ReactJs Interview Questions

What are the advantages of ReactJS?

Entry

Below are the advantages of ReactJS:

- Increases the application's performance with Virtual DOM
- JSX makes code easy to read and write
- It renders both on the client and server-side
- Easy to integrate with other frameworks (Angular, BackboneJS) since it is only a view library
- Easy to write UI Test cases and integration with tools such as JEST.

How does React work?

Entry

React creates a virtual DOM. When state changes in a component it firstly runs a "diffing" algorithm, which identifies what has changed in the virtual DOM. The second step is reconciliation, where it updates the DOM with the results of the difference.

What is the use of refs?

Entry

Refs provide a way to access DOM nodes or React elements created in the render method. They should be avoided in most cases, however, they can be useful when we need direct access to the DOM element or an instance of a component.

There are a few good use cases for refs:

- Managing focus, text selection, or media playback.
- Triggering imperative animations.
- Integrating with third-party DOM libraries.

Refs are created using `React.createRef()` and attached to React elements via the `ref` attribute. Refs are commonly assigned to an instance property when a component is

constructed so they can be referenced throughout the component.

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.myRef = React.createRef();  }
  render() {
    return <div ref={this.myRef} />;  }
}
```

What are props in React?

Entry

Props are inputs to a React component. They are single values or objects containing a set of values that are passed to React Components on creation using a naming convention similar to HTML-tag attributes. i.e, *They are data passed down from a parent component to a child component.*

The primary purpose of props in React is to provide the following component functionality:

Pass custom data to your React component.

Trigger `state` changes.

Use via `this.props.reactProp` inside component's `render()` method.

For example, let us create an element with `reactProp` property,

```
<Element reactProp = "1" />
```

This `reactProp` (or whatever you came up with) the name then becomes a property attached to React's native props object which originally already exists on all components created using React library.

```
props.reactProp;
```

What is Context API in ReactJS?

Entry

Context provides a way to pass data through the component tree without having to pass props down manually at every level.

Context is designed to share data that can be considered “global” for a tree of React components, such as the current authenticated user, theme, or preferred language. Using context, we can avoid passing props through intermediate elements.

```
// Context lets us pass a value deep into the component tree
// without explicitly threading it through every component.
// Create a context for the current theme (with "light" as the default).
const ThemeContext = React.createContext('light');

class App extends React.Component {
  render() {
    // Use a Provider to pass the current theme to the tree below.
    // Any component can read it, no matter how deep it is.
    // In this example, we're passing "dark" as the current value.
    return (
      <ThemeContext.Provider value="dark">
        <Toolbar />
      </ThemeContext.Provider>
    );
  }
}

// A component in the middle doesn't have to
// pass the theme down explicitly anymore.
function Toolbar() {
  return (
    <div>
      <ThemedButton />
    </div>
  );
}

class ThemedButton extends React.Component {
  // Assign a contextType to read the current theme context.
  // React will find the closest theme Provider above and use its value.
  // In this example, the current theme is "dark".
  static contextType = ThemeContext;
  render() {
    return <Button theme={this.context} />;
  }
}
```

What are React Hooks?

Entry

Hooks are a new addition to React 16.8. They let you use state and other React features without writing a class.

With Hooks, you can extract stateful logic from a component so it can be tested independently and reused. Hooks allow you to reuse stateful logic without changing your component hierarchy. This makes it easy to share Hooks among many components or with the community.

What are the major features of ReactJS?

Entry

The major features of ReactJS are as follows,

It uses **VirtualDOM** instead RealDOM considering that RealDOM manipulations are expensive.

Supports **server-side rendering**

Follows **Unidirectional** data flow or data binding

Uses **reusable/composable** UI components to develop the view

How would you write an inline style in React?

Entry

For example:

```
<div style={{ height: 10 }}>
```

What is the difference between state and props?

Entry

The **state** is a data structure that starts with a default value when a Component mounts. It may be mutated across time, mostly as a result of user events.

Props (short for properties) are a Component's configuration. They are received from above and immutable as far as the Component receiving them is concerned. A Component cannot change its props, but it is responsible for putting together the props of its child Components. Props do not have to just be data - callback functions may be passed in as props.

What is JSX?

Entry

JSX is a syntax notation for **JavaScript XML** (XML-like syntax extension to ECMAScript). It stands for JavaScript XML. It provides the expressiveness of JavaScript along with HTML-like template syntax.

For example, the below text inside the **h1** tag return as a javascript function to the render function,

```
render(){
    return(
     <div>
```

```
            <h1> Welcome to React world!!</h1>
        </div>
      );
    }
```

What are the differences between a Class component and a Functional component?

Junior

Class Components

Class-based Components use ES6 class syntax. It can make use of the lifecycle methods.

Class components extend from `React.Component`.

Here you have to use this keyword to access the props and functions that you declare inside the class components.

Functional Components

Functional Components are simpler compared to class-based functions.

Functional Components mainly focus on the UI of the application, not on the behavior.

To be more precise these are basically rendered functions in the class component.

Functional Components can have state and mimic lifecycle events using Reach Hooks

What is the difference between a Presentational component and a Container component?

Junior

Presentational components are concerned with *how things look*. They generally receive data and callbacks exclusively via props. These components rarely have their own state, but when they do it generally concerns the UI state, as opposed to the data state.

Container components are more concerned with *how things work*. These components provide the data and behavior to presentational or other container components. They call Flux actions and provide these as callbacks to the presentational components. They are also often stateful as they serve as data sources.

What's the difference between a Controlled component and an Uncontrolled one in React?

Junior

This relates to stateful DOM components (form elements) and the React docs explain the difference:

> A Controlled Component is one that takes its current value through `props` and notifies changes through callbacks like `onChange`. A parent component "controls" it by handling the callback and managing its own state and passing the new values as props to the controlled component. You could also call this a "dumb component".
>
> An Uncontrolled Component is one that stores its own state internally, and you query the DOM using a `ref` to find its current value when you need it. This is a bit more like traditional HTML.

Most native React form components support both controlled and uncontrolled usage:

```
// Controlled:
<input type="text" value={value} onChange={handleChange} />

// Uncontrolled:
<input type="text" defaultValue="foo" ref={inputRef} />
// Use `inputRef.current.value` to read the current value of <input>
```

What does it mean for a component to be mounted in React?

Junior

It has a corresponding element created in the DOM and is connected to that.

Mention some limitations of React?

Junior

> React is just a view library, not a full-blown framework
>
> There is a learning curve for beginners who are new to web development.
>
> Integrating React.js into a traditional MVC framework requires some additional configuration
>
> The code complexity increases with inline templating and JSX.
>
> Too many smaller components lead to over-engineering or boilerplate

What are Stateless components in React?

Junior

If the behavior is independent of its state then it can be a **stateless component**. You can use either a function or a class for creating stateless components. But unless you need to use a lifecycle hook in your components, you should go for stateless functional components.

Stateful/Container/Smart component:

```
class Main extends Component {
 constructor() {
   super()
   this.state = {
     books: []
   }
 }
 render() {
   <BooksList books={this.state.books} />
 }
}
```

Stateless/Presentational/Dumb component:

```
const BooksList = ({books}) => {
 return (
   <ul>
     {books.map(book => {
       return <li>book</li>
     })}
   </ul>
 )
}
```

There are a lot of benefits if you decide to use stateless functional components here; they are:

easy to write, understand, test, and

you can avoid this keyword altogether.

What is the purpose of the callback function as an argument of setState?

Junior

The callback function is invoked when `setState` finished and the component gets rendered. Since `setState` is **asynchronous** the callback function is used for any post-action.

Note: It is recommended to use the lifecycle method rather than this callback function.

```
setState({name: 'protechstack'}, () => console.log('The name has updated and component re-rendered'
```

What are the advantages of using React Hooks?

Junior

Primarily, hooks in general enable the extraction and reuse of stateful logic that is common across multiple components without the burden of higher order components or render props. Hooks allow us to easily manipulate the state of our functional components without needing to convert them into class components.

Hooks don't work inside classes (because they let you use React without classes). By using them, we can totally avoid using lifecycle methods, such as `componentDidMount`, `componentDidUpdate`, `componentWillUnmount`. Instead, we will use built-in hooks like `useEffect` .

What are portals in React and when do we need them?

Junior

Portals provide a first-class way to render children into a DOM node that exists outside the DOM hierarchy of the parent component.

Sometimes it's useful to insert a child into a different location in the DOM:

```
render() {
  // React does *not* create a new div. It renders the children into `domNode`.
  // `domNode` is any valid DOM node, regardless of its location in the DOM.
  return ReactDOM.createPortal(
    this.props.children,
    domNode  );
}
```

A typical use case for portals is when a parent component has an `overflow: hidden` or `z-index` style, but you need the child to visually "break out" of its container.

What happens during the lifecycle of a React component?

Junior

At the highest level, React components have lifecycle events that fall into three general categories:

- Initialization
- State/Property Updates
- Destruction

What are inline conditional expressions in ReactJS?

Junior

You can use either if statements or ternary expressions which are available from JS to conditionally render expressions.

Apart from these approaches, you can also embed any expressions in JSX by wrapping them in curly braces and then followed by JS logical operator(&&).

```
if(this.state.mode === 'view') {
  return (
      <button onClick={this.handleEdit}>
        Edit
      </button>
  );
} else {
  return (
      <button onClick={this.handleSave}>
        Save
      </button>
  );
}
// or
 {
   view
  ? null
  : (
    <p>
      <input
        onChange={this.handleChange}
        value={this.state.inputText} />
    </p>
  )
}
```

What is the purpose of using super constructor with props argument in React? Junior

A child class constructor cannot make use of **this** reference until `super()` the method has been called. The same applies to ES6 sub-classes as well. The main reason for passing the props parameter to `super()` call is to access `this.props` your child constructors.

Passing props:

```
class MyComponent extends React.Component {
    constructor(props) {
        super(props);
        console.log(this.props);  // Prints { name: 'protechstack',age: 30 }
    }
}
```

Not passing props:

```
class MyComponent extends React.Component {
    constructor(props) {
        super();
        console.log(this.props); // Prints undefined
        // But Props parameter is still available
        console.log(props); // Prints { name: 'protechstack',age: 30 }
    }

    render() {
        // No difference outside constructor
        console.log(this.props) // Prints { name: 'protechstack',age: 30 }
    }
}
```

The above code snippets reveal that `this.props` behavior is different only within the constructor. It would be the same outside the constructor.

How to access DOM elements in React?

Junior

One of the useful application of the `useRef()` hook is to access DOM elements. This is performed in 3 steps:

Define the reference to access the element `const elementRef = useRef();`

Assign the reference to `ref` attribute of the element: `<div ref={elementRef}></div>`;

After mounting, `elementRef.current` points to the DOM element.

Consider:

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

function AccessingElement() {

  const elementRef = useRef();

  useEffect(() => {
    const divElement = elementRef.current;
    console.log(divElement); // logs <div>I'm an element</div>
  }, []);

  return (
    <div ref={elementRef}>
      I'm an element
    </div>
```

```
    );
}
```

What are Higher-Order Components (HOC) in React? Junior

A higher-order component **(HOC)** is a function that takes a component and returns a new component. Basically, it's a pattern that is derived from React's compositional nature We call them as **"pure' components"** because they can accept any dynamically provided child component but they won't modify or copy any behavior from their input components.

```
const EnhancedComponent = higherOrderComponent(WrappedComponent);
```

HOC can be used for many use cases as below,

- Code reuse, logic, and bootstrap abstraction
- Render High jacking
- State abstraction and manipulation
- Props manipulation

What happens when you call setState? Junior

The first thing React will do when `setState` is called is to merge the object you passed into `setState` into the current state of the component. This will kick off a process called **reconciliation**. The end goal of reconciliation is to, in the most efficient way possible, update the UI based on this new state.

To do this, React will construct a new tree of React elements (which you can think of as an object representation of your UI). Once it has this tree, in order to figure out how the UI should change in response to the new state, React will diff this new tree against the previous element tree.

By doing this, React will then know the exact changes which occurred, and by knowing exactly what changes occurred, will be able to minimize its footprint on the UI by only making updates where absolutely necessary.

What is the difference between Element and Component in ReactJS? Junior

An **element** is a plain object describing what you want to appear on the screen in terms of the DOM nodes or other components. Elements can contain other elements in their props.

Creating a React element is cheap. Once an element is created, it is never mutated. The object representation of React element would be as follows,

```
const element = React.createElement(
  'div',
  {id: 'login-btn'},
  'Login'
)
```

The above `createElement` returns as object as below,

```
{
  type: 'div',
  props: {
    children: 'Login',
    id: 'login-btn'
  }
}
```

And finally it renders to the DOM using `ReactDOM.render` as below,

```
<div id='login-btn'>Login</div>
```

Whereas a **component** can be declared in several different ways. It can be a class with a `render()` method. Alternatively, in simple cases, it can be defined as a function. In either case, it takes props as an input and returns an element tree as the output. JSX transpiled as `createElement` at the end.

```
function Button ({ onLogin }) {
  return React.createElement(
    'div',
    {id: 'login-btn', onClick: onLogin},
    'Login'
  )
}
```

What is Lifting State Up in ReactJS?

Mid

When several components need to share the same changing data then it is recommended to lift the shared state up to their closest common ancestor. For example, if two child

components share the same data from their parent then move the state to the parent instead of maintaining the local state in both child components.

What is the Key and benefit of using it in lists?

Mid

A **key** is a special string attribute you need to include when creating lists of elements. Keys help React identify which items have changed, are added, or are removed.

For example, most often we use IDs from your data as keys

```
const todoItems = todos.map((todo) =>
    <li key={todo.id}>
    {todo.text}
    </li>
);
```

When you don’t have stable IDs for rendered items, you may use the item index as a key as a last resort:

```
const todoItems = todos.map((todo, index) =>
    <li key={index}>
        {todo.text}
    </li>
);
```

Note:

> We don’t recommend using indexes for keys if the order of items may change. This can negatively impact performance and may cause issues with component state
>
> If you extract the list item as a separate component then apply keys on the list component instead li tag.

There will be a warning in the console if the key is not present on the list items.

Why does React use className over class attribute?

Mid

class is a reserved keyword in javascript and JSX is an extension of javascript. That's the principal reason why React uses `className` instead of class.

```
render() {
  return <span className="menu navigation-menu">Menu</span>
}
```

How to prevent a function from being called every time when component renders? Mid

You need to make sure that the function is not being called while passing the function as a parameter.

```
render() {
  // Wrong way: handleClick is called instead of passed as a reference!
  return <button onClick={this.handleClick()}>Click Me</button>
}
```

Instead, pass the function itself without parenthesis:

```
render() {
  // Correct way: handleClick is passed as a reference!
  return <button onClick={this.handleClick}>Click Me</button>
}
```

What is StrictMode in React? Mid

React's `StrictMode` is sort of a helper component that will help you write better react components, you can wrap a set of components with `<StrictMode />` and it'll basically:

- Verify that the components inside are following some of the recommended practices and warn you if not in the console.
- Verify the deprecated methods are not being used, and if they're used strict mode will warn you in the console.
- Help you prevent some side effects by identifying potential risks.

Why we should not update the state directly? Mid

If you try to update `state` directly then it won't re-render the component.

```
//Wrong
This.state.message =”Hello world”;
```

Instead, use `setState()` method. It schedules an update to a component's state object. When the state changes, the component responds by re-rendering

```
//Correct
this.setState({message: ‘Hello World’});
```

What is the difference between ShadowDOM and VirtualDOM?

Senior

Virtual DOM

Virtual DOM is about avoiding unnecessary changes to the DOM, which are expensive performance-wise because changes to the DOM usually cause re-rendering of the page. Virtual DOM also allows to collect of several changes to be applied at once, so not every single change causes a re-render, but instead, re-rendering only happens once after a set of changes was applied to the DOM.

Shadow DOM

Shadow dom is mostly about the encapsulation of the implementation. A single custom element can implement more-or-less complex logic combined with more-or-less complex DOM. An entire web application of arbitrary complexity can be added to a page by an import and `<body><my-app></my-app>` but also simpler reusable and composable components can be implemented as custom elements where the internal representation is hidden in the shadow DOM like `<date-picker></date-picker>`.

How would you go about investigating slow React application rendering?

Senior

One of the most common issues in React applications is when components *re-render unnecessarily*. There are two tools provided by React that are helpful in these situations:

`React.memo()`: This prevents unnecessary re-rendering of function components

`PureComponent`: This prevents unnecessary re-rendering of class components

Both of these tools rely on a shallow comparison of the props passed into the component—if the props have not changed, then the component will not re-render. While both tools are very useful, the shallow comparison brings with it an additional performance penalty, so both can have a negative performance impact if used incorrectly. By using the React Profiler, performance can be measured before and after using these tools to ensure that performance is actually improved by making a given change.

When shall we use useReducer hook in ReactJS?

Senior

`useReducer` is an alternative to `useState`. `useReducer` is usually preferable to `useState` when you have complex state logic that involves **multiple** sub-values or when the next state *depends* on the previous one.

An example will be a list of items, where you need to add, update and remove items in the state, Here you might have noticed that the state management logic takes a good part of the component body. `useReducer` helps to separate the concerns of rendering vs a concern of state management.

`useReducer` also lets you optimize performance for components that trigger deep updates because you can pass `dispatch` down instead of callbacks.

```
const initialState = {count: 0};

function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return {count: state.count + 1};
    case 'decrement':
      return {count: state.count - 1};
    default:
      throw new Error();
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <>
      Count: {state.count}
      <button onClick={() => dispatch({type: 'decrement'})}>-</button>
      <button onClick={() => dispatch({type: 'increment'})}>+</button>
    </>
  );
}
```

React guarantees that `dispatch` function identity is stable and won't change on re-renders. This is why it's safe to omit it from the `useEffect` or `useCallback` dependency list.

What is React Fiber?

Expert

React Fiber is an ongoing reimplementation of React's core algorithm. The main difference between react and react fiber are these new features:-

Incremental Rendering:- React v16.0 includes a completely rewritten server renderer. It's really fast. It supports streaming, so you can start sending bytes to the client faster

Handle errors in the render API: To make class component an error boundary we define a new lifecycle method called componentDidCatch(error, info).

Return multiple elements from render: With this new feature in React v16.0 now we can also return an array of elements, and strings from the component's render method.

Portals: Portals provide a first-class way to render children into a DOM node that exists outside the DOM hierarchy of the parent component.

Fragments: A common pattern in React is for a component to return multiple elements. Fragments let you group a list of children without adding extra nodes to the DOM.

What is the key architectural difference between JavaScript libraries such as React and JavaScript? Expert

React enables developers to render a user interface. To create a full front-end application, developers need other pieces, such as state management tools like Redux.

Like React, Angular enables developers to render a user interface, but it is a "batteries included" framework that includes prescriptive, opinionated solutions to common requirements like state management.

While there are many other considerations when comparing React and Angular specifically, this key architectural difference means that:

Using a library such as React can give a project a greater ability to evolve parts of the system—again for example, state management—over time, when new solutions are created by the open source community.

Using a framework such as Angular can make it easier for developers to get started and can also simplify maintenance.

What is the order of useInsertionEffect, useEffect and useLayoutEffect hooks at component generation Expert

`useInsertionEffect`

It fires synchronously **before** all DOM mutations. Use this to inject styles into the DOM **before** reading the layout in `useLayoutEffect`. So it runs before `useLayoutEffect`.

`useLayoutEffect`

It fires synchronously **after** all DOM mutations. Use this to read the layout from the DOM and synchronously re-render.

`useEffect`

It will run after the render is committed to the screen. So it runs after `useLayoutEffect`.

Therefore the order of running is:

```
useInsertionEffect

useLayoutEffect

useEffect
```