React

1.1.1 1. Introduction

React.js is a JavaScript library for building user interfaces, particularly single-page applications where data changes over time. Developed by Facebook, React allows developers to create large web applications that can update and render efficiently.

- **Key Features**: Declarative syntax, component-based architecture, unidirectional data flow, virtual DOM.
- **React's Purpose**: To build interactive UIs with ease, allowing developers to design views for each state of their application.

Example: A simple counter app that updates the count whenever a button is clicked.

1.1.2 2. Hello World on React App.js

React apps are usually initiated with npx create-react-app. The App.js file is the main entry point for defining your application's UI.


```
import React from 'react';

function App() {
 return (
  <div>
   <h1>Hello, World!</h1>
  </div>
 );
```

```
}

export default App;
```

Explanation: This example demonstrates the simplest possible React component, displaying "Hello, World!" on the page.

1.1.3 3. Folder Structure on Creating a React App

A typical React project structure:

- **node_modules/**: Contains all dependencies.
- **public/**: Contains static files like index.html.
- **src/**: Contains the application code.
  - **App.js**: Main component file.
  - **index.js**: Entry point for rendering components.
  - **components/**: Folder for reusable components.

Example: Understanding this structure helps in maintaining organized code and scalability.

1.1.4 4. Components

React applications are made up of components, which can be thought of as reusable pieces of UI.

```
function Greeting() {
  return <h1>Hello, React!</h1>;
}
```

Explanation: Components can be functional or class-based, and they help in breaking down the UI into smaller, manageable pieces.

1.1.5 5. Functional Components

Functional components are simple functions that return JSX. They are often used when you don't need to manage state.

```
const Welcome = () => <h1>Welcome to React!</h1>;
```

Explanation: Functional components are more concise and easier to read, especially when dealing with presentational components.

1.1.6 6. Class Components

Class components are ES6 classes that extend `React.Component` and include a `render()` method to return JSX.

```
class Welcome extends React.Component {
  render() {
    return <h1>Welcome to React!</h1>;
```

```
  }
}
```

Explanation: Class components are typically used when you need lifecycle methods or state management.

1.1.7 7. Hooks Update

Hooks allow you to use state and other React features in functional components.

- **Example of useState Hook**:

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

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
}
```

Explanation: Hooks make it easier to use state and other React features without writing a class.

1.1.8 **8. JSX**

JSX (JavaScript XML) is a syntax extension that allows writing HTML elements in JavaScript.

```
const element = <h1>Hello, JSX!</h1>;
```

Explanation: JSX makes it easier to write and add HTML in React, but it needs to be compiled by Babel before it can be interpreted by browsers.

1.1.9 **9. Props**

Props (short for "properties") allow you to pass data from one component to another.

```
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}

// Usage
<Greeting name="Alice" />
```

Explanation: Props are read-only, making them ideal for passing immutable data to components.

1.1.10 **10. State**

State is a built-in object used to store property values that belong to the component.

```
class Counter extends React.Component {
  constructor() {
    super();
    this.state = { count: 0 };
  }

  render() {
    return <h1>Count: {this.state.count}</h1>;
  }
}
```

Explanation: State is mutable and is typically managed within the component where it is defined.

1.1.11 **11. setState**

setState is the method used to update the component's state.

```
this.setState({ count: this.state.count + 1 });
```

Explanation: React automatically re-renders the component when state changes, keeping the UI in sync with the data.

1.1.12 **12. Destructuring Props and State**

Destructuring allows for cleaner and more concise code when working with props and state.

```
function Greeting({ name }) {
  return <h1>Hello, {name}!</h1>;
}
```

Explanation: Destructuring improves code readability by directly extracting the necessary values.

1.1.13 **13. Event Handling**

React handles events using camelCase syntax and prevents the default behavior using preventDefault().

```
function Button() {
  const handleClick = (e) => {
    e.preventDefault();
    alert('Button clicked!');
```

```
  };

  return <button onClick={handleClick}>Click me</button>;
}
```

Explanation: Event handlers in React are passed as functions rather than strings.

1.1.14 **14. Binding Event Handlers**

In class components, event handlers need to be bound to the class instance.

```
class Button extends React.Component {
  constructor() {
    super();
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    alert('Button clicked!');
  }

  render() {
    return <button onClick={this.handleClick}>Click me</button>;
  }
```

```
}
```

Explanation: Binding is necessary to ensure the this keyword refers to the correct instance within the handler.

1.1.15 **15. Methods as Props**

You can pass methods as props to child components.

```
function Parent() {
  const showMessage = () => {
    alert('Hello from Parent!');
  };

  return <Child onClick={showMessage} />;
}

function Child({ onClick }) {
  return <button onClick={onClick}>Click me</button>;
}
```

Explanation: Passing methods as props allows child components to interact with parent components.

1.1.16 **16. Conditional Rendering**

Render components conditionally using if-else or ternary operators.

```
function Greeting({ isLoggedIn }) {
  if (isLoggedIn) {
    return <h1>Welcome back!</h1>;
  }
  return <h1>Please sign up.</h1>;
}
```

Explanation: Conditional rendering enables the display of different UI elements based on the application's state.

1.1.17 **17. List Rendering**

Rendering lists in React can be done using the map() function.

```
function NameList() {
  const names = ['Alice', 'Bob', 'Charlie'];
  return (
    <ul>
      {names.map((name) => (
        <li key={name}>{name}</li>
```

```
    ))}
   </ul>
  );
}
```

Explanation: The map() function iterates over an array, returning JSX for each item.

1.1.18 **18. Lists and Keys**

Keys help React identify which items have changed, been added, or removed.

```
function NameList() {
  const names = ['Alice', 'Bob', 'Charlie'];
  return (
    <ul>
      {names.map((name, index) => (
        <li key={index}>{name}</li>
      ))}
    </ul>
  );
}
```

Explanation: Keys should be unique and consistent among siblings to optimize rendering.

1.1.19 19. Index as Key Anti-pattern

Using the index of an array as a key can lead to problems if the list is re-ordered or items are added/removed.

```
const listItems = list.map((item, index) =>
  <li key={index}>{item}</li>
);
```

Explanation: Use unique identifiers from your data as keys whenever possible.

1.1.20 20. Styling and CSS Basics

In React, you can style components using inline styles, CSS files, or CSS-in-JS libraries.

```
const divStyle = {
  color: 'blue',
  backgroundColor: 'lightgray',
};

function StyledComponent() {
  return <div style={divStyle}>This is a styled div!</div>;
}
```

Explanation: Inline styles are defined as objects with camelCased properties.

1.1.21 **21. Basics of Form Handling**

Form handling in React requires managing form data using state.

```
class MyForm extends React.Component {
  constructor() {
    super();
    this.state = { value: '' };

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({ value: event.target.value });
  }

  handleSubmit(event) {
    alert('A name was submitted: ' + this.state.value);
    event.preventDefault();
  }

  render() {
```

```
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text" value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

Explanation: Forms in React are controlled components where the input values are managed by the component's state.