Q: How to create nested routes? Write react-router-dom configuration.
A:

```
const appRoute=[
        {
            path:"/",
            element:<Body />,
            errorElement: <RoutesError />,
            children: [{
                    path:"about",
                    element: <About />,
                    children: [{
                        path: "profile",
                        element:< Profile />
                        }]
                }]
        }]
```

To create nested routes, we write the path config inside the children element of the parent("about") config. This is the config for route- "/about/profile"

Q: createHashRouter, createMemoryRouter.
A: **createHashRouter:** This is used when we are not able to configure our web server to direct all the traffic to our react router application. Instead of using normal url, it used hash portion of the url. Other than this, it behaves same way as createBrowserRouter. In cases, where we shouldn't or cannot sent the url to the server due to some reasons like in case of shared hoisting scenario where we do not have full control over the server then the hash router stores our current location in hash portion of the url so that its not sent to the server.

createMemoryRouter: Memory Router maintains the current location internally in an array by having its own history stack. Unlike <BrowserHistory> and <HashHistory> it is not tied to external source, like history stack of browser. It is primarily used for testing and component development tools like Storybook but can also be used for running React Router in non-browser environments.

Q: What is the order of lifecycle method calls in Class-based components?
A:

- **Mounting Phase:** In Mounting phase, when a component is rendered for the first time then:
    1. constructor()
    2. render()
    3. DOM elements are updated.
    4. componentDidMount().
- **Updating Phase:** In updating phase when the state or props changes then:
    1. render()
    2. DOM elements updated by React.
    3. componentDidUpdate() called
- **Unmounting Phase:** When component is about to be unloaded from browser then:
    1. componentWillUnmount() called

Q: Why do we use componentDidMount() ?
A: This method is used to perform side-effects like accessing DOM nodes, setting up subscriptions or fetching data through an API. This method is used after the render() is called and after the component is mounted on DOM.

Q: Why do we use componentWillUnmount() ?
A: This method is invoked right before the component is unmounted(unloaded) from the DOM. Any subscriptions that we set up when the component was mounted must be unsubscribed here as in SPA the subscriptions are not removed by themself when we change component. Any timer that was set up in mounting phase must also be removed else it will keep on running even after we have unmounted the component. Not removing subscription, timer, API call or any such code may impact the performance of the application.

Q: Why do we use super(props) in constructor?
A: super() is used to call the constructor() of parent class i.e, React.Component. It gives us access to the variables defined inside React.Component constructor(). Without calling super() inside constructor() we cannot get access to ***this***. As **this** gets accessed after calling super(), state can only be initialized after calling super() also.
We pass props to super method (super(props)) so that props can get passed to the constructor function of React.Component. This allows us to use this.props inside constructor() of child component and we can also use methods inside constructor() of child component that needs props inside the method without having to explicitly pass props as a parameter to the method.
Ex:
```

```
export default class ProfileClass extends Component {
 constructor(props) {
   super(props);
   console.log(this.props); //shows props object
   this.showDetails(); //Can use “this” bcoz of calling super(props)
 }
 showDetails() {
   console.log(this.props.name + ":" + this.props.country); //Can use this.props
without passing props as parameter to method.
 }
 render() {
   return <h1>Profile: Class-based component</h1>;
 }
}
```

```

**Q: Why can’t we have the callback function of useEffect async?**
A: ```
```
useEffect(async () => {
   console.log("working");
      return () => {
```

```
    console.log("Unmounting");
  };

}, []);
```

Using async with the callback function consoles “working” but also throws error to not use async with the callback function. This is because useEffect expects the callback function to return another function(cleanup function) or nothing but when we use async in front of the callback function its going to always return a promise which is not expected from it. The cleanup function expected to run before component unmount also never gets called.