React Day 02

https://react.dev/learn

Contents

React is a popular JavaScript library used for building user interfaces (UIs). It provides a way to efficiently update and render UI components in response to changes in application state. Here are some reasons why React is widely used and its advantages:

Component-Based Architecture:

1. React follows a component-based architecture, which means UIs are built by composing reusable components. Each component encapsulates its own logic, styles, and UI structure, making it easier to develop and maintain complex applications. Components can be reused throughout the application, promoting code reusability and modularity.

Virtual DOM:

2. React utilizes a virtual DOM (Document Object Model) to efficiently update and render UI components. The virtual DOM is a lightweight representation of the actual DOM, allowing React to perform efficient diffing and update only the necessary parts of the UI when the application state changes. This results in improved performance and a smoother user experience.

Unidirectional Data Flow:

1. React follows a unidirectional data flow pattern, also known as one-way data binding. Data flows from parent components to child components, ensuring predictable and easy-to-understand data management. This helps in debugging and maintaining application state, as changes are propagated in a controlled manner.

Reusable and Composable Components:

2. React promotes the creation of reusable and composable components. Components can be easily combined to create more complex UI structures, allowing developers to build UIs using a modular approach. This reusability reduces code duplication and simplifies maintenance.

React Hooks:

3. React Hooks, introduced in React 16.8, provide a way to use state and other React features in functional components. Hooks allow developers to manage component state and perform side effects without the need for class components. They make it easier to write and reason about React code, improving code readability and reducing complexity.

Here's a simple code snippet showcasing the usage of React:

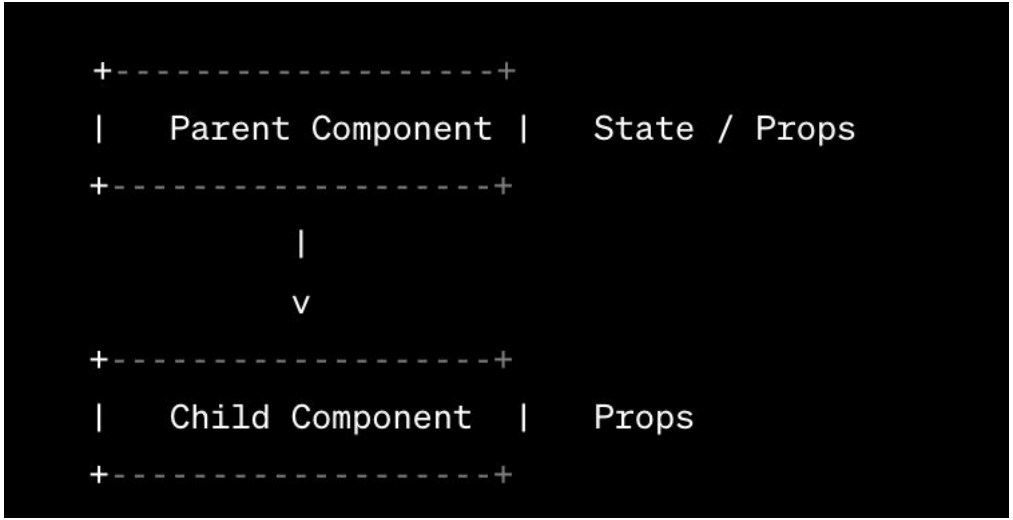
```jsx
import React from 'react';

const App = () => {
  const name = 'John Doe';

  return <h1>Hello, {name}!</h1>;
};

export default App;
```

In the code above, we define a functional component App that renders a simple greeting message. The {name} inside the JSX represents dynamic content, allowing us to render the value of the name variable within the HTML.



Differences from Traditional JavaScript:

React introduces a different approach to building UIs compared to traditional JavaScript frameworks or vanilla JavaScript:

Declarative Syntax:

1. React uses a declarative syntax, where you describe what you want the UI to look like, and React takes care of updating the UI to match the desired state. In contrast, traditional JavaScript often involves manual DOM manipulation and imperative code, where you specify how to update the UI step by step.

Component Reusability:

2. React emphasizes component reusability, allowing developers to create self-contained UI components that can be reused throughout the application. Traditional JavaScript may involve writing functions or methods to handle UI updates, but it may not provide a clear separation of concerns or a systematic way to reuse UI code.

Efficient Updates with Virtual DOM:

3. React's virtual DOM enables efficient updates by calculating and applying the minimum necessary changes to the UI. Traditional JavaScript may involve directly manipulating the DOM, which can be less performant and result in slower UI updates.

Unidirectional Data Flow:

4. React follows a unidirectional data flow, which provides better predictability and simplifies state management. Traditional JavaScript may involve bidirectional data binding, where changes in one part of the application can have unpredictable effects on other parts.

By leveraging these advantages and the ecosystem around React, developers can build scalable, performant, and maintainable UIs for web

JSX (JavaScript XML) is an extension to JavaScript that allows you to write HTML-like code directly within your JavaScript code. It is a syntax extension provided by React to define the structure and appearance of React components.
Here's a simple explanation of JSX with a code snippet and a diagram:

In the code, we define a functional component App. Within the return statement, we write HTML-like code using JSX syntax. We have a element containing an <h1> heading and a <p> paragraph.

In the diagram, we represent the JSX code as a tree-like structure. The outermost element is the element, which contains two child elements: the <h1> heading and the <p> paragraph.
JSX allows you to write HTML-like syntax directly within your JavaScript code. It makes the rendering of

```jsx
import React from 'react';

const App = () => {
  return (
    <div>
      <h1>Hello, JSX!</h1>
      <p>This is a JSX example.</p>
    </div>
  );
};

export default App;
```

To convert an HTML page into a React component, you can follow these steps:

1. Create a new React component file with a .jsx or .js extension, such as MyComponent.jsx.
2. Import React and any other necessary dependencies:

```
import React from 'react';
```

3. Define your React component:

```
const MyComponent = () => {
  return (
    // JSX code representing the HTML structure
    <div>
      <h1>Hello, React Component!</h1>
      <p>This is a converted HTML page.</p>
    </div>
  );
};
```

4. Export your component:

```
export default MyComponent;
```

The above code creates a React component called MyComponent. Within the component, we use JSX syntax to represent the HTML structure. In this example, we have a element containing an <h1> heading and a <p> paragraph.

With this code, you can use the MyComponent React component in your application by importing it and including it within other components or in the ReactDOM.render() method to render it as the root component.

```
import React from 'react';

const MyComponent = () => {
  return (
    <div>
      <h1>Hello, React Component!</h1>
      <p>This is a converted HTML page.</p>
    </div>
  );
};

export default MyComponent;
```

To code Sample React Code online

https://codesandbox.io/s/hacker-dormitory-v1fgb?file=/src/studentsList.js

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>My HTML Page</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <header>
    <h1>My HTML Page</h1>
    <nav>
      <ul>
        <li><a href="#">Home</a></li>
        <li><a href="#">About</a></li>
        <li><a href="#">Contact</a></li>
      </ul>
    </nav>
  </header>

  <section>
    <h2>About Me</h2>
    <p>This is a section about me.</p>
  </section>

  <footer>
    <p>&copy; 2023 My HTML Page. All rights reserved.</p>
  </footer>
</body>
</html>
```

```jsx
import React from 'react';
import './styles.css'; // Import the CSS file

const MyComponent = () => {
  return (
    <div>
      <header>
        <h1>My HTML Page</h1>
        <nav>
          <ul>
            <li><a href="#">Home</a></li>
            <li><a href="#">About</a></li>
            <li><a href="#">Contact</a></li>
          </ul>
        </nav>
      </header>

      <section>
        <h2>About Me</h2>
        <p>This is a section about me.</p>
      </section>

      <footer>
        <p>&copy; 2023 My HTML Page. All rights reserved.</p>
      </footer>
    </div>
  );
};

export default MyComponent;
```

Passing Data through Props

```jsx
import React from 'react';

const ParentComponent = () => {
  const message = "Hello from Parent Component!";

  return <ChildComponent message={message} />;
};

const ChildComponent = (props) => {
  return <h1>{props.message}</h1>;
};

export default ParentComponent;
```

In the code, we have a parent component (ParentComponent) that defines a message variable with the value "Hello from Parent Component!". We then render the child component (ChildComponent) and pass the message value as a prop.
The child component receives the prop (message) through its function parameters (props) and renders the value within an <h1> heading.
When the ParentComponent is rendered, it will display the message "Hello from Parent Component!" through the ChildComponent.
Props allow you to pass data from a parent component to its child component(s). They provide a way to share information and control the behavior of child components. Here are a few key points to remember:

1. Props are passed from the parent component to the child component within JSX by adding attributes to the child component's tag.
2. Props are accessible in the child component through the props object, which is automatically passed as an argument to the child component's function.
3. Props are read-only and should not be modified within the child component. They represent the data passed down from the parent and should be treated as immutable.

By passing data through props, you can create reusable and modular components that can be easily customized and controlled based on the specific needs of the parent component. This enables a flexible and dynamic composition of UI components in React applications.