REACTJS NOTES WITH MOST COMMON QUESTIONS

@himanshu_shekhar16

Reacts Most Asked Questions

- What is React js
- What is difference between virtual dom and shallow dom, dom in React js
- What is controlled and uncontrolled component in React js
- What is hooks in React js
- What is jsx, babel, webpack
- What is Redux
- What is reducer, action, store in Redux
- What is middleware in Redux
- Explain data flow in Redux
- What is Redux-Thunk
- What is Redux-Saga, Difference between Redux-thunk and Redux-saga
- Difference between class component and function component
- How can we implement componentWillUnmount in function component
- useEffect,UseState, useMemo.useCallback hooks in Details
- Explain lifecycle method in React js
- What is difference between export default and export in React js
- What is portal in React js

- What is reconciliation in React js
- What is useRef in React js
- What is server side render in React js
- What is useStrict in React js
- What is fragment in React js
- What is react router in React js
- What is node module in React js
- What is the default localhost server port in react js. how can we change the local server port
- What is high order component in React js
- What is pure component in React js
- What is difference state and props in React js
- How to optimize React js app
- What is difference between React js and Angular js
- What is prop drilling in React js how to overcome it
- What is context api in React js
- What is super, constructor, render function in React

What is React?

- React is officially defined as a "JavaScript library for creating user interfaces," but what does that really mean?
- React is a library, made in JavaScript and which we code in JavaScript, to build great applications that run on the web.

What do I need to know for React?

- In other words, you do need to have a basic understanding of JavaScript to become a solid React programmer.
- The most basic JavaScript concepts you should be familiar with are variables, basic data types, conditionals, array methods, functions, and ES modules.
- How do I learn all of these JavaScript skills? Check out the comprehensive guide to learn all of the JavaScript you need for React.

If React was made in JavaScript, why don't we just use JavaScript?

- While React was written in JavaScript, which was built from the ground up for the express purpose of building web applications and gives us tools to do so.
- JavaScript is a 20 year old language which was created for adding small bits of behavior to the browser through scripts and was not designed for creating complete applications.
- In other words, while JavaScript was used to create React, they were created for very different purposes.

Can I use JavaScript in React applications?

- Yes! Any valid JavaScript code can be included within your React applications.
- You can use any browser or window API, such as relocation or the fetch API.

- Also, since React (when it is compiled) runs in the browser, you can perform common JavaScript actions like DOM querying and manipulation.

How to Create React Apps

Three different ways to create a React application

- Putting React in an HTML file with external scripts
- Using an in-browser React environment like Code Sandbox
- Creating a React app on your computer using a tool like Create React App

What is the best way to create a React app?

- Which is the best approach for you? The best way to create your application depends on what you want to do with it.
- If you want to create a complete web application that you want to ultimately push to the web, it is best to create that React application on your computer using a tool like Create React App.
- If you are interested in creating React apps on your computer, check out the complete guide to using Create React App.
- The easiest and most beginner-friendly way to create and build React apps for learning and prototyping is to use a tool like Code Sandbox. You can create a new React app in seconds by going to react. New!

JSX Elements

JSX is a powerful tool for structuring applications

- **JSX** is meant to make create user interfaces with JavaScript applications easier.
- JSX borrows its syntax from the most widely used programming language: HTML
- As a result, JSX is a powerful tool to structure our applications.

- The code example below is the most basic example of a React element which displays the text "Hello World"

```
<div>Hello React!</div>
```

> To be displayed in the browser, React elements need tobe rendered (using ReactDOM.render())

How JSX differs from HTML

- We can write valid HTML element in JSX, but what differs slightly is the way some attributes are written.
- Attributes that consist of multiple words are written in the camel-case syntax (i.e. `className`) and have different names than standard HTML (`class`).

```
<div id="header">
  <h1 className="title">Hello React!</h1>
</div>
```

- The reason JSX has this different way of writing attributes is because it is actually made using JavaScript functions (more on this later).

JSX must have a trailing slash if it is made of one tag

- Unlike standard HTML, elements like `input` , `img` , or `br` must close with a trailing forward slash for it to be valid JSX.

```
<input type="email" />// <input type="email"> is a syntax error
```

JSX elements with two tags must have a closing tag

- Elements that should have two tags, such as `div` , `main` or `button` , must have their closing, second tag in JSX, otherwise it will result in a syntax error.

```
<button>Click me</button>// <button> or </button> is a syntax error
```

How JSX elements are styled

- Inline styles are written differently as well as compared to plain HTML.
- Inline styles must not be included as a string, but within an object.
- Once again, the style properties that we use must be written in the camel-case style.

```
<h1 style={{ color: "blue", fontSize: 22, padding: "0.5em 1em" }}>
  Hello React!
</h1>;
```

Style properties that accept pixel values (like width, height, padding, margin, etc), can use integers instead of strings. For example, fontSize: 22 instead of fontSize: "22px"

JSX can be conditionally displayed

- New React developers may be wondering how it is beneficial that React can use JavaScript code.
- One simple example if that to conditionally hide or display JSX content, we can use any valid JavaScript conditional, like an if statement or switch statement.

```
const isAuthUser = true;

if (isAuthUser) {
  return <div>Hello user!</div>
} else {
  return  <button>Login</button>
}
```

- Where are we returning this code? Within a React component, which we will cover in a later section.

JSX cannot be understood by the browser

- As mentioned above, JSX is not HTML, but composed of JavaScript functions.

- In fact, writing `<div>Hello React</div>` in JSX is just a more convenient and understandable way of writing code like the following:

```
React.createElement("div", null, "Hello React!")
```

- Both pieces of code will have the same output of "Hello React".
- To write JSX and have the browser understand this different syntax, we must use a **transpiler** to convert JSX to these function calls.
- The most common transpiler is called **Babel.**

What are React components?

- Instead of just rendering one or another set of JSX elements, we can include them within React **components**.
- Components are created using what looks like a normal JavaScript function, but is different in that it returns JSX elements.

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

Why use React components?

- React components allow us to create more complex logic and structures within our React application than we would with JSX elements alone.
- Think of React components as our custom React elements that have their own functionality.
- As we know, functions allow us to create our own functionality and reuse it where we like across our application.
- Components are reusable wherever we like across our app and as many times as we like.

Components are not normal JavaScript functions

- How would we render or display the returned JSX from the component above?

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


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

- We use the `React` import to parse the JSX and `ReactDOM` to render our component to a **root element** with the id of "root."

What can components return?

- Components can return valid JSX elements, as well as strings, numbers, booleans, the value `null`, as well as arrays and fragments.
- Why would we want to return `null`? It is common to return `null` if we want a component to display nothing.

```
function Greeting() {
  if (isAuthUser) {
    return "Hello again!";
  } else {
    return null;
  }
```

- Another rule is that JSX elements must be wrapped in one parent element. Multiple sibling elements cannot be returned.
- If you need to return multiple elements, but don't need to add another element to the DOM (usually for a conditional), you can use a special React component called a fragment.
- Fragments can be written as `<></>` or when you import React into your file, with `<React.Fragment></React.Fragment>`.

```
function Greeting() {
  const isAuthUser = true;


  if (isAuthUser) {
    return (
      <>
        <h1>Hello again!</h1>
        <button>Logout</button>
      </>
    );
  } else {
```

> Note that when attempting to return a number of JSX elementsthat are spread over multiple lines, we can return it all using a set of parentheses () as you see in the example above.

Components can return other components

- The most important thing components can return is other components.
- Below is a basic example of a React application contained with in a component called `App` that returns multiple components:

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


import Layout from './components/Layout';
import Navbar from './components/Navbar';
import Aside from './components/Aside';
import Main from './components/Main';
import Footer from './components/Footer';


function App() {
  return (
    <Layout>
      <Navbar />
      <Main />
      <Aside />
```

```
ReactDOM.render(<App />, document.getElementById('root'));
```

- What is powerful about this is that we are using the customization of components to describe what they are (i.e. Layout) and their function in our application. This tells us how they should be used just by looking at their name.
- Additionally, we are using the power of JSX to compose these components. In other words, to use the HTML-like syntax of JSX to structure them in an immediately understandable way (i.e. the Navbar is at the top of the app, the Footer at the bottom, etc).

JavaScript can be used in JSX using curly braces

- Just as we can use JavaScript variables within our components, we can use them directly within our JSX as well.
- There are a few core rules to using dynamic values within JSX, however.
- JSX can accept any primitive values (strings, booleans, numbers), but it will not accept plain objects.
- JSX can also include expressions that resolve to these values.
- For example, conditionals can be included within JSX using the ternary operator, since it resolves to a value.

```
function Greeting() {
  const isAuthUser = true;


  return <div>{isAuthUser ? "Hello!" : null}</div>;
```

Props

Components can be passed values using props

- Data passed to components in JavaScript are called **props**

- Props look identical to attributes on plain JSX/HTML elements, but you can access their values within the component itself
- Props are available in parameters of the component to which they are passed. Props are always included as properties of an object

```
ReactDOM.render(
  <Greeting username="John!" />,
  document.getElementById("root")
);


function Greeting(props) {
```

Props cannot be directly changed

- Props must never be directly changed within the child component.
- Another way to say this is that props should never be **mutated**, since props are a plain JavaScript object

```
// We cannot modify the props object:function Header(props) {
  props.username = "Doug";


  return <h1>Hello {props.username}</h1>;
```

> Components are consider pure functions. That is, for every input, we should be able to expect the same output. This means we cannot mutate the props object, only read from it.

Special props: the children prop

- The **children** prop is useful if we want to pass elements / components as props to other components
- The children prop is especially useful for when you want the same component (such as a Layout component) to wrap all other components.