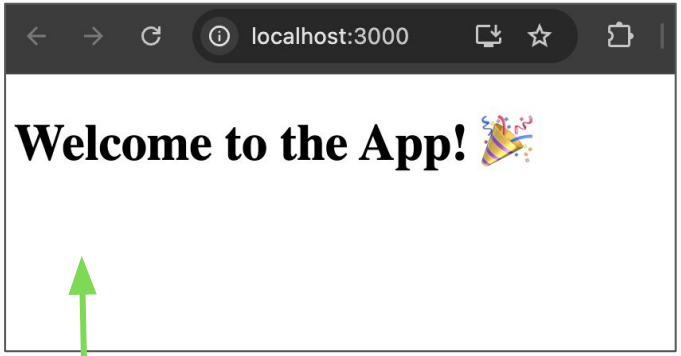
Conditional Rendering: AND (&&)

Newton School of Technology

Let’s change isLoggedIn to false for a moment:

```
1   function App() {
2     const isLoggedIn = false;
3     // Change to false to hide the button
4
5     return (
6       <div>
7         <h1>Welcome to the App! 🎉</h1>
8         {isLoggedIn && <button>Logout</button>}
9       </div>
10    );
11  }
12
13  export default App;
```

isLoggedIn decides rendering of button



Since isLoggedIn is false, <button> has gotten hidden

Conditional Rendering: Ternary (? :)

Need a fallback when isLoggedIn is false? Use a ternary operator: show "Logout" if true, else "Login."

```
1  function App() {
2    const isLoggedIn = false;
3    // Change to false to hide the button
4
5    return (
6      <div>
7        <h1>Welcome to the App! 🎉</h1>
8        {isLoggedIn ?
9          <button>Logout</button> :
10         <button>Login</button>
11       }
12     </div>
13   );
14 }
15
16 export default App;
```

If *isLoggedIn* is true, then Logout button would appear

If *isLoggedIn* is false, then Login button would appear

Newton School of Technology

Loop Rendering in JSX

Why use loops in JSX?

Newton School of Technology

Often, we need to dynamically generate lists of items. Since JSX doesn't support for loops directly inside {}, we use array methods like map() to loop over data.

✅ Displaying a list of products
✅ Rendering menu items
✅ Showing a leaderboard

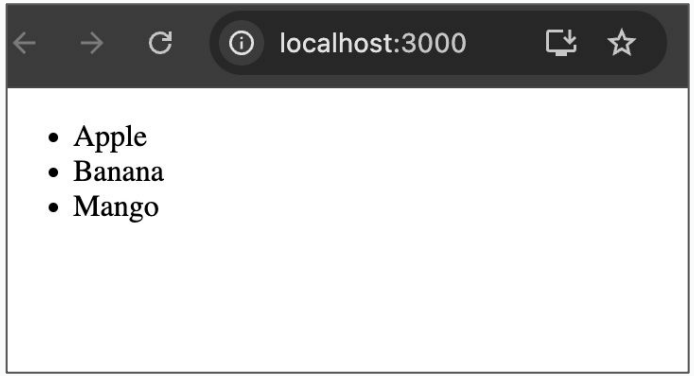
Common Use Cases

```jsx
function App() {
  const fruits = ["Apple", "Banana", "Mango"];
  return (
    <ul>
      {fruits.map((fruit) => {
          return (
            <li>
              {fruit}
            </li>
          );
        }
      )}
    </ul>
  );
}

export default App;
```

Rendering a list of items

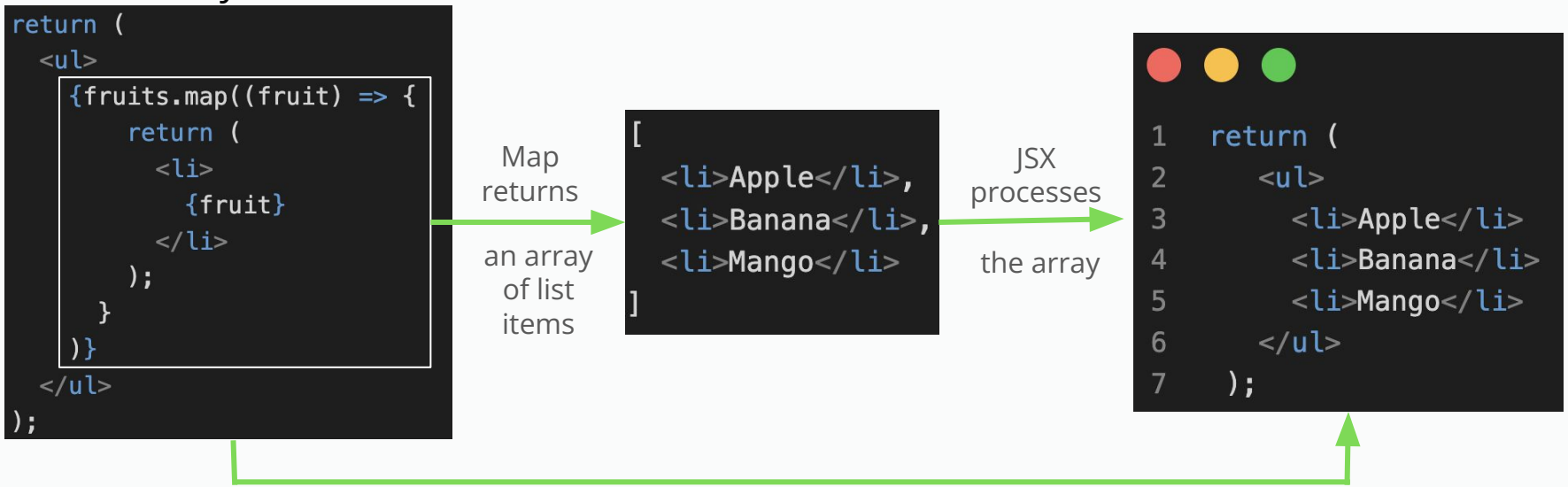
Here we have an array, and we are iterating over that array using the map() function of the array.

```jsx
function App() {
  const fruits = ["Apple", "Banana", "Mango"];
  return (
    <ul>
      {fruits.map((fruit) => {
          return (
            <li>
              {fruit}
            </li>
          );
        }
      )}
    </ul>
  );
}

export default App;
```



How it works?

The map function returns an array of list items, which JSX can directly render inside elements like <ul> or . JSX supports rendering arrays, so each item is displayed automatically.

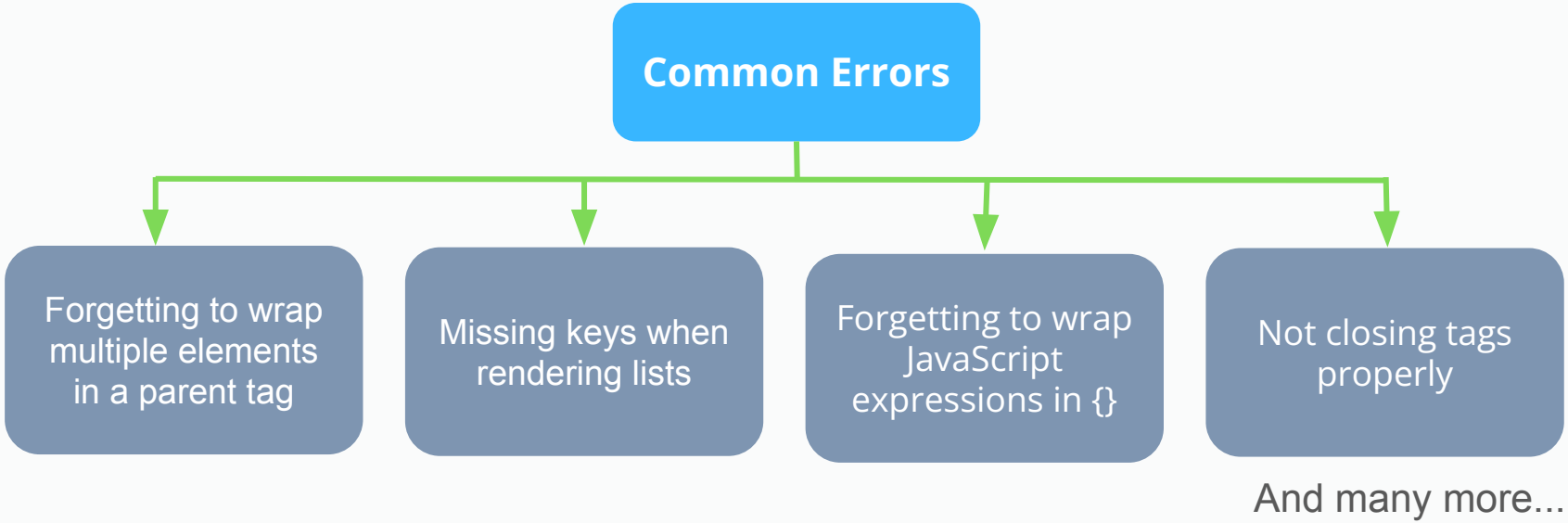


Common Errors in JSX

Errors in JSX

Newton School of Technology

JSX is powerful, but small mistakes can cause errors. Some of the common errors we have listed below:



Forget to wrap elements in parent tag

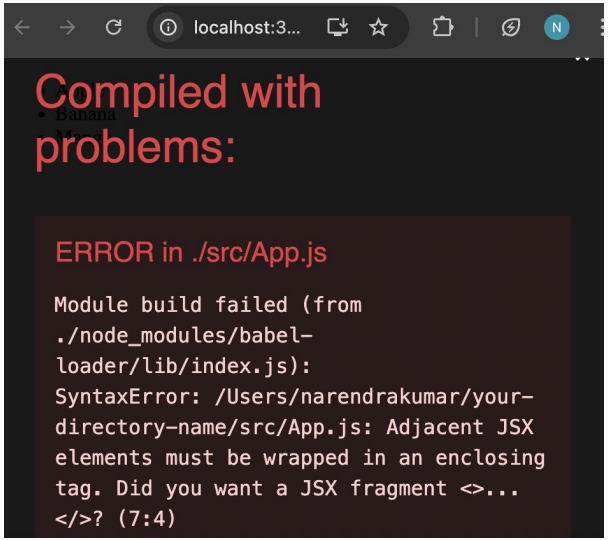
Newton School of Technology

JSX requires a single parent element to wrap all other elements inside it. Without this, JSX will throw an error.

```
1  function App() {
2    return (
3      <h1>Hello</h1>
4      <p>Welcome to JSX!</p>
5    );
6  }
```

No parent element

Will throw an error

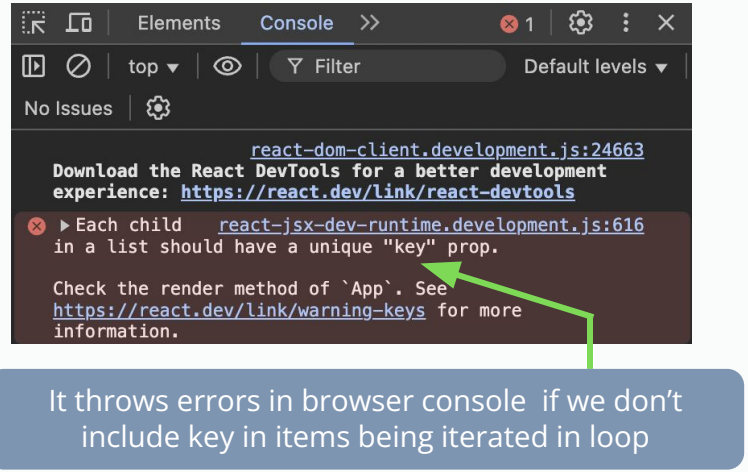


Missing Keys in Lists

Newton School of Technology

React uses keys to efficiently update and re-render lists. Without a key, React may not track items correctly, leading to unwanted re-renders or UI bugs.

```jsx
function App() {
  const fruits = ["Apple", "Banana", "Mango"];
  return (
    <ul>
      {fruits.map((fruit) => {
          return (
            <li>
              {fruit}
            </li>
          );
        }
      )}
    </ul>
  );
}

export default App;
```



It throws errors in browser console if we don't include key in items being iterated in loop

Missing Keys in Lists

Newton School of Technology

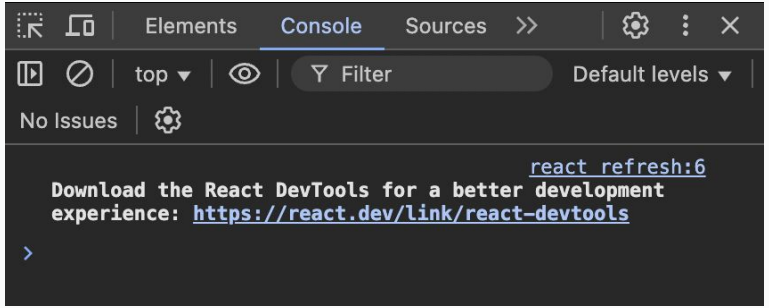
Including keys in an iterable list makes the React rendering process efficient.

```jsx
function App() {
  const fruits = [
    "Apple", "Banana", "Mango"
  ];

  return (
    <ul>
      {fruits.map((fruit, index) => {
        return (
            <li key={index}>{fruit}</li>
        )
      })}
    </ul>
  );
}
export default App;
```

Using keys for efficient re-renders

No errors

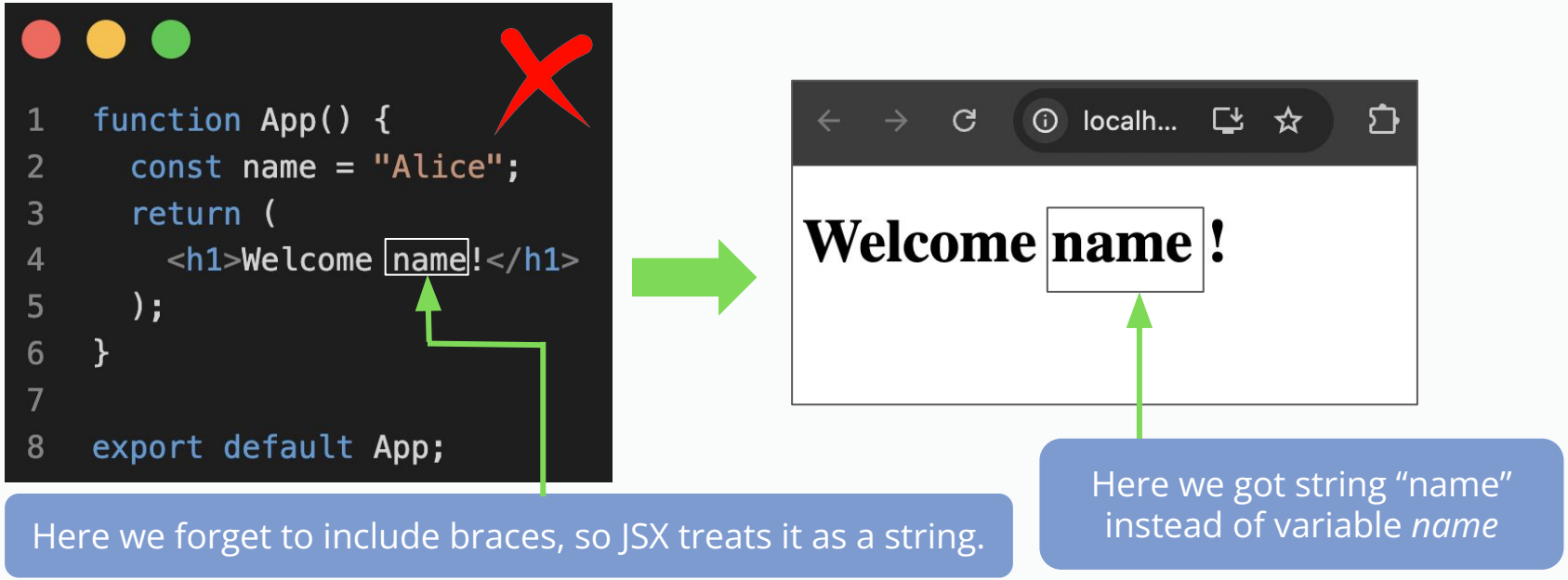
after including key



Forgetting to wrap expressions in {}

Newton School of Technology

Many a time people might want to include a variable, but since they forget to use curly braces, they end up writing a string instead of a variable.



Forgetting to wrap expressions in {}

Newton School of Technology

Many a time people might want to include a variable, but since they forget to use curly braces, they end up writing a string instead of a variable.

```
1  function App() {
2    const name = "Alice";
3    return (
4      <h1>Welcome {name} !</h1>
5    );
6  }
7
8  export default App;
```

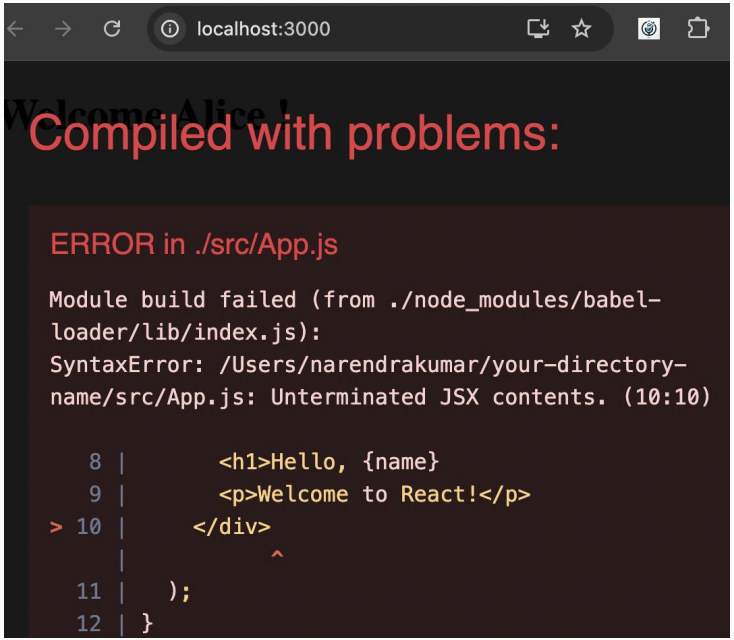
Here we correctly used braces

Not Closing Tags Properly

Many a time people forget closing a tag, which can result in an error.

```
1  import React from "react";
2
3  function App() {
4    const name = "Narendra";
5
6    return (
7      <div>
8        <h1>Hello, {name}
9        <p>Welcome to React!</p>
10     </div>
11   );
12 }
```

Forget to close h1



In-class questions

References

1. **MDN React Guide**: Comprehensive and beginner-friendly documentation for React https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started
2. **Websites and Tutorials:**
   a. 📌 w3schools: https://www.w3schools.com/REACT/DEFAULT.ASP
   b. 📌codecademy: https://www.codecademy.com/learn/react-101
3. **Other Useful Resources**
   a. 📌 React GitHub Repository: https://github.com/facebook/react
   b. 📌 React Patterns & Best Practices: https://reactpatterns.com/

Newton School of Technology

Thanks for watching!