React Question Answers

1. What is Hooks ?

Hooks is a new feature of react js version 16.8 which is use for use all the features and props of Class components without using class comp. and for stop writing **'THIS'** word. They let developers use state and other React features without writing a class For example- State of a component It is important to note that hooks are not used inside the classes. The useState hook returns a pair of values: the current state and a function that updates it. In this example, we're using the setCount function to increment the count state variable whenever the button is clicked.**Simple syntax of hooks in React JS:**

- Hooks are functions that start with the word "use".
- Hooks can be used to manage state and side effects.
- Hooks can only be called at the top level of a function component.
- Hooks can only be called from React function components.

2. What is functional component?

A **React functional component** is a simple JavaScript function that accepts **props** and returns a React element.

After the introduction of React Hooks, writing functional components has become the standard way of writing React components in modern applications. React Hooks were **introduced** in React 16.8[16.8 – they allow you to use state without writing *class components*. It is the traditional way of writing React components in modern applications since the introduction of React Hooks. The only constraint for a functional component is to accept props as an argument and return valid JSX.5 The primary purpose of a React component is to define the displayed view and bind it to the code that drives its behavior.1 Functional components can be created using specific function keywords, such as Arrow, and their primary purpose is to render the view and data to the browser.

3. Difference between useEffect and useMemo

Feature	useEffect	useMemo
Purpose	Perform side effects.Side effects are anything that happens outside of the component, such as making an API call, setting a timer, or updating the DOM.	Memoize the result of a calculation or function.Memoization is a technique where the result of a calculation is stored so that it can be reused later without having to recalculate it.
When to use	When you need to perform a side effect, such as making an API call, setting a timer, or updating the DOM	When you need to memoize the result of a calculation or function to avoid expensive recalculations
Dependencies	Takes an optional array of dependencies. If any of the dependencies change, the side effect will be run again.	Takes an array of dependencies. If any of the dependencies change, the memoized value will be recalculated.
Return value	None	The memoized value
Uses	• Make an API call to fetch data • Set a timer • Update the DOM • Clean up state after a component unmounts	• Make an API call to fetch data • Set a timer • Update the DOM • Clean up state after a component unmounts

4. What is Render ()

Basically Condition Rendering means providing **Conditions in render functions.** conditional rendering is **the process of displaying different content based on certain conditions or states**. It allows you to create dynamic user interfaces that can adapt to changes in data and user interactions. In this process, you can use conditional statements to decide what content should be rendered. Rendering is **React's process of describing a user interface based on the application's current state and props**. The initial render in a React app is the first render when the application starts up, while re-rendering occurs when there is a change in the state to figure out which parts of the UI need an update.

5. Describe State

The state is **a built-in React object that is used to contain data or information about the component**. A component's state can change over time; whenever it changes, the component re-renders. state is a part of react JS 's Functional Components which used for handling form data. State in React is an object that contains data that is specific to a component. State can be used to track changes in data over time, and it can be used to update the component's render based on those changes. State is often used to manage user input, and it can also be used to store data that is retrieved from an API.

To define state in a React component, you can use the useState hook. The useState hook takes an initial value as its argument, and it returns an array of two values. The first value is the current state of the component, and the second value is a function that can be used to update the state.Generally, class components use the state property, while the new function component use hooks to manage the state. Learning state management in class components will help you troubleshoot state management based code and give you an understanding of when we have to make use of class components or functional components.

6. What is API?

API means **Application Programming Interface** it's a **mediator** which make a connection between clint and server. Api used for getting data from server. n API in React JS is a way for your React application to interact with other systems and exchange information with them. APIs are often used to fetch data from a server or to send data to a server. There are many different ways to make API calls in React. One common way is to use the Fetch API. The Fetch API is a built-in JavaScript API for retrieving resources from a server or an API endpoint.

To make an API call with the Fetch API, you first need to create a new fetch object. The fetch object takes the URL of the resource you want to fetch as its argument. Once you have created a fetch object, you can call the then() method on it. The then() method takes a callback function as its argument. The callback function will be executed when the fetch request is complete. The then() method returns a Promise object. A Promise object represents the eventual completion (or failure) of an asynchronous operation.

If the fetch request is successful, the callback function will be passed a Response object. The Response object contains the response data from the server. If the fetch request fails, the callback function will be passed an Error object.

7. Describe useReducer with Example.

The useReducer Hook is a built-in React Hook that allows you to manage state in your components. It is similar to the useState Hook, but it provides a number of advantages, such as:

- It allows you to encapsulate your state logic in a reducer function.
- It makes it easier to handle complex state updates.
- It can help you to improve the performance of your application.

The useReducer Hook takes two arguments:

- A reducer function: This is a function that takes the current state and an action object as its arguments and returns a new state.
- An initial state: This is the initial state of your component.

Here is an example of how to use the useReducer

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

const reducer = (state, action) => {
  switch (action.type) {
    case 'INCREMENT':
      return { count: state.count + 1 };
    case 'DECREMENT':
      return { count: state.count - 1 };
    default:
      return state;
  }
};

const initialState = { count: 0 };

const Counter = () => {
  const [state, dispatch] = useReducer(reducer, initialState);

  return (
    <div>
      <h1>{state.count}</h1>
      <button onClick={() => dispatch({ type: 'INCREMENT' })}>Increment</button>
      <button onClick={() => dispatch({ type: 'DECREMENT' })}>Decrement</button>
    </div>
  );
};

export default Counter;
```

8. Write a Query solution of useContext.

The useContext Hook allows you to consume context values from a parent component without having to pass them down through the component tree explicitly. This can be useful for avoiding prop drilling and for sharing values between components that are not directly related to each other.

To use the useContext Hook, you first need to create a context object using the createContext() function. The createContext() function takes a default value as its argument and returns an object with two properties: Provider and Consumer.

Here is an example of how to use the useContext Hook to share the results of a query between different components:

```
import React, { createContext, useContext } from 'react';

const QueryContext = createContext(null);

function QueryProvider({ children }) {
 const [queryResults, setQueryResults] = useState(null);

 // Fetch data from an API
 async function fetchQueryResults() {
  const results = await fetch('https://api.example.com/data');
  setQueryResults(results);
 }

 // Provide the query results to the context
 return (
  <QueryContext.Provider value={queryResults}>
   {children}
  </QueryContext.Provider>
 );
}

function ComponentA() {
 const queryResults = useContext(QueryContext);

 // Display the query results
 return (
  <div>
   <h1>Query Results</h1>
   {queryResults && queryResults.map((result) => (
    <p key={result.id}>{result.name}</p>
   ))}
  </div>
 );
}

function ComponentB() {
 const queryResults = useContext(QueryContext);

 // Display the query results
 return (
  <div>
   <h1>Query Results</h1>
   {queryResults && queryResults.map((result) => (
    <p key={result.id}>{result.name}</p>
   ))}
  </div>
 );
}

function App() {
 return (
  <QueryProvider>
   <ComponentA />
   <ComponentB />
  </QueryProvider>
 );
}
```

9. What is components in React?

Components are the building blocks of React applications. They are reusable pieces of code that can be combined to create complex user interfaces. Components can be nested inside of other components, and they can be reused throughout an application.

Components are typically defined as JavaScript functions that return React elements. React elements are lightweight objects that describe what should be rendered on the screen.

Components can be created using two different approaches:

- **Class components:** Class components are created using the ES6 class syntax. They are more verbose than function components, but they offer more features, such as state and lifecycle hooks.
- **Function components:** Function components are created using JavaScript functions. They are less verbose than class components, but they do not offer state or lifecycle hooks.

Components can also be classified into two different types:

- **Presentation components:** Presentation components are responsible for rendering the UI. They do not manage state or perform any other side effects.
- **Container components:** Container components manage state and perform side effects. They often wrap presentation components and provide them with the data they need to render.

Components are a powerful tool that can be used to create complex and reusable user interfaces in React. By understanding how to use components, you can write more efficient and maintainable code.

10. Write a Code of useRef.

JavaScript

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

function MyComponent() {
 const inputRef = useRef(null);
 const textAreaRef = useRef(null);

 const handleSubmit = () => {
  const inputValue = inputRef.current.value;
  const textAreaValue = textAreaRef.current.value;

  console.log(inputValue, textAreaValue);
 };

 return (
  <div>
   <input ref={inputRef} />
   <textarea ref={textAreaRef} />
   <button onClick={handleSubmit}>Submit</button>
  </div>
 );
}
```

Here is a breakdown of the code:

- useRef is a React hook that allows you to create a mutable ref object.
- inputRef.current and textAreaRef.current are the current values of the ref objects. In this case, they are the input element and the text area element, respectively.
- value is a property on the input element and the text area element that gets or sets the value of the element.
- handleSubmit() is a function that is called when the button is clicked. This function gets the values of the input and text area elements and logs them to the console.

When the button is clicked, the handleSubmit() function is called. This function gets the values of the input and text area elements and logs them to the console.

11. what is useLayout?

useLayoutEffect is a React Hook that allows you to perform side effects after the DOM has been updated, but before the browser has had a chance to paint. This can be useful for tasks like measuring the layout of the DOM or animating elements.

useLayoutEffect is similar to useEffect, but it runs synchronously after all DOM mutations. This means that it will block the browser from repainting the screen until it has finished running.

useLayoutEffect is a powerful tool, but it should be used with caution. Because it blocks the browser from repainting the screen, it can cause performance problems if not used correctly.

Here are some examples of situations where you might want to use useLayoutEffect:

- To measure the layout of the DOM before animating elements.
- To scroll to a specific element after it has been rendered.
- To focus an input element after it has been rendered.
- To update the position of a fixed element after the DOM has been updated.

12. what is function describe any 4 function of js

A function in JavaScript is a block of code that can be reused and passed around as an argument to other functions. Functions can take parameters and return values. A function in JavaScript is similar to a procedure—**a set of statements that performs a task or calculates a value**, but for a procedure to qualify as a function, it should take some input and return an output where there is some obvious relationship between the input and the output.

Here are four common functions in JavaScript:

1. **alert():** This function displays an alert box with the specified message.
2. **console.log():** This function prints the specified message to the console.
3. **prompt():** This function displays a dialog box with the specified message and prompts the user for input.
4. **confirm():** This function displays a dialog box with the specified message and asks the user to confirm or cancel.

Here are some examples of how to use these functions:

JavaScript

```
// alert():
alert("Hello, world!");

// console.log():
console.log("This is a message to the console.");

// prompt():
const name = prompt("What is your name?");

// confirm():
const confirmed = confirm("Are you sure?");
```

13. What is SPA ? and Describe it.

A single-page application (SPA) is a web application that loads a single HTML page and dynamically updates that page as the user interacts with the application. This approach makes the application more user-friendly and efficient, as users do not have to wait for new pages to load.

SPAs are often used for complex web applications, such as social media applications, email applications, and online banking applications. They can also be used for simpler web applications, such as landing pages and product pages.

React is a JavaScript library that is well-suited for building SPAs. React allows you to create reusable components that can be dynamically updated as the user interacts with the application.

Here are some of the benefits of using React to build SPAs:

- React is easy to learn and use.
- React is component-based, which makes it easy to create reusable code.
- React is fast and efficient.
- React is well-supported by the community.

Here are some examples of popular SPAs built with React:

- Facebook
- Instagram
- Airbnb
- Netflix
- Reddit

14. What is useCallback?

useCallback is a React hook that allows you to memoize a callback function. Memoization is a technique used to store the results of function calls so that they can be reused later without having to recalculate them.

useCallback is useful for preventing unnecessary re-renders of your components. When you pass a callback function to a child component, React will recreate the function each time the parent component re-renders. This can cause performance problems if the callback function is expensive to create.

useCallback can be used to memoize the callback function, which will prevent React from recreating it each time the parent component re-renders. This can improve the performance of your application by reducing the number of re-renders that occur. his can improve the performance of our application because the handleClick function is expensive to create. It has to calculate the new value of the count variable.

useCallback is a powerful tool that can be used to improve the performance of your React applications. By memoizing callback functions, you can prevent unnecessary re-renders and make your applications more responsive and efficient.

15. Give the definitions of hooks.

- **useState:** Allows you to manage state in your components.
- **useEffect:** Allows you to run side effects after a component has rendered or updated.
- **useContext:** Allows you to consume context values from a parent component without having to pass them down through the component tree explicitly.
- **useMemo:** Allows you to memoize a value so that it is only recalculated when its dependencies change.
- **useCallback:** Allows you to memoize a callback function so that it is only recreated when its dependencies change.
- **useReducer:** Allows you to manage state in your components using a reducer function.
- **useRef:** Allows you to create a ref to an element or object.
- **useLayoutEffect:** Allows you to run side effects after the DOM has been updated, but before the browser has had a chance to paint.
- **useImperativeHandle**: Allows you to create a ref that can be used to access and modify the imperatives of a component.
- **useDebugValue**: Allows you to display a custom debug value in React's Developer Tools.
- **useTransition**: Allows you to manage the visibility of components in an animated way.
- **useDeferredValue**: Allows you to delay the rendering of a component until its value has been resolved.
- **useMemoCache**: Allows you to create a cache of memoized values.
- **useOpaqueIdentifier**: Allows you to create a unique identifier for a component that is stable across re-renders.
- **useMutableSource**: Allows you to create a mutable source of data that can be used to update multiple components.
- **useSyncExternalStore**: Allows you to synchronize the state of your React component with an external store.
- **useId**: Allows you to generate a unique ID for a component that is stable across re-renders.
- **useInsertionEffect**: Allows you to run side effects after a component has been inserted into the DOM.
- **useIsomorphicLayoutEffect**: Allows you to run side effects after the DOM has been updated, but before the browser has had a chance to paint.