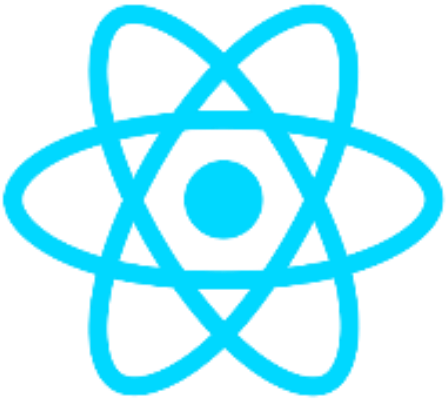

React Js

Interview Questions and Answers

Aswanth Alakkadan

1. What is React?

React is an open-source JavaScript library developed by Facebook for building user interfaces or UI components. It is commonly used for building single-page applications where fast and interactive user experiences are essential.

2. What are the major features of React?

Major features of React include components for building UI, a virtual DOM for efficient updates, JSX for writing UI components in a syntax similar to HTML, unidirectional data flow, React Router for navigation, and the ability to manage state and props.

3. What is JSX?

JSX (JavaScript XML) is a syntax extension for JavaScript used with React. It allows developers to write HTML-like code within JavaScript, making it easier to describe the structure of UI components. JSX is later transpiled into regular JavaScript by tools like Babel.

4. What is the difference between Element and Component?

In React, an element is a plain JavaScript object that represents a DOM element or a component. An element is the smallest building block in React, while a component is a more advanced and reusable piece of code composed of one or more elements.

5. How to create components in React?

Components in React can be created using either class components or function components. Class components are ES6 classes that extend from `React.Component`, and function components are simple JavaScript functions. Both types can return JSX to describe what should be rendered.

6. When to use a Class Component over a Function Component?

Class components were traditionally used for more advanced features like state management and lifecycle methods. However, with the introduction of React Hooks, function components can now also manage state and use lifecycle methods, making them more popular. As of React 16.8, functional components with hooks are the recommended way to create components.

7. What are Pure Components?

Pure Components in React are a type of class component that automatically implements the `shouldComponentUpdate` method with a shallow prop and state comparison. This optimization can improve performance by preventing unnecessary renders when props or state haven't changed.

8. What is state in React?

State in React is a JavaScript object that represents the internal data of a component. It can be changed over time in response to user actions or other events. Components can have local state, managed by the component itself, and it is used for dynamic and interactive UIs.

9. What are props in React?

Props (short for properties) are a mechanism for passing data from a parent component to a child component in React. Props are immutable and help in making components more flexible and reusable by allowing them to receive data from outside sources.

10. What is the difference between state and props?

- **State:** Represents the internal data of a component. It is mutable and can be changed by the component itself. Used for dynamic and interactive UIs.

- **Props:** Short for properties, props are immutable and passed from a parent component to a child component. They are used to configure a component and make it customizable and reusable.

11. Why should we not update the state directly?

Directly updating the state in React is discouraged because React relies on the concept of virtual DOM and its ability to efficiently determine the difference between the current state and the new state to update the actual DOM. If you update the state directly, React might not be aware of the change, and it could lead to unexpected behavior. Instead, use the `setState` method, which ensures that React is notified of the state change and can perform the necessary updates.

12. What is the purpose of a callback function as an argument of ѕetState()?

The callback function passed as an argument to `setState` is executed after the state is successfully updated. This is useful when you need to perform some action or execute code that relies on the updated state. It ensures that the code inside the callback runs after the state has been applied.

13. What is the difference between HTML and React event handling?

In React, event handling is similar to HTML, but there are some syntactical differences. In React, event names are camelCase (e.g., `onClick` instead of `onclick`). Additionally, in React, you pass a function as an event handler, whereas in HTML, you provide a string. React uses synthetic events to normalize the differences between browser implementations.

14. How to bind methods or event handlers in JSX callbacks?

There are a few ways to bind methods or event handlers in JSX callbacks:

- Use arrow functions in the callback directly: `onClick={() => this.handleClick()}`

- Bind the method in the constructor: `constructor() { this.handleClick = this.handleClick.bind(this); }`

- Use class properties syntax: `handleClick = () => { /* method code */ }`

15. How to pass a parameter to an event handler or callback?

To pass parameters to an event handler in React, you can use arrow functions or the `bind` method. For example:

- Arrow function: `onClick={() => this.handleClick(param)}`

- Binding in the constructor `onClick={this.handleClick.bind(this, param)}`

16. What are synthetic events in React?

Synthetic events in React are wrappers around the native browser events, providing a consistent interface across different browsers. React creates synthetic events to ensure that the event handling behavior is the same across various environments. This abstraction helps in normalizing the differences in event handling between browsers.

17. What are inline conditional expressions?

Inline conditional expressions in React allow you to conditionally render content based on a condition. You can use the ternary operator or logical `&&` to achieve this. For example:

```jsx
{condition ? <TrueComponent /> : <FalseComponent />}
```

18. What is the "key" prop, and what is the benefit of using it in arrays of elements?

The "key" prop is a special attribute used in React when rendering arrays of elements. It helps React identify which items have changed, been added, or been removed. Using keys improves the efficiency of rendering and updating components, especially in the context of dynamic lists. Keys should be unique among siblings, but they don't need to be globally unique.

19. What is the use of refs?

Refs in React provide a way to access and interact with the DOM directly or to get a reference to a React component. They are useful in scenarios like accessing input values, triggering imperative animations, or integrating with third-party DOM libraries that don't work well with React's declarative approach.

20. How to create refs?

Refs can be created using the `React.createRef()` method. For example:

```jsx
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.myRef = React.createRef();
  }

  render() {
    return <div ref={this.myRef}>Hello, World!</div>;
  }
}
```

After creating a ref, you can access the DOM node or React component it refers to using `this.myRef.current`.

21. What are forward refs?

Forwarding refs is a technique in React that allows a component to pass its `ref` property to a child component. This enables the parent component to interact with the child's DOM node or React component. The `React.forwardRef` function is typically used to create components that forward refs.

22. Which is the preferred option within callback refs and findDOMNode()?

Using callback refs is generally preferred over `findDOMNode()`. Callback refs provide a more flexible and explicit way to get a reference to a DOM node or React component. `findDOMNode()` is considered legacy, and using it can make it harder to follow the data flow in your application. Callback refs are more in line with React's modern ref handling.

23. Why are String Refs legacy?

String refs (using `ref="myRef"`) were the original way to create refs in React. However, they are considered legacy, and the use of callback refs (functions) or `React.createRef()` is now

recommended. String refs have some drawbacks, such as not being a reliable way to access refs in function components and being prone to namespace collisions.

24. What is Virtual DOM?

The Virtual DOM (Document Object Model) is a programming concept used in React to improve the performance of updating the user interface. It is a lightweight copy of the actual DOM, maintained by React. When the state of a component changes, React first updates the Virtual DOM, compares it with the previous state, and then efficiently updates only the parts of the real DOM that have changed.

25. How does the Virtual DOM work?

When a component's state changes in React, a new Virtual DOM tree is created. React then compares this new tree with the previous one to identify the differences (diffing). Once the differences are identified, React calculates the most efficient way to update the real DOM to reflect these changes. This process of updating the real DOM is called reconciliation.

26. What is the difference between Shadow DOM and Virtual DOM?

- Virtual DOM: A concept used in React to optimize the updating of the actual DOM by first updating a lightweight copy (virtual DOM) and then efficiently applying the changes to the real DOM.

- Shadow DOM: A web standard that allows encapsulation of styling and structure in a web component, providing a way to create self-contained components with their own encapsulated DOM.

27. What is React Fiber?

React Fiber is a complete rewrite of the core algorithm that React uses to reconcile the Virtual DOM and update the UI. It was introduced to improve the performance and enable new features like asynchronous rendering and better support for concurrent updates.

28. What is the main goal of React Fiber?

The main goal of React Fiber is to improve the responsiveness and performance of React applications by enabling asynchronous rendering and better handling of concurrent updates. It allows React to break down the work into smaller units and prioritize rendering based on the priority of each unit.

29. What are controlled components?

Controlled components in React are components whose state is controlled by React. The state of a controlled component is typically handled by the parent component, and the child component

receives its state and updates through props. This ensures that the parent has full control over the child component's behavior.

30. What are uncontrolled components?

Uncontrolled components in React are components that manage their own state internally. Instead of being controlled by the parent component through props, an uncontrolled component directly manages its own state. This is often done using refs to access and modify the DOM directly. Uncontrolled components are less common than controlled components in React applications.

31. What is the difference between `createElement` and `cloneElement`?

- `createElement`: It is a method used to create and return a new React element. It takes three arguments: the type of the element (string or React component), an optional set of properties (props) for the element, and the element's children.

- `cloneElement`: It is a method used to clone and return a React element with a new set of properties. It takes an element to clone as the first argument and an optional set of new properties as the second argument. It is often used to extend or override the properties of an existing element.

32. What is Lifting State Up in React?

Lifting State Up is a pattern in React where the state is moved from a child component to its parent component. This is done to share state between components that need access to the same data. By lifting the state up to a common ancestor, you avoid prop-drilling (passing props through multiple layers of components) and make the state more accessible to the components that need it.

33. What are the different phases of component lifecycle?

The component lifecycle in React has three main phases:

1. Mounting: The phase where a component is being created and inserted into the DOM.

2. Updating: The phase where a component is being re-rendered as a result of changes in either its props or state.

3. Unmounting: The phase where a component is being removed from the DOM.

34. What are the lifecycle methods of React?

- Mounting Phase:

- `constructor()`

- `static getDerivedStateFromProps()`

- `render()`

- `componentDidMount()`

- Updating Phase:

- `static getDerivedStateFromProps()`

- `shouldComponentUpdate()`

- `render()`

- `getSnapshotBeforeUpdate()`

- `componentDidUpdate()`

- Unmounting Phase:

- `componentWillUnmount()`

35. What are Higher-Order components?

Higher-Order Components (HOCs) are functions that take a component and return a new component with enhanced functionality. HOCs allow you to reuse component logic, share code between components, and modify the behavior of components. They are a powerful pattern in React for code reuse.

36. How to create a props proxy for an HOC component?

To create a props proxy for an HOC component, you typically pass the original component as an argument to the HOC function and return a new component. Here's a simplified example:

```jsx
const myHigherOrderComponent = (WrappedComponent) => {
  return class extends React.Component {
    render() {
      return <WrappedComponent {...this.props} extraProp="additional data" />;
    }
  };
};
```

37. What is context?

Context in React is a way to pass data through the component tree without having to pass props manually at every level. It is often used when data needs to be accessible to many components at different levels of the component tree.

38. What is the children prop?

The `children` prop is a special prop in React that represents the content between the opening and closing tags of a component. It is used to pass elements, components, or text content as children to a component. You can access and manipulate the `children` prop within the component to modify or render content.

39. How to write comments in React?

In JSX, comments are written inside curly braces `{/* */}`. For example:

```jsx
<div>
  {/* This is a comment */}
  <p>Hello, World!</p>
</div>
```

40. What is the purpose of using the super constructor with a props argument?

When you define a constructor in a React component that extends `React.Component`, you need to call `super(props)` within the constructor. This is necessary to ensure that the component class properly inherits from `React.Component` and sets up the initial state and other internal mechanisms. It is a requirement in ES6 classes when defining a constructor in a class that extends another class.

41. What is reconciliation?

Reconciliation is the process by which React updates the DOM to match the virtual DOM after a component's state or props change. React's diffing algorithm identifies the differences between the previous and current state or props, and then efficiently updates only the parts of the DOM that have changed.

42. How to set state with a dynamic key name?

You can set state with a dynamic key name using the computed property name syntax introduced in ECMAScript 2015 (ES6). Here's an example:

```jsx
handleInputChange = (key, value) => {
  this.setState({ [key]: value });
};

// Usage
this.handleInputChange("dynamicKey", "dynamicValue");
```

43. What would be the common mistake of a function being called every time the component renders?

One common mistake is not memoizing or properly handling functions passed as props. If a new function instance is created on every render, it can cause unnecessary re-renders of child components, affecting performance. To avoid this, use callback functions or memoization techniques.

44. Is the lazy function supports named exports?

As of my last knowledge update in January 2022, the `lazy` function in React primarily supports default exports. However, support for named exports was an open issue in the React repository. Please check the latest React documentation or release notes for any updates or changes.

45. Why does React use `className` over the `class` attribute?

React uses `className` instead of the traditional `class` attribute to define HTML class names. This is because `class` is a reserved keyword in JavaScript, and using it in JSX would lead to a syntax error. Therefore, React uses `className` to set the CSS class of an element.

46. What are fragments?

Fragments are a feature in React that allows you to group multiple children elements without introducing an additional parent wrapper element in the DOM. Fragments do not create extra nodes in the DOM hierarchy. They provide a cleaner way to structure your JSX when you need to return multiple elements without a common parent.

47. Why are fragments better than container divs?

Fragments are preferred over container divs when you want to group multiple elements without introducing an extra DOM node. Using container divs can lead to unnecessary and sometimes semantically incorrect markup. Fragments avoid this issue by allowing you to group elements without adding any extra nodes to the DOM.

48. What are portals in React?

Portals in React provide a way to render children components outside their parent hierarchy, typically at the top level of the DOM or in a different container. This can be useful for rendering components like modals that need to appear above all other elements but remain logically related to the rest of the application.

49. What are stateless components?

Stateless components, also known as functional components, are components in React that are defined as plain JavaScript functions. They do not have internal state (using the `useState` hook) and do not have lifecycle methods. Stateless components are mainly used for presenting UI based on the input props they receive.

50. What are stateful components?

Stateful components, also known as class components or components using hooks, are components in React that can have and manage their own internal state using the `state` property or `useState` hook. Stateful components can also implement lifecycle methods and handle user interactions, making them suitable for managing dynamic and interactive parts of the UI.

51. How to apply validation on props in React?

You can apply validation on props in React by using PropTypes. PropTypes is a type-checking library that helps you define the types of props a component should receive. Here's an example:

```jsx
import PropTypes from 'prop-types';


MyComponent.propTypes = {
  name: PropTypes.string.isRequired,
  age: PropTypes.number,
  isStudent: PropTypes.bool.isRequired,
};
```

52. What are the advantages of React?

- **Declarative Syntax:** React uses a declarative syntax that makes it easier to understand and reason about the code.

- **Component-Based:** The component-based architecture promotes reusability and maintainability.

- **Virtual DOM:** React's virtual DOM enhances performance by efficiently updating the actual DOM.

- **One-Way Data Binding:** React follows a unidirectional data flow, making it easier to manage and understand state changes.

- **Ecosystem:** React has a vast ecosystem with a strong community, supporting libraries, and tools.

53. What are the limitations of React?

- **Steep Learning Curve:** React can have a steep learning curve, especially for beginners.

- **JSX Complexity:** JSX syntax might be challenging for developers coming from HTML backgrounds.

- **Overhead of Virtual DOM**: While the virtual DOM improves performance, it introduces a certain amount of overhead.

- **Tooling:** The tooling around React can be overwhelming, and choosing the right set of tools can be challenging.

54. What are error boundaries in React v16?

Error boundaries are a feature in React v16 and later that allow components to catch JavaScript errors anywhere in their component tree, log those errors, and display a fallback UI instead of crashing the entire application. To use error boundaries, you define a special method called `componentDidCatch` in your component.

55. How are error boundaries handled in React v15?

In React v15 and earlier, error boundaries were not supported. Errors in components could lead to the entire application crashing. React v16 introduced error boundaries to address this issue and provide a way to gracefully handle errors within components.

56. What are the recommended ways for static type checking?

For static type checking in React, you can use TypeScript or Flow. TypeScript is a superset of JavaScript that adds static typing, and Flow is a static type checker developed by Facebook. Both tools help catch type-related errors during development and provide better tooling support.

57. What is the use of the react-dom package?

The `react-dom` package is responsible for rendering React components in the DOM (Document Object Model). It provides methods like `render` for mounting React components into the browser, `hydrate` for server-side rendering, and other utilities for interacting with the DOM.

58. What is the purpose of the render method of react-dom?

The `render` method in `react-dom` is used to render a React element into the DOM in a specified container. It takes two arguments: the element to render and the container DOM element where the rendering should take place. For example:

```jsx
import React from 'react';
import ReactDOM from 'react-dom';

const element = <h1>Hello, World!</h1>;
ReactDOM.render(element, document.getElementById('root'));
```

59. What is ReactDOMServer?

`ReactDOMServer` is a package in React that provides server-side rendering APIs. It allows you to render React components to static HTML on the server, making it possible to send pre-rendered HTML to the client, which can improve initial page load performance.

60. How to use InnerHTML in React?

In React, the use of `innerHTML` is discouraged due to potential security vulnerabilities (e.g., Cross-Site Scripting). If you need to render HTML content, you can use the `dangerouslySetInnerHTML` prop. However, it should be used with caution, and you must ensure that the HTML content is sanitized to prevent security issues.

61. How to use styles in React?

Styles in React can be applied using inline styles or by importing external stylesheets. For inline styles, you can use the `style` attribute with a JavaScript object. For external styles, you can import CSS files using `import` statements.

62. How are events different in React?

In React, events are named using camelCase (e.g., `onClick` instead of `onclick`). Additionally, React uses a synthetic event system that wraps the native browser events, providing a consistent interface

across different browsers. React events also have differences in behavior, such as event delegation and automatic binding.

63. What will happen if you use ѐsetStatè in the constructor?

Using `setState` in the constructor is not recommended because it can lead to unexpected behavior. The `constructor` is called only once during the component's lifecycle, and directly calling `setState` in the constructor can cause unnecessary re-renders. Instead, you should initialize the state directly in the constructor without using `setState`, or use class properties for state initialization.

64. What is the impact of indexes as keys?

Using indexes as keys in React can lead to issues with component reordering and reconciliation. React uses keys to track the identity of components in a list, and using indexes can cause problems when the list order changes. It's generally recommended to use stable and unique identifiers as keys to ensure proper component behavior during updates.

65. Is it good to use ѐsetState()`in the ѐomponentWillMount()`method?

Using `setState` in the `componentWillMount` method is not recommended because it can lead to potential bugs and unexpected behavior. The `componentWillMount` lifecycle method is deprecated, and React might remove it in the future. It's better to use `componentDidMount` for data fetching or side effects that involve state updates.

66. What will happen if you use props in the initial state?

Using props to initialize state directly in the constructor can lead to bugs, as the initial state will not be updated if the props change later. Instead, if you need to derive state from props, you should use the `getDerivedStateFromProps` lifecycle method or update the state in the `componentDidUpdate` method.

67. How do you conditionally render components?

Components in React can be conditionally rendered using conditional statements or ternary operators within the JSX. For example:

```jsx
{condition ? <ComponentA /> : <ComponentB />}
```

68. Why do we need to be careful when spreading props on DOM elements?

When spreading props on DOM elements in React, it's essential to be careful about passing non-standard HTML attributes. React doesn't warn or validate custom attributes, and these attributes might not be recognized by the browser or may have unexpected behavior. It's advisable to use standard HTML attributes and pass custom data through `data-*` attributes.

69. How do you use decorators in React?

Decorators are a feature in JavaScript that can be used with React classes. You can decorate your class components, which is the same as passing the component into a function. Decorators are flexible and readable way of modifying component functionality. To use decorators, you need to enable the "decorators" proposal in your build setup (e.g., Babel). Decorators can be applied to class declarations or class methods. For example:

```jsx
@myDecorator
class MyComponent extends React
```

70. How do you memoize a component?

Memoization in React refers to the optimization technique of preventing unnecessary renders for functional components. You can use the `React.memo` higher-order component to memoize a functional component. Here's an example:

```jsx
import React from 'react';

const MemoizedComponent = React.memo((props) => {
  // component logic
});

export default MemoizedComponent;
```

71. How do you implement Server-Side Rendering or SSR?

Server-Side Rendering (SSR) in React involves rendering React components on the server before sending the HTML to the client. Key steps include configuring a server (e.g., Node.js with Express), using a rendering library (e.g., ReactDOMServer), and setting up routes to handle SSR. Libraries like Next.js simplify the process by providing SSR out of the box.

72. How to enable production mode in React?

To enable production mode in React, you typically set the `NODE_ENV` environment variable to `'production'` before building your application. For example:

```bash
NODE_ENV=production npm start
```

73. What is CRA and its benefits?

CRA stands for Create React App. It is a toolchain provided by the React team to set up a new React project quickly with zero build configuration. Benefits of CRA include easy project setup, pre-configured build tools, optimized production builds, and a smooth development experience.

74. What is the lifecycle methods order in mounting?

The order of lifecycle methods during mounting in React is as follows:

1. `constructor`
2. `static getDerivedStateFromProps`
3. `render`
4. `componentDidMount`

75. What are the lifecycle methods going to be deprecated in React v16?

In React v16, the `componentWillMount`, `componentWillReceiveProps`, and `componentWillUpdate` lifecycle methods are considered legacy and may be deprecated in the future. Developers are encouraged to use `componentDidMount`, `componentDidUpdate`, and `getDerivedStateFromProps` instead.

76. What is the purpose of `getDerivedStateFromProps()` lifecycle method?

`getDerivedStateFromProps` is a static method in React used for derived state. It is called before every render, providing an opportunity to update the component's state based on changes in props. It returns an object to update the state or `null` to indicate no state update is necessary.

77. What is the purpose of `getSnapshotBeforeUpdate()` lifecycle method?

`getSnapshotBeforeUpdate` is a lifecycle method called before the most recently rendered output is committed to the DOM. It receives the previous props and state, and it allows you to capture

information about the DOM before it potentially changes. The value returned from this method will be passed as a third parameter to `componentDidUpdate`.

78. Do Hooks replace render props and higher-order components?

Yes, Hooks provide an alternative to render props and higher-order components for sharing stateful logic in functional components. Hooks, such as `useState` and `useEffect`, allow functional components to manage state and side effects, reducing the need for render props and higher-order components in many cases.

79. What is the recommended way for naming components?

The recommended convention for naming components in React is to use PascalCase (start with a capital letter). For example, `MyComponent` or `UserProfile`. This helps distinguish components from regular HTML elements and makes the code more readable.

80. What is the recommended ordering of methods in a component class?

The recommended ordering of methods in a React component class typically follows this order:

1. `static` methods (like `propTypes` or `defaultProps`)
2. `constructor`
3. Lifecycle methods (e.g., `render`, `componentDidMount`)
4. Event handlers
5. Other methods

81. What is a switching component?

A switching component is a component that conditionally renders different content based on a specified condition or state. This can be achieved using conditional statements, such as `if` or the ternary operator, to determine which content to display.

82. Why do we need to pass a function to `setState()`?

In React, `setState` is asynchronous, and it takes a function as an argument to ensure that the state update is based on the current state. This is important because multiple calls to `setState` may be batched for performance reasons, and using a function guarantees the correct order of state updates.

84. What are React Mixins?

React Mixins are a way to reuse component logic in multiple components by providing a way to share code between components. However, mixins are considered an anti-pattern in React, and their use has been discouraged. Instead, other patterns like higher-order components and custom hooks are recommended for code reuse.

85. Why is `isMounted()` an anti-pattern and what is the proper solution?

`isMounted()` is considered an anti-pattern because it can lead to race conditions and unreliable checks for component mount status. A proper solution is to use the `componentWillUnmount` lifecycle method to set a flag indicating whether the component is mounted. Modern React patterns, such as using hooks, often eliminate the need for such checks.

86. What are the Pointer Events supported in React?

React supports pointer events like `onClick`, `onMouseDown`, `onMouseUp`, etc., for handling user interactions. These events provide a unified way to handle both touch and mouse input, making it easier to create cross-platform applications.

87. Why should component names start with a capital letter?

Component names in React should start with a capital letter to distinguish them from regular HTML elements and make it clear that they are custom components. This follows the convention of PascalCase for naming components and helps improve code readability.

88. Are custom DOM attributes supported in React v16?

Yes, custom DOM attributes are supported in React v16. React allows you to pass custom attributes to HTML elements using the `data-*` convention. These attributes are prefixed with

"data-" and can be accessed in the component using the `props` object.

89. What is the difference between `constructor` and `getInitialState`?

In modern React, there is no `getInitialState` method. Instead, the `constructor` is used to initialize the component's state. If you need to set an initial state in a class component, you do it inside the constructor using `this.state = { /* initial state */ };`.

90. Can you force a component to re-render without calling `setState`?

Yes, you can force a component to re-render without calling `setState` by using the `forceUpdate` method. However, using `forceUpdate` is discouraged in most cases, and it's recommended to

manage component state and triggering updates through `setState` for a more controlled and predictable behavior.

91. What is the difference between `super()` and `super(props)` in React using ES6 classes?

In a React component's constructor, `super()` and `super(props)` both call the constructor of the parent class (usually `React.Component`). The difference lies in whether the `props` are passed to the parent class's constructor.

- **`super()`:** Calls the constructor of the parent class without passing any arguments. This is used when the parent class's constructor does not rely on the `props` parameter.

- **`super(props)`:** Calls the constructor of the parent class and passes the `props` as an argument. This is used when the parent class's constructor expects to receive `props`.

Example:

```jsx
class MyComponent extends React.Component {
  constructor(props) {
    // Using super() or super(props) depends on the parent class's constructor
    super(props);
    // Component-specific initialization
  }
}
```

92. How to loop inside JSX?

You can use the `map` function to loop inside JSX. Here's an example:

```jsx
const numbers = [1, 2, 3, 4, 5];

const listItems = numbers.map((number) =>
  <li key={number.toString()}>{number}</li>
```

```
);

return (
  <ul>{listItems}</ul>
);
```

93. How do you access props in attribute quotes?

You can access props in attribute quotes by using curly braces `{}`. Here's an example:

```jsx
const MyComponent = (props) => {
  return (
    <div className={props.isActive ? 'active' : 'inactive'}>
      Content
    </div>
  );
};
```

94. What is React PropType array with shape?

The `PropTypes.arrayOf` and `PropTypes.shape` can be combined to define an array of objects with a specific shape. Here's an example:

```jsx
import PropTypes from 'prop-types';

const MyComponent = ({ items }) => {
  // Component logic
};

MyComponent.propTypes = {
  items: PropTypes.arrayOf(PropTypes.shape({
```

```
    name: PropTypes.string.isRequired,
    age: PropTypes.number.isRequired,
  })).isRequired,
};
```

95. How to conditionally apply class attributes?

You can conditionally apply class attributes using the ternary operator or logical AND (`&&`) in JSX. Here are examples of both approaches:

```jsx
// Using ternary operator
<div className={isError ? 'error' : 'normal'}>Content</div>

// Using logical AND
<div className={isError && 'error'}>Content</div>
```

96. What is the difference between React and ReactDOM?

- **React:** The core library for building user interfaces in React. It provides the `React` and `Component` objects and deals with the components, props, and state.

- **ReactDOM:** A separate package that provides DOM-specific methods for interacting with the browser's DOM. It includes methods like `ReactDOM.render` for rendering React components into the DOM.

97. Why ReactDOM is separated from React?

Separating `ReactDOM` from `React` allows React to be more flexible and agnostic about the rendering environment. It enables React to support different platforms and render targets, not just the browser's DOM. `ReactDOM` is specific to web rendering, while other packages like `ReactNative` cater to different rendering environments.

98. How to use React label element?

You can use the `<label>` element in React to associate text with form controls, providing a better user experience and accessibility. Here's an example:

```jsx
const MyForm = () => {
 return (
  <label htmlFor="username">
   Username:
   <input type="text" id="username" name="username" />
  </label>
 );
};
```

99. How to combine multiple inline style objects?

You can combine multiple inline style objects in React using the spread operator (`...`). Here's an example:

```jsx
const MyComponent = () => {
 const style1 = { color: 'red' };
 const style2 = { fontSize: '16px' };

 const combinedStyles = { ...style1, ...style2 };

 return <div style={combinedStyles}>Content</div>;
};
```

100. How to re-render the view when the browser is resized?

You can use the `resize` event and the `useState` hook to trigger a re-render when the browser is resized. Here's an example:

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

const ResizeAwareComponent = () => {
```

```
    const [windowSize, setWindowSize] = useState({ width: window.innerWidth, height: window.innerHeight });

    useEffect(() => {
      const handleResize = () => {
        setWindowSize({ width: window.innerWidth, height: window.innerHeight });
      };

      window.addEventListener('resize', handleResize);

      return () => {
        window.removeEventListener('resize', handleResize);
      };
    }, []);

    return (
      <div>
        Window Width: {windowSize.width}, Height: {windowSize.height}
      </div>
    );
  };
```

101. What is the difference between ŝetStatè and ŕeplaceStatè methods?

- **`setState`:** Updates the component's state by merging the new state with the current state. It is asynchronous and schedules a re-render of the component.

- **`replaceState`:** Replaces the entire state of the component with the provided state. It is also asynchronous and schedules a re-render.

In modern React, `replaceState` is not commonly used, and `setState` is preferred for updating the state.

102. How to listen to state changes?

You can use the `useEffect` hook to listen to state changes. Inside the `useEffect` callback, you can perform actions whenever the specified state variables change. Here's an example:

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

const MyComponent = () => {
  const [count, setCount] = useState(0);

  useEffect(() => {
    // This function will be called whenever 'count' changes
    console.log('State changed:', count);
  }, [count]);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
};
```

103. What is the recommended approach for removing an array element in React state?

The recommended approach for removing an array element from React state is to use the `filter` method to create a new array without the element to be removed. Here's an example:

```jsx
const removeItem = (indexToRemove) => {
  setArray((prevArray) => prevArray.filter((_, index) => index !== indexToRemove));
```

```
};
```

In this example, `setArray` is a function that updates the state, and `filter` is used to create a new array that excludes the element at the specified index.

104. Is it possible to use React without rendering HTML?

No, React is specifically designed for building user interfaces, and its core functionality revolves around rendering components to the DOM (Document Object Model). While React can be used in various environments, such as server-side rendering or mobile app development, it always involves rendering UI components, which ultimately generate HTML.

105. How to pretty print JSON with React?

To pretty print JSON in React, you can use the `JSON.stringify` method with the third parameter as the number of spaces to use for indentation. Here's an example:

```jsx
const MyComponent = () => {
  const data = { name: 'John', age: 25 };

  return (
    <pre>
      {JSON.stringify(data, null, 2)}
    </pre>
  );
};
```

The `2` in `JSON.stringify(data, null, 2)` specifies that the output should be indented with two spaces.

106. Why you can't update props in React?

In React, props are passed from parent components to child components, and they are meant to be immutable. The idea is that a parent component passes data down to its child components, and those child components should not modify the received props directly.

React enforces a one-way data flow, and props are intended to be read-only within the receiving component. If you need to modify data, it should be done at the component that owns the state.

107. How to focus an input element on page load?

You can focus an input element on page load by using the `ref` attribute and the `focus` method. Here's an example:

```jsx
import React, { useRef, useEffect } from 'react';

const MyComponent = () => {
  const inputRef = useRef(null);

  useEffect(() => {
    // Focus the input element on mount
    inputRef.current.focus();
  }, []);

  return <input ref={inputRef} />;
};
```

In this example, the `useEffect` hook with an empty dependency array ensures that the focus is set only once when the component mounts.

108. What are the possible ways of updating objects in state?

There are several ways to update objects in state in React:

1. Using the spread operator:

```jsx
setMyObject({ ...myObject, key: 'new value' });
```

2. Using `Object.assign`:

```jsx
setMyObject(Object.assign({}, myObject, { key: 'new value' }));
```

3. Using the functional form of `setState`:

```jsx
setMyObject((prevObject) => ({ ...prevObject, key: 'new value' }));
```

The third approach is preferred when the new state depends on the previous state, especially in asynchronous scenarios.

110. How can we find the version of React at runtime in the browser?

You can find the version of React at runtime in the browser by checking the `React.version` property. Here's an example:

```jsx
const reactVersion = React.version;
console.log('React version:', reactVersion);
```

This can be useful for logging or conditionally applying logic based on the React version being used in your application. Note that this property is available if you have React included in your project.

111. What are the approaches to include polyfills in your create-react-app?

To include polyfills in a Create React App (CRA), you can use the `react-app-polyfill` package. Follow these steps:

1. Install the package:

```bash
npm install react-app-polyfill
```

2. Import the polyfill at the top of your application's entry point (usually `index.js` or `index.tsx`):

```jsx
import 'react-app-polyfill/ie11'; // For IE11 support
import 'react-app-polyfill/stable';
```

This approach allows you to add polyfills for specific browsers or features, improving compatibility.

112. How to use HTTPS instead of HTTP in create-react-app?

By default, Create React App serves your application over HTTP during development. To use HTTPS instead, you can run the following command:

```bash
HTTPS=true npm start
```

This command sets the `HTTPS` environment variable to `true` before starting the development server, enabling HTTPS.

113. How to avoid using relative path imports in create-react-app?

In Create React App, you can set up absolute imports by creating a `jsconfig.json` file in the root of your project and configuring the `baseUrl` option. For example:

```json
{
  "compilerOptions": {
```

```
    "baseUrl": "src"
  },
  "include": ["src/**/*"]
}
```

With this setup, you can import modules using absolute paths:

```jsx
import MyComponent from 'components/MyComponent';
```

114. How to add Google Analytics for react-router?

To add Google Analytics for React Router, you can use the `react-ga` library. Follow these steps:

1. Install the `react-ga` package:

```bash
npm install react-ga
```

2. Import and initialize `react-ga` in your application's entry point (e.g., `index.js` or `index.tsx`):

```jsx
import ReactGA from 'react-ga';

ReactGA.initialize('Your-GA-Tracking-ID');
```

3. Use `ReactGA.pageview` to track page views in your route components:

```jsx
import React, { useEffect } from 'react';
import ReactGA from 'react-ga';
```

```
const MyComponent = () => {
  useEffect(() => {
    ReactGA.pageview(window.location.pathname + window.location.search);
  }, []);

  return (
    // Component content
  );
};

export default MyComponent;
```

115. How to update a component every second?

You can use the `setInterval` function within the `useEffect` hook to update a component every second. Here's an example using functional components and hooks:

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

const MyComponent = () => {
  const [count, setCount] = useState(0);

  useEffect(() => {
    const intervalId = setInterval(() => {
      setCount((prevCount) => prevCount + 1);
    }, 1000);

    return () => clearInterval(intervalId);
  }, []);
```

```
  return <div>{count}</div>;
};

export default MyComponent;
```

116. How do you apply vendor prefixes to inline styles in React?

Instead of manually applying vendor prefixes, you can use the `autoprefixer` feature provided by the `postcss` tool. Create React App already includes this setup by default, so you don't need to worry about it.

When you use inline styles with React, the styles are automatically transformed and prefixed during the build process.

117. How to import and export components using React and ES6?

You can import and export components using ES6 syntax. Here's an example:

Exporting a Component:

```jsx
// MyComponent.js
import React from 'react';

const MyComponent = () => {
  // Component logic
};

export default MyComponent;
```

Importing a Component:

```jsx
```

```
// AnotherComponent.js
import React from 'react';
import MyComponent from './MyComponent'; // Assuming it's in the same directory

const AnotherComponent = () => {
 return (
  <div>
   <MyComponent />
  </div>
 );
};

export default AnotherComponent;
```

118. What are the exceptions on React component naming?

React component names must begin with a capital letter. There are two exceptions to this rule:

1. Built-in Components: Components like `div` or `span` must be lowercase.

2. React Fragments: You can use a lowercase name for React fragments, which is a common pattern:

```jsx
const MyComponent = () => (
 <>
  {/* Fragment contents */}
 </>
);
```

119. Why is a component constructor called only once?

The constructor of a React component is called only once during the component's lifecycle when the component is being initialized. It is responsible for setting up the initial state and binding event handlers. Subsequent renders of the component do not invoke the constructor again.

120. How to define constants in React?

You can define constants in React using the `const` keyword. Here's an example:

```jsx
// Constants.js
export const API_BASE_URL = 'https://api.example.com';
export const MAX_COUNT = 10;
```

Then, you can import these constants in your components:

```jsx
// MyComponent.js
import React from 'react';
import { API_BASE_URL, MAX_COUNT } from './Constants';

const MyComponent = () => {
  console.log(API_BASE_URL, MAX_COUNT);
  // Component logic
};

export default MyComponent;
```

121. How to programmatically trigger click event in React?

You can programmatically trigger a click event in React using the `ref` attribute and the `click` method. Here's an example:

```jsx
import React, { useRef } from 'react';

const MyComponent = () => {
  const buttonRef = useRef(null);

  const handleClick = () => {
    console.log('Button clicked');
  };

  // Trigger click programmatically
  const triggerClick = () => {
    buttonRef.current.click();
  };

  return (
    <div>
      <button ref={buttonRef} onClick={handleClick}>


Click me
      </button>
      <button onClick={triggerClick}>
        Trigger Click
      </button>
    </div>
  );
};
export default MyComponent;
```

aswanth6000

122. Is it possible to use async/await in plain React?

Yes, it is possible to use `async/await` in React, but you typically use it within lifecycle methods or functional components with the help of the `useEffect` hook. Here's an example:

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

const MyComponent = () => {
  const [data, setData] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await fetch('https://api.example.com/data');
        const result = await response.json();
        setData(result);
      } catch (error) {
        console.error('Error fetching data:', error);
      }
    };

    fetchData();
  }, []);

  return (
    <div>
      {/* Render data */}
    </div>
  );
};
```

```
export default MyComponent;
```
```

### 123. What are the common folder structures for React?

Common folder structures for React projects include:

- **src:** Contains the source code of the React application.
- **components:** React components.
- **containers:** Higher-level components or containers that manage state.
- **pages:** Top-level components representing different pages or views.
- **services:** API calls, utility functions, etc.
- **styles:** CSS or styling related files.
- **assets:** Images, fonts, etc.
- **context:** React context providers and consumers.
- **utils:** Utility functions.
- **public:** Contains public assets like `index.html`, images, and other files.
- **tests:** Test files and configurations.
- **config:** Configuration files.
- **build:** Output directory after the build.

### 124. What are the popular packages for animation?

Some popular packages for animation in React include:

- **react-spring:** A spring-physics-based animation library.
- **framer-motion:** A declarative animation library for React.
- **react-transition-group:** A set of components for managing component states during transitions.
- **react-animate-on-scroll:** An animation library for scrolling animations.
- **react-reveal:** A library for simple fade and zoom effects during component mount.
```

125. What is the benefit of styles modules?

Styles modules, commonly known as CSS modules in React, provide local scope for styles. This helps in avoiding naming conflicts and allows encapsulation of styles within a component. Each component can have its own unique styles without worrying about global styles affecting it.

For example:

```jsx
// styles.module.css
.button {
  background-color: red;
}
```

```jsx
// ButtonComponent.js
import React from 'react';
import styles from './styles.module.css';

const ButtonComponent = () => {
  return <button className={styles.button}>Click me</button>;
};

export default ButtonComponent;
```

126. What are the popular React-specific linters?

Popular React-specific linters include:

- **eslint-plugin-react:** ESLint plugin for React.
- **eslint-plugin-react-hooks:** ESLint plugin for React hooks.
- **eslint-config-react-app:** ESLint configuration used by Create React App.
- **prettier:** Code formatter often used in combination with ESLint.

These tools help maintain a consistent and high-quality codebase in React applications.

127. How to make an AJAX call, and in which component lifecycle methods should I make an AJAX call?

You can make AJAX calls (e.g., fetching data from an API) using the `fetch` API or libraries like Axios. The common lifecycle methods for making AJAX calls in a class component are `componentDidMount` or `componentDidUpdate`.

Example using `fetch` in a functional component with `useEffect`:

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

const MyComponent = () => {
  const [data, setData] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await fetch('https://api.example.com/data');
        const result = await response.json();
        setData(result);
      } catch (error) {
        console.error('Error fetching data:', error);
      }
    };

    fetchData();
  }, []); // Empty dependency array ensures it runs once on mount

  return (
    <div>
```

```
      {/* Render data */}
    </div>
  );
};

export default MyComponent;
```

128. What are render props?

Render props is a pattern in React where a component receives a function as a prop and uses that function to render its content. This allows for greater component composition and code reuse. Here's a simple example:

```jsx
import React from 'react';

const RenderPropsComponent = ({ render }) => {
  return <div>{render('Hello from Render Props!')}</div>;
};

const App = () => {
  return (
    <RenderPropsComponent
      render={(content) => (
        <div>
          <h1>{content}</h1>
        </div>
      )}
    />
  );
};
```

```
export default App;
```
```

In this example, the `RenderPropsComponent` takes a `render` function as a prop and uses it to render content. The `App` component provides a function to be rendered by `RenderPropsComponent`.
```