Web - Mock Technical Interview Questions

🟢 BEGINNER 🟢

What is the difference between a block-level element and an inline element in HTML?

An element is considered a **block-level element** if it takes up the full width of its parent element and creates a new line after it. Block-level elements are typically used to define the structure of the page, such as headings, paragraphs, and sections.

Example:

```
<div>
<h1>
<p>
<form>
```

An **inline element**, on the other hand, is an element that is contained within a block-level element and only takes up as much width as necessary. Inline elements are typically used to add formatting or style to specific parts of a block of text, such as bolding or italicizing certain words.

Example:

```
<span>
<a>
<strong>
<em>
```

What is the difference between an ID and a class in CSS?

An **ID** is a unique identifier that is used to identify a single element on a page. An element can have only one ID, and an ID can be used to apply styles to a single element.

A **class**, on the other hand, is a way of identifying multiple elements on a page that share a common characteristic. A class can be used to apply styles to multiple elements at once.

Example:

```
<style>
  /* Style applied to a single element with the ID "header" */
  #header {
    background-color: blue;
  }

  /* Style applied to all elements with the class "highlight" */
  .highlight {
    color: red;
  }
</style>

<div id="header">This element has the ID "header"</div>
<p class="highlight">This element has the class "highlight"</p>
<p class="highlight">This element also has the class "highlight"</p>
```

How do you use JavaScript to add interactivity to a webpage?

1. **Event handling:** You can use JavaScript to specify what should happen when a user interacts with a webpage, such as clicking on a button or hovering over a link. For example, you might use JavaScript to display a message or change the color of an element when a user clicks on a button.

2. **Manipulating the DOM:** The Document Object Model (DOM) is a way of representing an HTML document as a tree-like structure of objects. You can use JavaScript to manipulate the DOM by adding, deleting, or changing elements on a webpage. For example, you might use JavaScript to add a new element to a page or to change the text of an existing element.

3. **Making HTTP requests:** You can use JavaScript to make requests to external servers and retrieve data, such as images or text. This can be used to load content dynamically or to update a webpage without needing to refresh the page.

Example:

```
<button onclick="changeText()">Click me</button>
<p id="my-paragraph">This is some initial text</p>

<script>
  function changeText() {
    // Get the element with the ID "my-paragraph"
    var paragraph = document.getElementById("my-paragraph");

    // Change the text of the element
    paragraph.innerHTML = "The text has been changed!";
  }
</script>
```

What is the DOM and how does it relate to JavaScript?

The **Document Object Model (DOM)** is a way of representing an HTML document as a tree-like structure of objects. Each element in the HTML document is represented as an object in the DOM, and the relationships between elements are represented as the tree structure.

JavaScript can be used to **interact** with the DOM by **manipulating the elements** and their properties. For example, you can use JavaScript to change the text of an element, add a new element to the page, or delete an existing element.

🟡 INTERMEDIATE 🟡

What is the difference between a function component and a class component in React?

A component is a piece of code that represents a part of a user interface (UI). There are two main ways to define a component in React: as a function component or as a class component.

A **function component** is a function that returns a React element. Function components are usually used for simple components that only need to render some UI and don't have any state or lifecycle methods.

Example:

```
import React from 'react';

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

A **class component**, on the other hand, is a class that extends the `React.Component` class. Class components are used for more complex components that need to have state or lifecycle methods.

Example:

```
import React from 'react';

class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

How do you use props and state in a React component?

Props and state are used to pass data and manage the internal state of a component.

Props (short for "**properties**") are values that are passed to a component from its parent. Props are read-only, which means that the component cannot modify its props. Instead, the parent component is responsible for updating the props if necessary.

Example:

```
import React from 'react';

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

// This component will render the message "Hello, Alice"
<Welcome name="Alice" />
```

State, on the other hand, is used to store data that is specific to a component and can change over time. A component can update its own state, but it should do so in a controlled way using React's `setState()` method.

Example:

```
import React from 'react';

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

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

  render() {
    return (
      <div>
        <button onClick={() => this.increment()}>Increment</button>
        <p>{this.state.count}</p>
      </div>
    );
  }
}
```

What is the difference between a controlled component and an uncontrolled component in React?

A **controlled component** is a form element whose value is controlled by the state of the component. This means that the value of the form element is determined by the state of the component, and any changes to the form element are reflected in the state.

An **uncontrolled component**, on the other hand, is a form element whose value is not controlled by the state of the component. This means that the value of the form element is determined by the user's input, and any changes to the form element are not reflected in the state.

How do you use the React lifecycle methods?

Lifecycle methods are special methods that are called at specific points in the life of a component. These methods allow you to perform certain actions at specific points in the component's lifecycle, such as setting up subscriptions or making API calls.

Common lifecycle methods:

- **componentDidMount()**: This method is called after the component has been rendered to the DOM. It is a good place to make API calls or set up subscriptions.

- **componentDidUpdate(prevProps, prevState)**: This method is called after the component has updated. It is a good place to make API calls or perform other actions based on the updated state of the component.

- **shouldComponentUpdate(nextProps, nextState)**: This method is called before the component updates. It allows you to decide whether or not the component should update based on the new props and state.

- **componentWillUnmount()**: This method is called before the component is removed from the DOM. It is a good place to clean up any subscriptions or resources that were set up in `componentDidMount()`.

How do you handle events in React?

You can handle events by using **event handlers**. An event handler is a function that is called when a specific event occurs, such as a button being clicked or a form being submitted.

To handle an event in React, you can use the `on[Event]` attribute, where `[Event]` is the name of the event you want to handle. For example, to handle a click event, you can use the `onClick` attribute.

Example:

```
import React from 'react';

class Button extends React.Component {
  handleClick() {
    console.log('The button was clicked!');
  }

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

🔴 ADVANCED 🔴

What is the concept of React Hooks and how they are used in a React application?

React Hooks are a new feature in React that allow you to use state and other React features without writing a class component. They were introduced in React 16.8 and have become an increasingly popular way to write React components.

Most commonly used Hooks:

- **useState()**: This Hook allows you to add state to a function component. It returns an array with two elements: the current state value and a function that allows you to update the state.

- **useEffect()**: This Hook allows you to perform side effects in a function component, such as making API calls or setting up subscriptions. It is similar to the `componentDidMount()`, `componentDidUpdate()`, and `componentWillUnmount()` lifecycle methods in a class component.

Example:

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

function DataFetcher() {
  const [data, setData] = useState(null);

  useEffect(() => {
    fetchData().then((response) => setData(response));
  }, []);

  return data ? <div>{data}</div> : <div>Loading...</div>;
}
```

How do you optimize the performance of a React application?

Common ways to optimize:

1. Use the `shouldComponentUpdate()` lifecycle method to avoid unnecessary re-renders. This method allows you to decide whether or not a component should update based on the new props and state. By returning `false` from this method, you can prevent the

component from re-rendering.

2. Use the `React.memo()` higher-order component to memorize functional components. This can help improve performance by avoiding unnecessary re-renders of components that have not changed.

3. Use the `useMemo()` Hook to memorize values that are expensive to compute. This can help improve performance by avoiding unnecessary computations.

4. Use the `useCallback()` Hook to memorize callback functions. This can help improve performance by avoiding unnecessary re-creating of functions.

5. Use the `React.lazy()` and `Suspense` components to lazy-load components that are not needed on the initial render. This can help improve performance by reducing the amount of code that needs to be loaded upfront.

6. Use a performance profiling tool, such as the React Developer Tools browser extension, to identify and fix performance issues in your application.

How do you use modern testing techniques, like snapshot testing, to test a React application?

Snapshot testing is a technique that allows you to test the output of a component by comparing it to a reference snapshot. The reference snapshot is a saved version of the component's output that is used as a baseline for comparison.

To use snapshot testing in a React application, you can use a library like *Jest*, which includes snapshot testing functionality out of the box.

Example:

```
import React from 'react';
import renderer from 'react-test-renderer';

test('Button component snapshot', () => {
  const component = renderer.create(<Button>Click me</Button>);
  const tree = component.toJSON();
  expect(tree).toMatchSnapshot();
});
```

How do you implement internationalization in a React application?

Internationalization, or "**i18n**" for short, refers to the process of adapting a software application for use in different languages and regions. In a React application, you can implement internationalization using a library or framework that provides i18n functionality.

One popular library for implementing i18n in a React application is `react-intl`. `react-intl` provides a set of components and APIs for formatting dates, numbers, and strings according to the conventions of a specific locale.

Example:

```
import { IntlProvider, FormattedDate } from 'react-intl';

function App() {
  return (
    <IntlProvider locale="en-US">
      <FormattedDate value={new Date()} />
    </IntlProvider>
  );
}
```