Institute for Advanced Computing And Software Development (IACSD) Akurdi, Pune

React JS

Dr. D.Y. Patil Educational Complex, Sector 29, Behind Akurdi Railway Station,

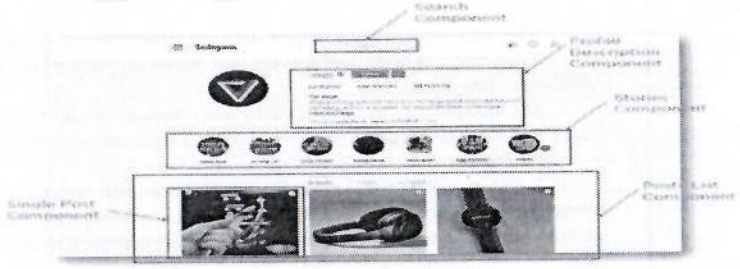
Nigdi Pradhikaran, Akurdi, Pune - 411044.

What is ReactJs

- React is a JavaScript library created for building fast and interactiveuser interfaces for web and mobile applications.
- It is an open-source, component-based, front-end library responsible only for the application's view layer.
- In Model View Controller (MVC) architecture, the view layer isresponsible for how the app looks and feels.
- React was created by Jordan Walke, a software engineer at Facebook.
- model-view
- Fig: MVC architecture

Instagram webpage example

- Instagram webpage, entirely built using React,
- To get a better understanding of how React works.
- As the illustration shows, React divides the UI into multiple components, which makes the code easier to debug. This way, each component has its property andfunction.
- instagram-components
- Fig: Instagram Components
- Now that we know what React is let's move on and see why React is the most popular front-end library for web application development.
- React divides the UI into multiple components, which makes the codeeasier to debug. This way, each component has its property and function.



WhyReactJS

- Easy creation of dynamic applications:
- Improved performance:
- Reusable components:
- Unidirectional data flow:
- Small learning curve:
- It can be used for the development of both web and mobile apps:
- Dedicated tools for easy debugging:

Features of React

- JSX - JavaScript Syntax Extension

- JSX is a syntax extension to JavaScript. It is used with React to describe what the user interface should look like. By using JSX, we can write HTML structures in the same file that contains JavaScript code. This makes the code easier to understand and debug, as it avoids the usage of complex JavaScript DOM structures.

Example

```
const name = 'IACSD;

const message = <h1>Hello, {name}</h1>;
```

- The above code shows how JSX is implemented in React. It is neither a string nor HTML. Instead, it embeds HTML into JavaScript code.

Virtual DOM

- React keeps a lightweight representation of the "real" DOM in the memory, and that is known as the "virtual" DOM (VDOM).
- Manipulating real DOM is much slower than manipulating VDOM because nothing gets drawn on the screen. When the state of an object changes, VDOM changes only that object in the real DOM instead of updating all of the objects.

- How do Virtual DOM and React DOM interact with each other?

- When the state of an object changes in a React application, VDOM gets updated.
- It then compares its previous state and then updates only those objects in the real DOM instead of updating all of the objects.
- This makes things move fast, especially when compared to other front-end technologies that have to update each object even if only a single object changes in the web application.

- Performance

- React uses VDOM, which makes the web applications run much faster than those developed with alternate front-end frameworks. React breaks a complex user interface into individual components, allowing multiple users to work on each component simultaneously, thereby speeding up the development time.

ReactJs is front end library used to design B2C type applications. The application is SPA(Single page Application)

It uses javascript with JSX (javascript extension). MVC

pattern we use(model, view, controller) Component

---------- smallest unit of react program

To use react we need to install

1. Nodejs and npm
2. React and react DOM
3. Webpack
4. Babel

To istall all these we use a cli tool creat-react-appnpm -g

install create-react-app

To create a new application, the name of the application should be in small case letters only.

Create-react-app myfirstapp

To create react application without installing creat-react-app, but then network should be availablenpx

create-react-app myapp

user defined components are of 2 types

1. Class component-- slower and heavier
2. Functional component faster and light weight

- Extensions

- React goes beyond simple UI design and has many extensions that offer complete application architecture support. It provides server-side rendering, which entails rendering a normally client-side only web application on the server, and then sends a fully rendered page to the client.
- It also uses Flux and Redux extensively in web application development.
- React Native, is a popular framework derived from React, used to create cross-compatible mobile applications.

- One-way Data Binding
- In a React app, child components are nested within parent components. It gives better control of the whole web application, helpful to find the errors.

- Debugging
- React applications are easy to test due to a large developer community.
- Facebook even provides a small browser extension that makes React debugging faster and easier.

- example,
- adds a React tab in the developer tools option within the Chrome web browser.
- The tab makes it easy to inspect React components directly.

concepts.
- Components
- State
- Piops

Components
- Components aie the building blocks of any React application,
- single app usually consists of multiple components.
- A component is essentially a poition of the usei inteiface.
- React splits the UI into independent, ieusable paits that can be piocessed sepaiately.
- Theie aie two types of componenn React:

1. Functional Component (Stateless Component)
2. Class Component (stateful component)

Stateless Component
- These components are simply javascript functions.
- We can create a functional component in React by writing a javascript function.
- These functions may or may not receive data as parameters

```
const ContactList=()=>
{

return <h1>Welcome Message!</h1>;
}
export default ContactList;
```

Stateful Component
- The class components are a little more complex than the functional components.
- The functional components are not aware of the other components in your program whereas the class components can work with each other.
- We can pass data from one class component to other class components.
- We can use JavaScript ES6 classes to create class-based components in React. Below example shows a valid class-based component in React:


```
class AddContactComponent extends React.Component
{
state={}
render(){
return <h1>Welcome Message!</h1>;
}
}
export default AddContactComponent
```

When would you use a stateless component?
- When you just need to present the props
- When you don't need a state, or any internal variables
- When creating element does not need to be interactive
- When you want reusable code.

When would you use a stateful component?
- When building element that accepts user input
- ..or element that is interactive on page
- When dependent on state for rendering, such as, fetching data before rendering
- When dependent on any data that cannot be passed down as props

Stateful Component Vs Stateless component

- 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.
- The change in state can happen as a response to user action or system-generated events, and these changes determine the behavior of the component and how it will render.

```
class AddContact extendsReact.Component {
   state = {
      id: "",
      name:"",
      email:"",
      mobile:""
   };
  updateName() {
      this.setState({ name: "Rajan" });
   }
render() {
                return(
                  <div>
                  {this.state.name}

       )
   }
}
```

Props

- Props are short for properties.
- It is a React built-in object which stores the value of a tag's attributes and works similar to the HTML attributes.
- It provides a way to pass data from one component to other components

Props Vs State

Props	State
Props are read-only.	State changes can be asynchronous.
Props are immutable.	State is mutable.
Props allow you to pass data from one component to other components as an argument.	State holds information about the components.
Props can be accessed by the child component.	State cannot be accessed by child components.
Props are used to communicate between components.	States can be used for rendering dynamic changes with the component.
Stateless component can have Props.	Stateless components cannot have State.
Props make components reusable.	State cannot make components reusable.
Props are external and controlled by whatever renders the component.	The State is internal and controlled by the React Component itself.

	Props	State
Can get initial value from parent Component?	Yes	Yes
Can be changed by parent Component?	Yes	No
Can set default values inside Component?	Yes	Yes
Can change inside Component?	No	Yes
Can set initial value for child Components?	Yes	Yes
Can change in child Components?	Yes	No

How to use map in react

```
function App() {
 // let person={id:11,name:"Kishori",desg:"manager"};
 let per_arr=[
  {id:11,name:"John",desg:"designer"},
  {id:12,name:"Kishori",desg:"manager"},
  {id:13,name:"Jack",desg:"programmer"}
 ];
 return (
   <div className="App">
    <h1>Person Details</h1>
    <div>
     <table border="2">
      <thead>
      <tr><th>Person Id</th>
      <th>Name</th>
      <th>Designation</th></tr>
      </thead>
<tbody>
      {per_arr.map((p)=>{
       return (<tr key={p.id}><td>{p.id}</td>
       <td>{p.name}</td>
       <td>{p.desg}</td>
       </tr>);
      })}
      </tbody>
     </table>
    </div>
   </div>
 );
}
```

What is key property while using map?

- Every React application displays an array list of some kind using the method map.
- React tells us that for each element of that list that we return for rendering, we must provide a unique key prop.

- But do you know why it's necessary? Why does React need this key prop?
  Keys help React identify which items have changed, are added, or are removed. Keys should be given to the elements inside the array to give the elements a stable identity.
- When the state of your component changes, the render function will return a new tree of React elements, different to the previous/current one.
- React needs to figure out what are the differences to efficiently update the UI with the most recent changes.
- This process of matching both element trees is called reconciliation.

React hooks

- useSate
- In functional component if local variable value changes the component will not get retendered.
- So to keep check on change of the value of the variable we need to consider those variables as state variables
- So we use useState hook
- It can be used directly inside functional component. It cannot be called outside functional component
- Also cannot be called in functions which are inside functional component

  - useState is a function of React library
  - It creates a special variable and also returns a function const Message= () => {

```
const [messageState,setMessageSate] = useState( '' ); // use state
assigns blank to messageState
const listState = useState( [] );
}
```

  - The component in which we call the setMessageState function will schedule the change in messageState and it will also rerender the component in which it is called.

1. What is React?

React is an open-source front-end JavaScript library that is used for building user interfaces, especially for single-page applications. It is used for handling view layer for web and mobile apps. React was created by Jordan Walke, a software engineer working for Facebook. React was first deployed on Facebook's News Feed in 2011 and on Instagram in 2012.

2. What are the major features of React?

The major features of React are:

- Uses JSX syntax, a syntax extension of JS that allows developers to write HTML in their JS code.
- It uses VirtualDOM instead of 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.

3. What is JSX?

JSX is a XML-like syntax extension to ECMAScript (the acronym stands for JavaScript XML). Basically it just provides syntactic sugar for
the React.createElement() function, giving us expressiveness of JavaScript along with HTML like template syntax.
In the example below text inside <h1> tag is returned as JavaScript function to the render function.

```
export default function App() { return (
<div>
<h1>{"Welcome to React world!"}</h1>
</div>
);
}
```

See Class

Note: JSX is stricter than HTML

4. What is the difference between Element and Component?

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 React.createElement() function returns an object:

```
{
```



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

And finally it renders to the DOM using ReactDOM.render():

```
<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 or it can be defined as a function. In either case, it takes props as an input, and returns a JSX tree as the output:

```
const Button = ({ onLogin }) => (
<div id={"login-btn"} onClick={onLogin}> Login
</div>
);
```

Then JSX gets transpiled to a React.createElement() function tree:

```
const Button = ({ onLogin }) => React.createElement(
"div",
{ id: "login-btn", onClick: onLogin }, "Login"
);
```

5. How to create components in React?

There are two possible ways to create a component.

i. Function Components: This is the simplest way to create a component. Those are pure JavaScript functions that accept props object as the first parameter and return React elements:

```
ii.  function Greeting({ message }) {
iii. return <h1>{`Hello, ${message}`}</h1>;
}
```

iv. Class Components: You can also use ES6 class to define a component. The above function component can be written as:

```
v.   class Greeting extends React.Component {
vi.  render() {
vii. return <h1>{`Hello, ${this.props.message}`}</h1>;
viii. }
}
```

6. When to use a Class Component over a Function Component?

If the component needs state or lifecycle methods then use class component otherwise use function component.
However, from React 16.8 with the addition of Hooks, you could use state , lifecycle methods and other features that were only available in class component right in your function component. So, it is always recommended to use Function components, unless you need a React functionality whose Function

component equivalent is not present yet, like Error Boundaries.

7. What are Pure Components?

React.PureComponent is exactly the same as React.Component except that it handles the shouldComponentUpdate() method for you. When props or state changes, PureComponent will do a shallow comparison on both props and state. Component on the other hand won't compare current props and state to next out of the box. Thus, the component will re-render by default whenever shouldComponentUpdate is called. In functional components we use React.memo() API. React.memo() is a higher-order component. It takes a React component as its first argument and returns a special type of React component that allows the renderer to render the component while memoizing the output.

Therefore, if the component's props are shallowly equal, the React.memo() component will bail out the updates.

8. What is state in React?

State of a component is an object that holds some information that may change over the lifetime of the component. We should always try to make our state as simple as possible and minimize the number of stateful components.

Let's create a user component with message state,

```
import React, { useState } from "react"; function User() {
const [message, setMessage] = useState("Welcome to React world");

return (
<div>
<h1>{message}</h1>
</div>
);
}
```

9. What are props in React?

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

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

. Pass custom data to your component.
i. Trigger state changes.
ii. 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) 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

Example: Props in Class Based Component

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

class ChildComponent extends React.Component { render() {
return (
<div>
<p>{this.props.name}</p>
<p>{this.props.age}</p>
</div>
);
}
}

class ParentComponent extends React.Component { render() {
return (
<div>
<ChildComponent name="John" age="30" />
<ChildComponent name="Mary" age="25" />
</div>
);
}
}
```

Example: Props in Functional Component

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

const ChildComponent = (props) => { return (
<div>
<p>{props.name}</p>
<p>{props.age}</p>
</div>
);
};

const ParentComponent = () => { return (
<div>
<ChildComponent name="John" age="30" />
<ChildComponent name="Mary" age="25" />

</div>
);
};
```

10. What is the difference between state and props?

Both props and state are plain JavaScript objects. While both of them hold information that influences the output of render, they are different in their functionality with respect to component. Props get passed to the component similar to function parameters whereas state is managed within the component similar to variables declared within a function.

11. Why should we not update the state directly?

If you try to update the 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 state changes, the component responds by re-rendering.

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

Note: You can directly assign to the state object either in constructor or using latest javascript's class field declaration syntax.

12. What is the purpose of callback function as an argument of setState()?

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 lifecycle method rather than this callback function.

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

13. What is the difference between HTML and React event handling?

Below are some of the main differences between HTML and React event handling,

. In HTML, the event name usually represents in lowercase as a convention:

```
<button onclick="activateLasers()"></button>
```

Whereas in React it follows camelCase convention:

```
<button onClick={activateLasers}>
```

i. In HTML, you can return false to prevent default behavior:

```
ii. <a
iii. href="#"
iv. onclick='console.log("The link was clicked."); return false;'
/>
```

Whereas in React you must call preventDefault() explicitly:


```
function handleClick(event) { event.preventDefault(); console.log("The link was clicked.");
}
```

v. In HTML, you need to invoke the function by appending () Whereas in react you should not append () with the function name. (refer "activateLasers" function in the first point for example)

14. How to bind methods or event handlers in JSX callbacks?

There are 3 possible ways to achieve this:

. Binding in Constructor: In JavaScript classes, the methods are not bound by default. The same thing applies for React event handlers defined as class methods. Normally we bind them in constructor.

```
i. class Foo extends Component {
ii. constructor(props) {
iii. super(props);
iv. this.handleClick = this.handleClick.bind(this);
v. }
vi. handleClick() {
vii. console.log("Click happened");
viii. }

ix. render() {
x. return <button onClick={this.handleClick}>Click Me</button>;
xi. }
}
```

xii. Public class fields syntax: If you don't like to use bind approach then public class fields syntax can be used to correctly bind callbacks.

```
xiii. handleClick = () => {
xiv. console.log("this is:", this);
};
<button onClick={this.handleClick}>{"Click me"}</button>
```

xv. Arrow functions in callbacks: You can use arrow functions directly in the callbacks.

```
xvi. handleClick() {
xvii. console.log('Click happened');
xviii. }
xix. render() {
xx. return <button onClick={() => this.handleClick()}>Click Me</button>;
}
```

Note: If the callback is passed as prop to child components, those components might do an extra re-rendering. In those cases, it is preferred to go with .bind() or public class fields syntax approach considering performance.

15. How to pass a parameter to an event handler or callback?

You can use an arrow function to wrap around an event handler and pass parameters:

```
<button onClick={() => this.handleClick(id)} />
```

This is an equivalent to calling .bind:

<button onClick={this.handleClick.bind(this, id)} />
Apart from these two approaches, you can also pass arguments to a function which is defined as arrow function
<button onClick={this.handleClick(id)} />; handleClick = (id) => () => {
console.log("Hello, your ticket number is", id);
};

16. What are synthetic events in React?
SyntheticEvent is a cross-browser wrapper around the browser's native event. Its API is same as the browser's native event,
including stopPropagation() and preventDefault(), except the events work identically across all browsers.

17. What are inline conditional expressions?
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 &&.
<h1>Hello!</h1>;
{
messages.length > 0 && !isLogin ? (
<h2>You have {messages.length} unread messages.</h2>
) : (
<h2>You don't have unread messages.</h2>
);
}

18. What is "key" prop and what is the benefit of using it in arrays of elements?
A key is a special string attribute you should include when creating arrays of elements. Key prop helps React identify which items have changed, are added, or are removed.
Keys should be unique among its siblings. Most often we use ID from our data as key:
const todoItems = todos.map((todo) => <li key={todo.id}>{todo.text});
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}
));
Note:

. Using indexes for keys is not recommended if the order of items may change. This can negatively impact performance and may cause issues with component state.
i. If you extract list item as separate component then apply keys on list component instead of li tag.
ii. There will be a warning message in the console if the key prop is not present on list items.

19. What is the use of refs?
The ref is used to return a reference to the element. They should be avoided in most cases, however, they can be useful when you need a direct access to the DOM element or an instance of a component.

20. How to create refs?
There are two approaches
. This is a recently added approach. Refs are created
using React.createRef() method and attached to React elements via the ref attribute. In order to use refs throughout the component, just assign the ref to the instance property within constructor.
i. class MyComponent extends React.Component {
ii. constructor(props) {
iii. super(props);
iv. this.myRef = React.createRef();
v. }
vi. render() {
vii. return ;
viii. }
}
ix. You can also use ref callbacks approach regardless of React version. For example, the search bar component's input element is accessed as follows,
x. class SearchBar extends Component {
xi. constructor(props) {
xii. super(props);
xiii. this.txtSearch = null;
xiv. this.state = { term: "" };
xv. this.setInputSearchRef = (e) => {
xvi. this.txtSearch = e;
xvii. };

xviii. }
xix. onInputChange(event) {
xx. this.setState({ term: this.txtSearch.value });
xxi. }
xxii. render() {
xxiii. return (
xxiv. <input
xxv. value={this.state.term}
xxvi. onChange={this.onInputChange.bind(this)}
xxvii. ref={this.setInputSearchRef}
xxviii. />

```
xxix.  );
xxx.   }
}
```

You can also use refs in function components using closures. Note: You can also use inline ref callbacks even though it is not a recommended approach.

21. What are forward refs?

Ref forwarding is a feature that lets some components take a ref they receive, and pass it further down to a child.

```
const ButtonElement = React.forwardRef((props, ref) => (
<button ref={ref} className="CustomButton">
{props.children}
</button>
));

// Create ref to the DOM button:
const ref = React.createRef();
<ButtonElement ref={ref}>{"Forward Ref"}</ButtonElement>;
```

22. Which is preferred option with in callback refs and findDOMNode()?

It is preferred to use callback refs over findDOMNode() API.

Because findDOMNode() prevents certain improvements in React in the future. The legacy approach of using findDOMNode:

```
class MyComponent extends Component { componentDidMount() {
findDOMNode(this).scrollIntoView();
}

render() { return <div />;

}
}
```

The recommended approach is:

```
class MyComponent extends Component { constructor(props) {
super(props);
this.node = createRef();
}
componentDidMount() { this.node.current.scrollIntoView();
}

render() {
return <div ref={this.node} />;
```



```
}
}
```

23. Why are String Refs legacy?

If you worked with React before, you might be familiar with an older API where the ref attribute is a string, like ref={'textInput'}, and the DOM node is accessed as this.refs.textInput. We advise against it because string refs have below issues, and are considered legacy. String refs were removed in React v16.

. They force React to keep track of currently executing component. This is problematic because it makes react module stateful, and thus causes weird errors when react module is duplicated in the bundle.

i. They are not composable — if a library puts a ref on the passed child, the user can't put another ref on it. Callback refs are perfectly composable.

ii. They don't work with static analysis like Flow. Flow can't guess the magic that framework does to make the string ref appear on this.refs, as well as its type (which could be different). Callback refs are friendlier to static analysis.

iii. It doesn't work as most people would expect with the "render callback" pattern (e.g.)

```
iv.   class MyComponent extends Component {
v.    renderRow = (index) => {
vi.           // This won't work. Ref will get attached to DataTable rather than MyComponent:
vii.  return <input ref={"input-" + index} />; viii.
ix.   // This would work though! Callback refs are awesome.
x.    return <input ref={(input) => (this["input-" + index] = input)} />;

xi.   };
xii.
xiii. render() {
xiv.  return (
xv.   <DataTable data={this.props.data} renderRow={this.renderRow} />
xvi.  );
xvii. }
}
```

24. What is Virtual DOM?

The Virtual DOM (VDOM) is an in-memory representation of Real DOM. The representation of a UI is kept in memory and synced with the "real" DOM. It's a step that happens between the render function being called and the displaying of elements on the screen. This entire process is called reconciliation.

26. What is the difference between Shadow DOM and Virtual DOM?

The Shadow DOM is a browser technology designed primarily for scoping variables and CSS in web components. The Virtual DOM is a concept implemented by libraries in JavaScript on top of browser APIs.

27. What is React Fiber?

Fiber is the new reconciliation engine or reimplementation of core algorithm in React v16. The goal of React Fiber is to increase its suitability for areas like animation, layout, gestures, ability to pause, abort, or reuse work and assign priority to different types of updates; and new concurrency primitives.

28. What is the main goal of React Fiber?

The goal of React Fiber is to increase its suitability for areas like animation, layout, and gestures. Its headline feature is incremental rendering: the ability to split rendering work into chunks and spread it out over multiple frames.
from documentation
Its main goals are:

. Ability to split interruptible work in chunks.
i. Ability to prioritize, rebase and reuse work in progress.
ii. Ability to yield back and forth between parents and children to support layout in React.
iii. Ability to return multiple elements from render().
iv. Better support for error boundaries.

29. What are controlled components?

A component that controls the input elements within the forms on subsequent user input is called Controlled Component, i.e, every state mutation will have an associated handler function.
For example, to write all the names in uppercase letters, we use handleChange as below,

```
handleChange(event) {
this.setState({value: event.target.value.toUpperCase()})
}
```

30. What are uncontrolled components?

The Uncontrolled Components are the ones that store their 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.
In the below UserProfile component, the name input is accessed using ref.

```
class UserProfile extends React.Component { constructor(props) {
super(props);
this.handleSubmit = this.handleSubmit.bind(this); this.input = React.createRef();
}

handleSubmit(event) {
alert("A name was submitted: " + this.input.current.value); event.preventDefault();
```



```
}

render() { return (
<form onSubmit={this.handleSubmit}>
<label>
{"Name:"}
<input type="text" ref={this.input} />
</label>
<input type="submit" value="Submit" />
</form>
);
}
}
```

In most cases, it's recommend to use controlled components to implement forms. In a controlled component, form data is handled by a React component. The alternative is uncontrolled components, where form data is handled by the DOM itself.

31. What is the difference between createElement and cloneElement?

JSX elements will be transpiled to React.createElement() functions to create React elements which are going to be used for the object representation of UI. Whereas cloneElement is used to clone an element and pass it new props.

32. What is Lifting State Up in React?

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. That means if two child components share the same data from its parent, then move the state to parent instead of maintaining local state in both of the child components.

33. What are the different phases of component lifecycle?

The component lifecycle has three distinct lifecycle phases:
. Mounting: The component is ready to mount in the browser DOM. This phase covers initialization from constructor(), getDerivedStateFromProps(), render(), and componentDidMount() lifecycle methods.
i. Updating: In this phase, the component gets updated in two ways, sending the new props and updating the state either
from setState() or forceUpdate(). This phase
covers getDerivedStateFromProps(), shouldComponentUpdate(), render(), getSn apshotBeforeUpdate() and componentDidUpdate() lifecycle methods.
ii. Unmounting: In this last phase, the component is not needed and gets unmounted from the browser DOM. This phase
includes componentWillUnmount() lifecycle method.
It's worth mentioning that React internally has a concept of phases when applying changes to the DOM. They are separated as follows
iii. Render The component will render without any side effects. This applies to Pure components and in this phase, React can pause, abort, or restart the render.
iv. Pre-commit Before the component actually applies the changes to the DOM, there is a moment that allows React to read from the DOM through the getSnapshotBeforeUpdate().

v. Commit React works with the DOM and executes the final lifecycles respectively componentDidMount() for mounting, componentDidUpdate() for updating, and componentWillUnmount() for unmounting.
React 16.3+ Phases (or an interactive version)

Before React 16.3

34. What are the lifecycle methods of React?
Before React 16.3
o componentWillMount: Executed before rendering and is used for App level configuration in your root component.
o componentDidMount: Executed after first rendering and here all AJAX requests, DOM or state updates, and set up event listeners should occur.
o componentWillReceiveProps: Executed when particular prop updates to trigger state transitions.
o shouldComponentUpdate: Determines if the component will be updated or not. By default it returns true. If you are sure that the component doesn't need to render after state or props are updated, you can return false value. It is a great place to improve performance as it allows you to prevent a re-render if component receives new prop.
o componentWillUpdate: Executed before re-rendering the component when there are props & state changes confirmed
by shouldComponentUpdate() which returns true.
o componentDidUpdate: Mostly it is used to update the DOM in response to prop or state changes.
o componentWillUnmount: It will be used to cancel any outgoing network requests, or remove all event listeners associated with the component.
React 16.3+
o getDerivedStateFromProps: Invoked right before calling render() and is invoked on every render. This exists for rare use cases where you need a derived state. Worth reading if you need derived state.
o componentDidMount: Executed after first rendering and where all AJAX requests, DOM or state updates, and set up event listeners should occur.
o shouldComponentUpdate: Determines if the component will be updated or not. By default, it returns true. If you are sure that the component doesn't need to render after the state or props are updated, you can return a false value. It is a great place to improve performance as it allows you to prevent a re-render if component receives a new prop.
o getSnapshotBeforeUpdate: Executed right before rendered output is committed to the DOM. Any value returned by this will be passed
into componentDidUpdate(). This is useful to capture information from the DOM i.e. scroll position.

o componentDidUpdate: Mostly it is used to update the DOM in response to prop or state changes. This will not fire
if shouldComponentUpdate() returns false.
o componentWillUnmount It will be used to cancel any outgoing network requests, or remove all event listeners associated with the component.

35. What are Higher-Order Components?
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 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:
. Code reuse, logic and bootstrap abstraction.
i. Render hijacking.
ii. State abstraction and manipulation.
iii. Props manipulation.

36. How to create props proxy for HOC component?
You can add/edit props passed to the component using props proxy pattern like this:

```
function HOC(WrappedComponent) { return class Test extends Component {
render() {
const newProps = { title: "New Header", footer: false, showFeatureX: false, showFeatureY: true,
};

return <WrappedComponent {...this.props} {...newProps} />;
}

};
}
```

37. What is context?
Context provides a way to pass data through the component tree without having to pass props down manually at every level.
For example, authenticated users, locale preferences, UI themes need to be accessed in the application by many components.
const { Provider, Consumer } = React.createContext(defaultValue);

38. What is children prop?
Children is a prop (this.props.children) that allows you to pass components as data to other components, just like any other prop you use. Component tree put between component's opening and closing tag will be passed to that component as children prop.
There are several methods available in the React API to work with this prop.
These
include React.Children.map, React.Children.forEach, React.Children.count, React.C hildren.only, React.Children.toArray.
A simple usage of children prop looks as below,

```
const MyDiv = React.createClass({ render: function () {
return <div>{this.props.children}</div>;
},
});

ReactDOM.render(
<MyDiv>
<span>{"Hello"}</span>
<span>{"World"}</span>
```

</MyDiv>, node
);

39. How to write comments in React?

The comments in React/JSX are similar to JavaScript Multiline comments but are wrapped in curly braces.
Single-line comments:

```
<div>
{/* Single-line comments(In vanilla JavaScript, the single-line comments are represented by double slash(//)) */}
{`Welcome ${user}, let's play React`}
</div>
```

Multi-line comments:

```
<div>
{/* Multi-line comments for more than one line */}
{`Welcome ${user}, let's play React`}
</div>
```

40. What is the purpose of using super constructor with props argument?
A child class constructor cannot make use of this reference until
the super() method has been called. The same applies to ES6 sub-classes as well.
The main reason for passing props parameter to super() call is to access this.props in your child constructors.
Passing props:

```
class MyComponent extends React.Component { constructor(props) {
super(props);

console.log(this.props); // prints { name: 'John', age: 42 }
}
}
```

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: 'John', age: 42 }
}

render() {
// no difference outside constructor console.log(this.props); // prints { name: 'John', age: 42 }
}
}
```

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

41. What is reconciliation?

When a component's props or state change, React decides whether an actual DOM update is necessary by comparing the newly returned element with the previously rendered one. When they are not equal, React will update the DOM. This process is called reconciliation.

42. How to set state with a dynamic key name?
If you are using ES6 or the Babel transpiler to transform your JSX code then you can accomplish this with computed property names.

```
handleInputChange(event) {
this.setState({ [event.target.id]: event.target.value })
}
```

43. What would be the common mistake of function being called every time the component renders?
You need to make sure that function is not being called while passing the function as a parameter.

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

44. Is lazy function supports named exports?
No, currently React.lazy function supports default exports only. If you would like to import modules which are named exports, you can create an intermediate module that reexports it as the default. It also ensures that tree shaking keeps working and don't pull unused components. Let's take a component file which exports multiple named components,

```
// MoreComponents.js
export const SomeComponent = /* ... */; export const UnusedComponent = /* ... */;
```

and reexport MoreComponents.js components in an intermediate file IntermediateComponent.js

```
// IntermediateComponent.js
export { SomeComponent as default } from "./MoreComponents.js";
```

Now you can import the module using lazy function as below,

```
import React, { lazy } from "react";
const SomeComponent = lazy(() => import("./IntermediateComponent.js"));
```

45. Why React uses className over class attribute?
class is a keyword in JavaScript, and JSX is an extension of JavaScript. That's the principal reason why React uses className instead of class. Pass a string as
the className prop.

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

```
}
```

46. What are fragments?

It's a common pattern in React which is used for a component to return multiple elements. Fragments let you group a list of children without adding extra nodes to the DOM.

```
render() { return (
<React.Fragment>
<ChildA />
<ChildB />
<ChildC />
</React.Fragment>

)
}
```

There is also a shorter syntax, but it's not supported in many tools:

```
render() { return (
<>
<ChildA />
<ChildB />
<ChildC />
</>
)
}
```

47. Why fragments are better than container divs?

Below are the list of reasons,

. Fragments are a bit faster and use less memory by not creating an extra DOM node. This only has a real benefit on very large and deep trees.

i. Some CSS mechanisms like Flexbox and CSS Grid have a special parent- child relationships, and adding divs in the middle makes it hard to keep the desired layout.

ii. The DOM Inspector is less cluttered.

48. What are portals in React?

Portal is a recommended way to render children into a DOM node that exists outside the DOM hierarchy of the parent component.

```
ReactDOM.createPortal(child, container);
```

The first argument is any render-able React child, such as an element, string, or fragment. The second argument is a DOM element.

49. What are stateless components?

If the behaviour of a component 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 function components. There are a lot of benefits if you decide to use function components here; they are easy to write, understand, and test, a little faster, and you can avoid the this keyword altogether.

50. What are stateful components?

If the behaviour of a component is dependent on the state of the component then it can be termed as stateful component. These stateful components are always class components and have a state that gets initialized in the constructor. class App extends Component {

```
constructor(props) { super(props);
this.state = { count: 0 };
}

render() {
// ...
}
}
```

React 16.8 Update:

Hooks let you use state and other React features without writing classes.

The Equivalent Functional Component

```
import React, {useState} from 'react'; const App = (props) => {
const [count, setCount] = useState(0);

return (
// JSX
)
}
```

51. How to apply validation on props in React?

When the application is running in development mode, React will automatically check all props that we set on components to make sure they have correct type. If the type is incorrect, React will generate warning messages in the console. It's disabled in production mode due to performance impact. The mandatory props are defined with isRequired.

The set of predefined prop types:

. PropTypes.number
i. PropTypes.string
ii. PropTypes.array
iii. PropTypes.object
iv. PropTypes.func
v. PropTypes.node
vi. PropTypes.element
vii. PropTypes.bool
viii. PropTypes.symbol
ix. PropTypes.any

We can define propTypes for User component as below:

```
import React from "react";
import PropTypes from "prop-types";

class User extends React.Component { static propTypes = {
```

```
name: PropTypes.string.isRequired, age: PropTypes.number.isRequired,
};

render() { return (
<>
<h1>{`Welcome, ${this.props.name}`}</h1>
<h2>{`Age, ${this.props.age}`}</h2>
</>
);
}
}
```

Note: In React v15.5 PropTypes were moved from React.PropTypes to prop- types library.
The Equivalent Functional Component

```
import React from "react";
import PropTypes from "prop-types";

function User({ name, age }) { return (
<>
<h1>{`Welcome, ${name}`}</h1>
<h2>{`Age, ${age}`}</h2>
</>
);
}

User.propTypes = {
name: PropTypes.string.isRequired, age: PropTypes.number.isRequired,

};
```

52. What are the advantages of React?

Below are the list of main advantages of React,

. Increases the application's performance with Virtual DOM.
i. JSX makes code easy to read and write.
ii. It renders both on client and server side (SSR).
iii. Easy to integrate with frameworks (Angular, Backbone) since it is only a view library.
iv. Easy to write unit and integration tests with tools such as Jest.

53. What are the limitations of React?

Apart from the advantages, there are few limitations of React too,

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

54. What are error boundaries in React v16?

Error boundaries are components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI instead of the component tree that crashed.
A class component becomes an error boundary if it defines a new lifecycle method called componentDidCatch(error, info) or static getDerivedStateFromError() :

```
class ErrorBoundary extends React.Component { constructor(props) {
super(props);
this.state = { hasError: false };
}

componentDidCatch(error, info) {
// You can also log the error to an error reporting service logErrorToMyService(error, info);
}

static getDerivedStateFromError(error) {
// Update state so the next render will show the fallback UI. return { hasError: true };
}

render() {
if (this.state.hasError) {
// You can render any custom fallback UI return <h1>{"Something went wrong."}</h1>;
}
return this.props.children;
}
}
```

After that use it as a regular component:

```
<ErrorBoundary>
<MyWidget />
</ErrorBoundary>
```

55. How are error boundaries handled in React v15?

React v15 provided very basic support for error
boundaries using unstable_handleError method. It has been renamed to componentDidCatch in React v16.

56. What are the recommended ways for static type checking?

Normally we use PropTypes library (React.PropTypes moved to a prop-
types package since React v15.5) for type checking in the React applications. For large code bases, it is recommended to use static type checkers such as Flow or TypeScript, that perform type checking at compile time and provide auto- completion features.

57. What is the use of react-dom package?

The react-dom package provides DOM-specific methods that can be used at the top level of your app. Most of the components are not required to use this module. Some of the methods of this package are:

. render()
i. hydrate()
ii. unmountComponentAtNode()
iii. findDOMNode()
iv. createPortal()

58. What is the purpose of render method of react-dom? This method is used to render a React element into the DOM in the supplied container and return a reference to the component. If the React element was previously rendered into container, it will perform an update on it and only mutate the DOM as necessary to reflect the latest changes.
ReactDOM.render(element, container, [callback])
If the optional callback is provided, it will be executed after the component is rendered or updated.

59. What is ReactDOMServer?
The ReactDOMServer object enables you to render components to static markup (typically used on node server). This object is mainly used for server-side rendering (SSR). The following methods can be used in both the server and browser environments:
. renderToString()
i. renderToStaticMarkup()
For example, you generally run a Node-based web server like Express, Hapi, or Koa, and you call renderToString to render your root component to a string, which you then send as response.

```
// using Express
import { renderToString } from "react-dom/server"; import MyPage from "./MyPage";

app.get("/", (req, res) => { res.write(
"<!DOCTYPE html><html><head><title>My Page</title></head><body>"
);
res.write('<div id="content">'); res.write(renderToString(<MyPage />));
res.write("</div></body></html>"); res.end();
});
```

60. How to use innerHTML in React?
The dangerouslySetInnerHTML attribute is React's replacement for
using innerHTML in the browser DOM. Just like innerHTML, it is risky to use this attribute considering cross-site scripting (XSS) attacks. You just need to pass a html object as key and HTML text as value.
In this example MyComponent uses dangerouslySetInnerHTML attribute for setting HTML markup:

```
function createMarkup() {
return { html: "First &middot; Second" };
}

function MyComponent() {
return <div dangerouslySetInnerHTML={createMarkup()} />;
}
```

61. How to use styles in React?
The style attribute accepts a JavaScript object with camelCased properties rather than a CSS string. This is consistent with the DOM style JavaScript property, is more efficient, and prevents XSS security holes.

```
const divStyle = { color: "blue",
```



```
backgroundImage: "url(" + imgUrl + ")",
};

function HelloWorldComponent() {
return <div style={divStyle}>Hello World!</div>;
}
```

Style keys are camelCased in order to be consistent with accessing the properties on DOM nodes in JavaScript (e.g. node.style.backgroundImage).

62. How events are different in React?
Handling events in React elements has some syntactic differences:
. React event handlers are named using camelCase, rather than lowercase.
i. With JSX you pass a function as the event handler, rather than a string.

63. What will happen if you use setState() in constructor?
When you use setState(), then apart from assigning to the object state React also re-renders the component and all its children. You would get error like this: Can only update a mounted or mounting component. So we need to
use this.state to initialize variables inside constructor.

64. What is the impact of indexes as keys?
Keys should be stable, predictable, and unique so that React can keep track of elements.
In the below code snippet each element's key will be based on ordering, rather than tied to the data that is being represented. This limits the optimizations that React can do.

```
{
todos.map((todo, index) => <Todo {...todo} key={index} />);
}
```

If you use element data for unique key, assuming todo.id is unique to this list and stable, React would be able to reorder elements without needing to reevaluate them as much.

```
{
todos.map((todo) => <Todo {...todo} key={todo.id} />);
}
```

65. Is it good to use setState() in componentWillMount() method?
Yes, it is safe to use setState() inside componentWillMount() method. But at the same it is recommended to avoid async initialization
in componentWillMount() lifecycle method. componentWillMount() is invoked immediately before mounting occurs. It is called before render(), therefore setting state in this method will not trigger a re-render. Avoid introducing any side-effects or subscriptions in this method. We need to make sure async calls for component initialization happened in componentDidMount() instead
of componentWillMount(). componentDidMount() {

```
axios.get(`api/todos`)
.then((result) => { this.setState({
messages: [...result.data]
```

```
})
})

}
```

66. What will happen if you use props in initial state?

If the props on the component are changed without the component being refreshed, the new prop value will never be displayed because the constructor function will never update the current state of the component. The initialization of state from props only runs when the component is first created.

The below component won't display the updated input value:

```
class MyComponent extends React.Component { constructor(props) {
super(props);

this.state = { records: [],
inputValue: this.props.inputValue,
};
}

render() {
return <div>{this.state.inputValue}</div>;
}
}
```

Using props inside render method will update the value:

```
class MyComponent extends React.Component { constructor(props) {
super(props);

this.state = { record: [],
};
}

render() {
return <div>{this.props.inputValue}</div>;
}
}
```

67. How do you conditionally render components?

In some cases you want to render different components depending on some state. JSX does not render false or undefined, so you can use conditional short-

circuiting to render a given part of your component only if a certain condition is true.

```
const MyComponent = ({ name, address }) => (
<div>
<h2>{name}</h2>
{address && <p>{address}</p>}
</div>
);
```

If you need an if-else condition then use ternary operator.

```
const MyComponent = ({ name, address }) => (
```



```
<div>
<h2>{name}</h2>
{address ? <p>{address}</p> : <p>{"Address is not available"}</p>}
</div>
);
```

68. Why we need to be careful when spreading props on DOM elements?

When we spread props we run into the risk of adding unknown HTML attributes, which is a bad practice. Instead we can use prop destructuring

with ...rest operator, so it will add only required props. For example,

```
const ComponentA = () => (
<ComponentB isDisplay={true} className={"componentStyle"} />
);

const ComponentB = ({ isDisplay, ...domProps }) => (
<div {...domProps}>{"ComponentB"}</div>
);
```

69. How you use decorators in React?

You can decorate your class components, which is the same as passing the component into a function. Decorators are flexible and readable way of modifying component functionality.

```
@setTitle("Profile")
class Profile extends React.Component {
//....
}

/*

title is a string that will be set as a document title WrappedComponent is what our decorator will receive
when
put directly above a component class as seen in the example above
*/
const setTitle = (title) => (WrappedComponent) => { return class extends React.Component {
componentDidMount() { document.title = title;
}

render() {
return <WrappedComponent {...this.props} />;
}
};
};
```

Note: Decorators are a feature that didn't make it into ES7, but are currently a stage 2 proposal.

70. How do you memoize a component?

There are memoize libraries available which can be used on function components.

For example moize library can memoize the component in another component.

```
import moize from "moize";
import Component from "./components/Component"; // this module exports a non- memoized component
const MemoizedFoo = moize.react(Component); const Consumer = () => {
<div>
{"I will memoize the following entry:"}
<MemoizedFoo />
</div>;
};
```

Update: Since React v16.6.0, we have a React.memo. It provides a higher order component which memoizes component unless the props change. To use it, simply wrap the component using React.memo before you use it.

```
const MemoComponent = React.memo(function MemoComponent(props) {
/* render using props */
}); OR;
export default React.memo(MyFunctionComponent);
```

71. How you implement Server Side Rendering or SSR?

React is already equipped to handle rendering on Node servers. A special version of the DOM renderer is available, which follows the same pattern as on the client side.

```
import ReactDOMServer from "react-dom/server"; import App from "./App";

ReactDOMServer.renderToString(<App />);
```

This method will output the regular HTML as a string, which can be then placed inside a page body as part of the server response. On the client side, React detects the pre-rendered content and seamlessly picks up where it left off.

72. How to enable production mode in React?

You should use Webpack's DefinePlugin method to set NODE_ENV to production, by which it strip out things like propType validation and extra warnings. Apart from this, if you minify the code, for example, Uglify's dead-code elimination to strip out development only code and comments, it will drastically reduce the size of your bundle.

73. What is CRA and its benefits?

The create-react-app CLI tool allows you to quickly create & run React applications with no configuration step.

Let's create Todo App using CRA:

```
# Installation
$ npm install -g create-react-app

# Create new project
$ create-react-app todo-app
$ cd todo-app

# Build, test and run
$ npm run build
$ npm run test
$ npm start
```


It includes everything we need to build a React app:

. React, JSX, ES6, and Flow syntax support.

i. Language extras beyond ES6 like the object spread operator.
ii. Autoprefixed CSS, so you don't need -webkit- or other prefixes.
iii. A fast interactive unit test runner with built-in support for coverage reporting.
iv. A live development server that warns about common mistakes.
v. A build script to bundle JS, CSS, and images for production, with hashes and sourcemaps.

74. What is the lifecycle methods order in mounting?

The lifecycle methods are called in the following order when an instance of a component is being created and inserted into the DOM.

. constructor()
i. static getDerivedStateFromProps()
ii. render()
iii. componentDidMount()

75. What are the lifecycle methods going to be deprecated in React v16?

The following lifecycle methods going to be unsafe coding practices and will be more problematic with async rendering.

. componentWillMount()
i. componentWillReceiveProps()
ii. componentWillUpdate()

Starting with React v16.3 these methods are aliased with UNSAFE_ prefix, and the unprefixed version will be removed in React v17.

76. What is the purpose of getDerivedStateFromProps() lifecycle method?

The new static getDerivedStateFromProps() lifecycle method is invoked after a component is instantiated as well as before it is re-rendered. It can return an object to update state, or null to indicate that the new props do not require any state updates.

```
class MyComponent extends React.Component { static getDerivedStateFromProps(props, state) {
// ...
}
}
```

This lifecycle method along with componentDidUpdate() covers all the use cases of componentWillReceiveProps().

77. What is the purpose of getSnapshotBeforeUpdate() lifecycle method?

The new getSnapshotBeforeUpdate() lifecycle method is called right before DOM updates. The return value from this method will be passed as the third parameter to componentDidUpdate().

```
class MyComponent extends React.Component { getSnapshotBeforeUpdate(prevProps, prevState) {
// ...
}
}
```

This lifecycle method along with componentDidUpdate() covers all the use cases of

componentWillUpdate().

78. Do Hooks replace render props and higher order components?
Both render props and higher-order components render only a single child but in most of the cases Hooks are a simpler way to serve this by reducing nesting in your tree.

79. What is the recommended way for naming components?
It is recommended to name the component by reference instead of using displayName.
Using displayName for naming component:

```
export default React.createClass({ displayName: "TodoApp",
// ...
});
```

The recommended approach:

```
export default class TodoApp extends React.Component {
// ...
}
```

also

```
const TodoApp = () => {
//...
};
export default TodoApp;
```

80. What is the recommended ordering of methods in component class?
Recommended ordering of methods from mounting to render stage:

. static methods
i. constructor()
ii. getChildContext()
iii. componentWillMount()
iv. componentDidMount()
v. componentWillReceiveProps()
vi. shouldComponentUpdate()
vii. componentWillUpdate()
viii. componentDidUpdate()
ix. componentWillUnmount()
x. click handlers or event handlers
like onClickSubmit() or onChangeDescription()
xi. getter methods for render like getSelectReason() or getFooterContent()
xii. optional render methods like renderNavigation() or renderProfilePicture()
xiii. render()

81. What is a switching component?
A switching component is a component that renders one of many components. We need to use object to map prop values to components.
For example, a switching component to display different pages based on page prop:


```
import HomePage from "./HomePage"; import AboutPage from "./AboutPage";

import ServicesPage from "./ServicesPage"; import ContactPage from "./ContactPage";

const PAGES = { home: HomePage, about: AboutPage,
services: ServicesPage, contact: ContactPage,
};

const Page = (props) => {
const Handler = PAGES[props.page] || ContactPage;

return <Handler {...props} />;
};

// The keys of the PAGES object can be used in the prop types to catch dev- time errors.
Page.propTypes = {
page: PropTypes.oneOf(Object.keys(PAGES)).isRequired,
};
```

82. Why we need to pass a function to setState()?
The reason behind for this is that setState() is an asynchronous operation. React batches state changes for performance reasons, so the state may not change immediately after setState() is called. That means you should not rely on the current state when calling setState() since you can't be sure what that state will be. The solution is to pass a function to setState(), with the previous state as an argument. By doing this you can avoid issues with the user getting the old state value on access due to the asynchronous nature of setState().
Let's say the initial count value is zero. After three consecutive increment operations, the value is going to be incremented only by one.

```
// assuming this.state.count === 0 this.setState({ count: this.state.count + 1 }); this.setState({ count:
this.state.count + 1 }); this.setState({ count: this.state.count + 1 });
// this.state.count === 1, not 3
```

If we pass a function to setState(), the count gets incremented correctly.

```
this.setState((prevState, props) => ({ count: prevState.count + props.increment,
}));
// this.state.count === 3 as expected
```

(OR)

Why function is preferred over object for setState()?
React may batch multiple setState() calls into a single update for performance. Because this.props and this.state may be updated asynchronously, you should not rely on their values for calculating the next state. This counter example will fail to update as expected:

```
// Wrong this.setState({
counter: this.state.counter + this.props.increment,
});
```

The preferred approach is to call setState() with function rather than object. That function will receive the previous state as the first argument, and the props at the time the update is applied as the second argument.

```
// Correct
this.setState((prevState, props) => ({ counter: prevState.counter + props.increment,
}));
```

83. What is strict mode in React?
React.StrictMode is a useful component for highlighting potential problems in an application. Just like <Fragment>, <StrictMode> does not render any extra DOM elements. It activates additional checks and warnings for its descendants. These checks apply for development mode only.

```
import React from "react";

function ExampleApplication() { return (
<div>
<Header />
<React.StrictMode>
<div>
<ComponentOne />
<ComponentTwo />
</div>
</React.StrictMode>
<Header />
</div>
);
}
```

In the example above, the strict mode checks apply
to <ComponentOne> and <ComponentTwo> components only.

84. What are React Mixins?
Mixins are a way to totally separate components to have a common functionality.
Mixins should not be used and can be replaced with higher-order components or decorators.
One of the most commonly used mixins is PureRenderMixin. You might be using it in some components to prevent unnecessary re-renders when the props and state are shallowly equal to the previous props and state:

```
const PureRenderMixin = require("react-addons-pure-render-mixin");

const Button = React.createClass({ mixins: [PureRenderMixin],
// ...
});
```

85. Why is isMounted() an anti-pattern and what is the proper solution?
The primary use case for isMounted() is to avoid calling setState() after a component has been unmounted, because it will emit a warning.

```
if (this.isMounted()) { this.setState({...})
}
```

Checking isMounted() before calling setState() does eliminate the warning, but it also defeats the purpose of the warning. Using isMounted() is a code smell because the only reason you would check is because you think you might be holding a reference after the component has unmounted.
An optimal solution would be to find places where setState() might be called after a component has unmounted, and fix them. Such situations most commonly occur due to callbacks, when a component is waiting for some data and gets unmounted before the data arrives. Ideally, any callbacks should be canceled
in componentWillUnmount(), prior to unmounting.

86. What are the Pointer Events supported in React?
Pointer Events provide a unified way of handling all input events. In the old days we had a mouse and respective event listeners to handle them but nowadays we have many devices which don't correlate to having a mouse, like phones with touch surface or pens. We need to remember that these events will only work in browsers that support the Pointer Events specification.

The following event types are now available in React DOM:

. onPointerDown
i. onPointerMove
ii. onPointerUp
iii. onPointerCancel
iv. onGotPointerCapture
v. onLostPointerCapture
vi. onPointerEnter
vii. onPointerLeave
viii. onPointerOver
ix. onPointerOut

87. Why should component names start with capital letter?
If you are rendering your component using JSX, the name of that component has to begin with a capital letter otherwise React will throw an error as an unrecognized tag. This convention is because only HTML elements and SVG tags can begin with a lowercase letter.

```
class SomeComponent extends Component {
// Code goes here
}
```

You can define component class which name starts with lowercase letter, but when it's imported it should have capital letter. Here lowercase is fine:

```
class myComponent extends Component { render() {
return <div />;
}
}

export default myComponent;
```

While when imported in another file it should start with capital letter:

```
import MyComponent from "./myComponent";
```

What are the exceptions on React component naming?
The component names should start with an uppercase letter but there are few exceptions to this convention. The lowercase tag names with a dot (property accessors) are still considered as valid component names. For example, the below tag can be compiled to a valid component,

```
render() {
return (
<obj.component/> // `React.createElement(obj.component)`
)
}
```

88. Are custom DOM attributes supported in React v16?
Yes. In the past, React used to ignore unknown DOM attributes. If you wrote JSX with an attribute that React doesn't recognize, React would just skip it.
For example, let's take a look at the below attribute:
<div mycustomattribute={"something"} />
Would render an empty div to the DOM with React v15:

In React v16 any unknown attributes will end up in the DOM:

This is useful for supplying browser-specific non-standard attributes, trying new DOM APIs, and integrating with opinionated third-party libraries.

89. What is the difference between constructor and getInitialState?
You should initialize state in the constructor when using ES6 classes, and getInitialState() method when using React.createClass().
Using ES6 classes:
class MyComponent extends React.Component { constructor(props) {
super(props); this.state = {
/* initial state */
};
}
}
Using React.createClass():
const MyComponent = React.createClass({ getInitialState() {
return {

/* initial state */
};
},
});
Note: React.createClass() is deprecated and removed in React v16. Use plain JavaScript classes instead.

90. Can you force a component to re-render without calling setState?
By default, when your component's state or props change, your component will re-render. If your render() method depends on some other data, you can tell React that the component needs re-rendering by calling forceUpdate(). component.forceUpdate(callback);
It is recommended to avoid all uses of forceUpdate() and only read from this.props and this.state in render().

91. What is the difference between super() and super(props) in React using ES6 classes?
When you want to access this.props in constructor() then you should pass props to super() method.
Using super(props):
class MyComponent extends React.Component { constructor(props) {
super(props);
console.log(this.props); // { name: 'John', ... }
}
}
Using super():

class MyComponent extends React.Component { constructor(props) {
super();
console.log(this.props); // undefined
}
}
Outside constructor() both will display same value for this.props.

92. How to loop inside JSX?
You can simply use Array.prototype.map with ES6 arrow function syntax.

For example, the items array of objects is mapped into an array of components:
<tbody>
{items.map((item) => (
<SomeComponent key={item.id} name={item.name} />
))}
</tbody>
But you can't iterate using for loop:
<tbody>
for (let i = 0; i < items.length; i++) {
<SomeComponent key={items[i].id} name={items[i].name} />
}
</tbody>
This is because JSX tags are transpiled into function calls, and you can't use statements inside expressions. This may change thanks to do expressions which are stage 1 proposal.

93. How do you access props in attribute quotes?
React (or JSX) doesn't support variable interpolation inside an attribute value. The below representation won't work:

But you can put any JS expression inside curly braces as the entire attribute value. So the below expression works:

Using template strings will also work:

94. What is React proptype array with shape?
If you want to pass an array of objects to a component with a particular shape then use React.PropTypes.shape() as an argument to React.PropTypes.arrayOf(). ReactComponent.propTypes = {
arrayWithShape: React.PropTypes.arrayOf(React.PropTypes.shape({
color: React.PropTypes.string.isRequired, fontSize: React.PropTypes.number.isRequired,
})
).isRequired,
};

95. How to conditionally apply class attributes?

You shouldn't use curly braces inside quotes because it is going to be evaluated as a string.

Instead you need to move curly braces outside (don't forget to include spaces between class names):
<div className={'btn-panel ' + (this.props.visible ? 'show' : 'hidden')}>
Template strings will also work:
<div className={`btn-panel ${this.props.visible ? 'show' : 'hidden'}`}>

96. What is the difference between React and ReactDOM?
The react package
contains React.createElement(), React.Component, React.Children, and other helpers related to elements and component classes. You can think of these as the isomorphic or universal helpers that you need to build components. The react- dom package contains ReactDOM.render(), and in react-dom/server we have server- side rendering support
with ReactDOMServer.renderToString() and ReactDOMServer.renderToStaticMarkup().

97. Why ReactDOM is separated from React?
The React team worked on extracting all DOM-related features into a separate library called ReactDOM. React v0.14 is the first release in which the libraries are split. By looking at some of the packages, react-native, react-art, react-canvas, and react-three, it has become clear that the beauty and essence of React has nothing to do with browsers or the DOM.
To build more environments that React can render to, React team planned to split the main React package into two: react and react-dom. This paves the way to writing components that can be shared between the web version of React and React Native.

98. How to use React label element?
If you try to render a <label> element bound to a text input using the
standard for attribute, then it produces HTML missing that attribute and prints a warning to the console.
<label for={'user'}>{'User'}</label>
<input type={'text'} id={'user'} />
Since for is a reserved keyword in JavaScript, use htmlFor instead.
<label htmlFor={'user'}>{'User'}</label>
<input type={'text'} id={'user'} />

99. How to combine multiple inline style objects?
You can use spread operator in regular React:
<button style={{ ...styles.panel.button, ...styles.panel.submitButton }}>
{"Submit"}
</button>
If you're using React Native then you can use the array notation:
<button style={[styles.panel.button, styles.panel.submitButton]}>
{"Submit"}
</button>

100. How to re-render the view when the browser is resized?

You can listen to the resize event in componentDidMount() and then update the dimensions (width and height). You should remove the listener
in componentWillUnmount() method.

```
class WindowDimensions extends React.Component { constructor(props) {
super(props);
this.updateDimensions = this.updateDimensions.bind(this);
}

componentWillMount() { this.updateDimensions();
}

componentDidMount() {
window.addEventListener("resize", this.updateDimensions);
}

componentWillUnmount() {

window.removeEventListener("resize", this.updateDimensions);
}

updateDimensions() { this.setState({
width: window.innerWidth, height: window.innerHeight,
});
}

render() { return (
<span>
{this.state.width} x {this.state.height}
</span>
);
}
}
```

101. What is the difference
between setState() and replaceState() methods?
When you use setState() the current and previous states are
merged. replaceState() throws out the current state, and replaces it with only what you provide. Usually setState() is used unless you really need to remove all previous keys for some reason. You can also set state to false/null in setState() instead of using replaceState().

102. How to listen to state changes?
The componentDidUpdate lifecycle method will be called when state changes. You can compare provided state and props values with current state and props to determine if something meaningful changed.
componentDidUpdate(object prevProps, object prevState)
Note: The previous releases of ReactJS also uses componentWillUpdate(object nextProps, object nextState) for state changes. It has been deprecated in latest releases.

103. What is the recommended approach of removing an array element in React state?

The better approach is to use Array.prototype.filter() method.
For example, let's create a removeItem() method for updating the state.

```
removeItem(index) { this.setState({
data: this.state.data.filter((item, i) => i !== index)
})
}
```

104. Is it possible to use React without rendering HTML?
It is possible. Below are the possible options:

```
render() { return false
}
render() { return true
}
render() { return null
}
```

React version >=16.0.0:

```
render() { return []
}
render() { return ""
}
```

React version >=16.2.0:

```
render() {
return <React.Fragment></React.Fragment>
}
render() { return <></>
}
```

React version >=18.0.0:

```
render() {
return undefined
}
```

105. How to pretty print JSON with React?

We can use <pre> tag so that the formatting of the JSON.stringify() is retained:

```
const data = { name: "John", age: 42 };

class User extends React.Component { render() {
return <pre>{JSON.stringify(data, null, 2)}</pre>;
}
}

React.render(<User />, document.getElementById("container"));
```

106. Why you can't update props in React?
The React philosophy is that props should be immutable and top-down. This means that a parent can send

any prop values to a child, but the child can't modify received props.

107. How to focus an input element on page load?
You can do it by creating ref for input element and using it in componentDidMount():

```
class App extends React.Component { componentDidMount() {
this.nameInput.focus();
}

render() { return (
<div>
<input defaultValue={"Won't focus"} />
<input
ref={(input) => (this.nameInput = input)} defaultValue={"Will focus"}
/>
</div>
);
}
}

ReactDOM.render(<App />, document.getElementById("app")); Also in Functional component (react
16.08 and above) import React, { useEffect, useRef } from "react";
const App = () => {
const inputElRef = useRef(null);


useEffect(() => { inputElRef.current.focus();
}, []);

return (
<div>
<input defaultValue={"Won't focus"} />
<input ref={inputElRef} defaultValue={"Will focus"} />
</div>
);
};

ReactDOM.render(<App />, document.getElementById("app"));
```

108. What are the possible ways of updating objects in state?
i. Calling setState() with an object to merge with state:
   Using Object.assign() to create a copy of the object:
   const user = Object.assign({}, this.state.user, { age: 42 }); this.setState({ user });
   Using spread operator:
   const user = { ...this.state.user, age: 42 }; this.setState({ user });
ii. Calling setState() with a function:
iii. this.setState((prevState) => ({
iv. user: {
v. prevState.user,

vi. age: 42,
vii. },
}));

110. How can we find the version of React at runtime in the browser?
You can use React.version to get the version.
const REACT_VERSION = React.version;

ReactDOM.render(
{`React version: ${REACT_VERSION}`} , document.getElementById("app")
);

111. What are the approaches to include polyfills in your create-react-app?
There are approaches to include polyfills in create-react-app,
i. Manual import from core-js:
Create a file called (something like) polyfills.js and import it into
root index.js file. Run npm install core-js or yarn add core-js and import your specific required features.
import "core-js/fn/array/find"; import "core-js/fn/array/includes"; import "core-js/fn/number/is-nan";
ii. Using Polyfill service:
Use the polyfill.io CDN to retrieve custom, browser-specific polyfills by adding this line to index.html:
<script src="https://cdn.polyfill.io/v2/polyfill.min.js?features=default,Array.p rototype.includes"></script>
In the above script we had to explicitly request
the Array.prototype.includes feature as it is not included in the default feature set.

112. How to use https instead of http in create-react-app?
You just need to use HTTPS=true configuration. You can edit your package.json scripts section:
"scripts": {
"start": "set HTTPS=true && react-scripts start"
}
or just run set HTTPS=true && npm start

113. How to avoid using relative path imports in create- react-app?
Create a file called .env in the project root and write the import path:
NODE_PATH=src/app
After that restart the development server. Now you should be able to import anything inside src/app
without relative paths.

114. How to add Google Analytics for React Router?
Add a listener on the history object to record each page view:
history.listen(function (location) {
window.ga("set", "page", location.pathname + location.search); window.ga("send", "pageview",
location.pathname + location.search);

});

115. How to update a component every second?
You need to use setInterval() to trigger the change, but you also need to clear the timer when the
component unmounts to prevent errors and memory leaks. componentDidMount() {
this.interval = setInterval(() => this.setState({ time: Date.now() }), 1000)
}

componentWillUnmount() { clearInterval(this.interval)
}

116. How do you apply vendor prefixes to inline styles in React?
React does not apply vendor prefixes automatically. You need to add vendor prefixes manually.
<div
style={{
transform: "rotate(90deg)",
WebkitTransform: "rotate(90deg)", // note the capital 'W' here msTransform: "rotate(90deg)", // 'ms' is the
only lowercase vendor prefix
}}
/>

117. How to import and export components using React and ES6?
You should use default for exporting the components
import React from "react";

import User from "user";

export default class MyProfile extends React.Component { render() {
return <User type="customer">//...</User>;
}
}
With the export specifier, the MyProfile is going to be the member and exported to this module and the
same can be imported without mentioning the name in other components.

119. Why is a component constructor called only once?
React's reconciliation algorithm assumes that without any information to the contrary, if a custom
component appears in the same place on subsequent renders, it's the same component as before, so reuses
the previous instance rather than creating a new one.

120. How to define constants in React?
You can use ES7 static field to define constant.
class MyComponent extends React.Component { static DEFAULT_PAGINATION = 10;

}

121. How to programmatically trigger click event in React?
You could use the ref prop to acquire a reference to the
underlying HTMLInputElement object through a callback, store the reference as a class property, then use that reference to later trigger a click from your event handlers using the HTMLElement.click method.
This can be done in two steps:
i. Create ref in render method:
<input ref={(input) => (this.inputElement = input)} />
ii. Apply click event in your event handler:
this.inputElement.click();

122. Is it possible to use async/await in plain React?
If you want to use async/await in React, you will need Babel and transform-async- to-generator plugin.
React Native ships with Babel and a set of transforms.

123. What are the common folder structures for React?
There are two common practices for React project file structure.
i. Grouping by features or routes:
One common way to structure projects is locate CSS, JS, and tests together, grouped by feature or route.
common/
├─ Avatar.js
├─ Avatar.css
├─ APIUtils.js
└─ APIUtils.test.js feed/
├─ index.js
├─ Feed.js
├─ Feed.css
├─ FeedStory.js
├─ FeedStory.test.js
└─ FeedAPI.js profile/
├─ index.js
├─ Profile.js
├─ ProfileHeader.js
├─ ProfileHeader.css
└─ ProfileAPI.js
ii. Grouping by file type:
Another popular way to structure projects is to group similar files together.
api/
├─ APIUtils.js
├─ APIUtils.test.js
├─ ProfileAPI.js
└─ UserAPI.js components/
├─ Avatar.js
├─ Avatar.css
├─ Feed.js

├─ Feed.css
├─ FeedStory.js

├─ FeedStory.test.js
├─ Profile.js
├─ ProfileHeader.js
└─ ProfileHeader.css

124. What are the popular packages for animation?
React Transition Group and React Motion are popular animation packages in React ecosystem.

125. What is the benefit of styles modules?
It is recommended to avoid hard coding style values in components. Any values that are likely to be used across different UI components should be extracted into their own modules.
For example, these styles could be extracted into a separate component:
export const colors = { white,
black, blue,
};

export const space = [0, 8, 16, 32, 64];
And then imported individually in other components:
import { space, colors } from "./styles";

126. What are the popular React-specific linters?
ESLint is a popular JavaScript linter. There are plugins available that analyse specific code styles. One of the most common for React is an npm package called eslint-plugin-react. By default, it will check a number of best practices, with rules checking things from keys in iterators to a complete set of prop types. Another popular plugin is eslint-plugin-jsx-a11y, which will help fix common issues with accessibility. As JSX offers slightly different syntax to regular HTML, issues with alt text and tabindex, for example, will not be picked up by regular plugins.

127. How to make AJAX call and in which component lifecycle methods should I make an AJAX call?
You can use AJAX libraries such as Axios, jQuery AJAX, and the browser built-
in fetch. You should fetch data in the componentDidMount() lifecycle method. This is so you can use setState() to update your component when the data is retrieved. For example, the employees list fetched from API and set local state:
class MyComponent extends React.Component { constructor(props) {
super(props); this.state = {
employees: [], error: null,
};
}

componentDidMount() { fetch("https://api.example.com/items")
.then((res) => res.json())
.then(
(result) => { this.setState({

```
employees: result.employees,
});
},
(error) => { this.setState({ error });
}
);
}

render() {
const { error, employees } = this.state; if (error) {
return <div>Error: {error.message}</div>;
} else { return (
<ul>
{employees.map((employee) => (
<li key={employee.name}>
{employee.name}-{employee.experience}
</li>
))}
</ul>
);
}
}
}
```

React Redux

152. What is flux?

Flux is an application design paradigm used as a replacement for the more traditional MVC pattern. It is not a framework or a library but a new kind of architecture that complements React and the concept of Unidirectional Data Flow. Facebook uses this pattern internally when working with React.

153. What is Redux?

Redux is a predictable state container for JavaScript apps based on the Flux design pattern. Redux can be used together with React, or with any other view library. It is tiny (about 2kB) and has no dependencies.

154. What are the core principles of Redux?

Redux follows three fundamental principles:

i. Single source of truth: The state of your whole application is stored in an object tree within a single store. The single state tree makes it easier to keep track of changes over time and debug or inspect the application.

ii. State is read-only: The only way to change the state is to emit an action, an object describing what happened. This ensures that neither the views nor the network callbacks will ever write directly to the state.

iii. Changes are made with pure functions: To specify how the state tree is transformed by actions, you write reducers. Reducers are just pure functions that take the previous state and an action as parameters, and return the next state.

155. What are the downsides of Redux compared to Flux?

Instead of saying downsides we can say that there are few compromises of using Redux over Flux. Those are as follows:

i. You will need to learn to avoid mutations: Flux is un-opinionated about mutating data, but Redux doesn't like mutations and many packages complementary to Redux assume you never mutate the state. You can enforce this with dev-only packages like redux-immutable-state-invariant, Immutable.js, or instructing your team to write non-mutating code.

ii. You're going to have to carefully pick your packages: While Flux explicitly doesn't try to solve problems such as undo/redo, persistence, or forms, Redux has extension points such as middleware and store enhancers, and it has spawned a rich ecosystem.

iii. There is no nice Flow integration yet: Flux currently lets you do very impressive static type checks which Redux doesn't support yet.

156. What is the difference

between mapStateToProps() and mapDispatchToProps()? mapStateToProps() is a utility which helps your component get updated state (which is updated by some other components):

```
const mapStateToProps = (state) => { return {
todos: getVisibleTodos(state.todos, state.visibilityFilter),
};
};
```

mapDispatchToProps() is a utility which will help your component to fire an action event (dispatching action which may cause change of application state):

```
const mapDispatchToProps = (dispatch) => { return {

onTodoClick: (id) => { dispatch(toggleTodo(id));
},
};
};
```

It is recommended to always use the "object shorthand" form for the mapDispatchToProps.

Redux wraps it in another function that looks like (...args) => dispatch(onTodoClick(...args)), and pass that wrapper function as a prop to your component.

```
const mapDispatchToProps = { onTodoClick,
};
```

157. Can I dispatch an action in reducer?

Dispatching an action within a reducer is an anti-pattern. Your reducer should be without side effects, simply digesting the action payload and returning a new state object. Adding listeners and dispatching actions within the reducer can lead to chained actions and other side effects.

158. How to access Redux store outside a component?

You just need to export the store from the module where it created with createStore(). Also, it shouldn't pollute the global window object. store = createStore(myReducer);

```
export default store;
```

188. What is the difference between React and Angular?

Let's see the difference between React and Angular in a table format.

React	Angular
React is a library and has only the View layer	Angular is a framework and has complete MVC functionality
React handles rendering on the server side	AngularJS renders only on the client side but Angular 2 and above renders on the server side
React uses JSX that looks like HTML in JS which can be confusing	Angular follows the template approach for HTML, which makes code shorter and easy to understand
React Native, which is a React type to build mobile applications are faster and more stable	Ionic, Angular's mobile native app is relatively less stable and slower
In React, data flows only in one way and hence debugging is easy	In Angular, data flows both way i.e it has two-way data binding between children and parent and hence debugging is often difficult

223. How to prevent unnecessary updates using setState?

You can compare the current value of the state with an existing state value and decide whether to rerender the page or not. If the values are the same then you need to return null to stop re-rendering otherwise return the latest state value.

For example, the user profile information is conditionally rendered as follows,

```
getUserProfile = (user) => {
const latestAddress = user.address; this.setState((state) => {
if (state.address === latestAddress) { return null;
} else {
return { title: latestAddress };
}
});
};
```

224. How do you render Array, Strings and Numbers in React 16 Version?

Arrays: Unlike older releases, you don't need to make sure render method return a single element in React16. You are able to return multiple sibling elements without a wrapping element by returning an array.

For example, let us take the below list of developers,

```
const ReactJSDevs = () => { return [
<li key="1">John</li>,
<li key="2">Jackie</li>,
<li key="3">Jordan</li>,
];
};
```


You can also merge this array of items in another array component.

```
const JSDevs = () => { return (
<ul>
<li>Brad</li>
<li>Brodge</li>
<ReactJSDevs />
<li>Brandon</li>
</ul>
);
};
```

Strings and Numbers: You can also return string and number type from the render method.

```
render() {
return 'Welcome to ReactJS questions';
}
// Number render() { return 2018;
}
```

225. How to use class field declarations syntax in React classes?

React Class Components can be made much more concise using the class field declarations. You can initialize the local state without using the constructor and declare class methods by using arrow functions without the extra need to bind them.

Let's take a counter example to demonstrate class field declarations for state without using constructor and methods without binding,

```
class Counter extends Component { state = { value: 0 };

handleIncrement = () => { this.setState((prevState) => ({
value: prevState.value + 1,
}));
};

handleDecrement = () => { this.setState((prevState) => ({
value: prevState.value - 1,
}));
};

render() { return (
<div>
{this.state.value}

<button onClick={this.handleIncrement}>+</button>
<button onClick={this.handleDecrement}>-</button>
</div>
);
}
}
```

226. What are hooks?
Hooks is a special function (introduced as a new feature in React 16.8) that lets you use state and other React features without writing a class.
Let's see an example of useState hook:

```
import { useState } from "react"; function Example() {
// Declare a new state variable, which we'll call "count" const [count, setCount] = useState(0);

return (
<div>
<p>You clicked {count} times</p>
<button onClick={() => setCount(count + 1)}>Click me</button>
</div>
);
}
```

Note: Hooks can be used inside an existing function component.

227. What rules need to be followed for hooks?
You need to follow two rules in order to use hooks,
i. Call Hooks only at the top level of your react functions. i.e, You shouldn't call Hooks inside loops, conditions, or nested functions. This will ensure that Hooks are called in the same order each time a component renders and it preserves the state of Hooks between multiple useState and useEffect calls.
ii. Call Hooks from React Functions only. i.e, You shouldn't call Hooks from regular JavaScript functions.

228. How to ensure hooks followed the rules in your project?
React team released an ESLint plugin called eslint-plugin-react-hooks that enforces these two rules. You can add this plugin to your project using the below command,

```
npm install eslint-plugin-react-hooks@next
```

And apply the below config in your ESLint config file,

```
// Your ESLint configuration
{
"plugins": [
// ... "react-hooks"
],
"rules": {
// ...
"react-hooks/rules-of-hooks": "error"
}
}
```

Note: This plugin is intended to use in Create React App by default.

238. What is the required method to be defined for a class component?
The render() method is the only required method in a class component. i.e, All methods other than render method are optional for a class component.

239. What are the possible return types of render method?
Below are the list of following types used and return from render method,
i. React elements: Elements that instruct React to render a DOM node. It includes html elements such as and user defined elements.
ii. Arrays and fragments: Return multiple elements to render as Arrays and Fragments to wrap multiple elements
iii. Portals: Render children into a different DOM subtree.
iv. String and numbers: Render both Strings and Numbers as text nodes in the DOM
v. Booleans or null: Doesn't render anything but these types are used to conditionally render content.

240. What is the main purpose of constructor?

The constructor is mainly used for two purposes,
i. To initialize local state by assigning object to this.state
ii. For binding event handler methods to the instance For example, the below code covers both the above cases,

```
constructor(props) { super(props);
// Don't call this.setState() here! this.state = { counter: 0 };
this.handleClick = this.handleClick.bind(this);
}
```

241. Is it mandatory to define constructor for React component?
No, it is not mandatory. i.e, If you don't initialize state and you don't bind methods, you don't need to implement a constructor for your React component.

242. What are default props?
The defaultProps are defined as a property on the component class to set the default props for the class. This is used for undefined props, but not for null props.
For example, let us create color default prop for the button component,

```
class MyButton extends React.Component {
// ...
}

MyButton.defaultProps = { color: "red",
};
```

If props.color is not provided then it will set the default value to 'red'. i.e,
Whenever you try to access the color prop it uses default value

```
render() {
return <MyButton /> ; // props.color will be set to red
}
```

Note: If you provide null value then it remains null value.

243. Why should not call setState in componentWillUnmount?
You should not call setState() in componentWillUnmount() because once a component instance is unmounted, it will never be mounted again.

244. What is the purpose of getDerivedStateFromError?
This lifecycle method is invoked after an error has been thrown by a descendant component. It receives the error that was thrown as a parameter and should return a value to update state.
The signature of the lifecycle method is as follows,

```
static getDerivedStateFromError(error)
```

Let us take error boundary use case with the above lifecycle method for demonstration purpose,

```
class ErrorBoundary extends React.Component { constructor(props) {
super(props);
this.state = { hasError: false };
}

static getDerivedStateFromError(error) {
// Update state so the next render will show the fallback UI. return { hasError: true };
}

render() {
if (this.state.hasError) {
// You can render any custom fallback UI return <h1>Something went wrong.</h1>;
}

return this.props.children;
}
}
```

245. What is the methods order when component re- rendered?

An update can be caused by changes to props or state. The below methods are called in the following order when a component is being re-rendered.

i. static getDerivedStateFromProps()
ii. shouldComponentUpdate()
iii. render()
iv. getSnapshotBeforeUpdate()
v. componentDidUpdate()

246. What are the methods invoked during error handling?
Below methods are called when there is an error during rendering, in a lifecycle method, or in the constructor of any child component.

i. static getDerivedStateFromError()

ii. componentDidCatch()

247. What is the purpose of displayName class property?
The displayName string is used in debugging messages. Usually, you don't need to set it explicitly because it's inferred from the name of the function or class that defines the component. You might want to set it explicitly if you want to display a different name for debugging purposes or when you create a higher-order component.
For example, To ease debugging, choose a display name that communicates that it's the result of a withSubscription HOC.

```
function withSubscription(WrappedComponent) { class WithSubscription extends React.Component {
/* ... */
}
WithSubscription.displayName = `WithSubscription(${getDisplayName( WrappedComponent
)})`;
return WithSubscription;
}
function getDisplayName(WrappedComponent) { return (
WrappedComponent.displayName || WrappedComponent.name || "Component"
);
}
```

248. What is the browser support for react applications?
React supports all popular browsers, including Internet Explorer 9 and above, although some polyfills are required for older browsers such as IE 9 and IE 10. If you use es5-shim and es5-sham polyfill then it even support old browsers that doesn't support ES5 methods.

249. What is the purpose of unmountComponentAtNode method?
This method is available from react-dom package and it removes a mounted React component from the DOM and clean up its event handlers and state. If no component was mounted in the container, calling this function does nothing.
Returns true if a component was unmounted and false if there was no component to unmount.
The method signature would be as follows,

```
ReactDOM.unmountComponentAtNode(container);
```

250. What is code-splitting?
Code-Splitting is a feature supported by bundlers like Webpack and Browserify which can create multiple bundles that can be dynamically loaded at runtime. The react project supports code splitting via dynamic import() feature.
For example, in the below code snippets, it will make moduleA.js and all its unique dependencies as a separate chunk that only loads after the user clicks the 'Load' button. moduleA.js

```
const moduleA = "Hello"; export { moduleA };
```

App.js

```
import React, { Component } from "react"; class App extends Component {
```

```
handleClick = () => { import("./moduleA")
.then(({ moduleA }) => {
// Use moduleA
})
.catch((err) => {
// Handle failure
});
};

render() { return (
<div>
<button onClick={this.handleClick}>Load</button>
</div>
);
}
}

export default App;
```

251. What is the benefit of strict mode?

The will be helpful in the below cases

i. Identifying components with unsafe lifecycle methods.
ii. Warning about legacy string ref API usage.
iii. Detecting unexpected side effects.
iv. Detecting legacy context API.
v. Warning about deprecated findDOMNode usage

252. What are Keyed Fragments?

The Fragments declared with the explicit <React.Fragment> syntax may have keys. The general use case is mapping a collection to an array of fragments as below,

```
function Glossary(props) { return (
<dl>
{props.items.map((item) => (
// Without the `key`, React will fire a key warning
<React.Fragment key={item.id}>
<dt>{item.term}</dt>
<dd>{item.description}</dd>
</React.Fragment>

))}
</dl>
);
}
```

Note: key is the only attribute that can be passed to Fragment. In the future, there might be a support for additional attributes, such as event handlers.

253. Does React support all HTML attributes?

As of React 16, both standard or custom DOM attributes are fully supported. Since React components often take both custom and DOM-related props, React uses the camelCase convention just like the DOM APIs.

Let us take few props with respect to standard HTML attributes,

```
<div tabIndex="-1" /> // Just like node.tabIndex DOM API
<div className="Button" /> // Just like node.className DOM API
<input readOnly={true} /> // Just like node.readOnly DOM API
```

These props work similarly to the corresponding HTML attributes, with the exception of the special cases. It also support all SVG attributes.

254. What are the limitations with HOCs?

Higher-order components come with a few caveats apart from its benefits. Below are the few listed in an order,

i. Don't use HOCs inside the render method: It is not recommended to apply a HOC to a component within the render method of a component.

```
ii. render() {
iii. // A new version of EnhancedComponent is created on every render
iv. // EnhancedComponent1 !== EnhancedComponent2
v. const EnhancedComponent = enhance(MyComponent);
vi. // That causes the entire subtree to unmount/remount each time!
vii. return <EnhancedComponent />;
}
```

The above code impacts on performance by remounting a component that causes the state of that component and all of its children to be lost.

Instead, apply HOCs outside the component definition so that the resulting component is created only once.

viii. Static methods must be copied over: When you apply a HOC to a component the new component does not have any of the static methods of the original component

```
ix. // Define a static method
x. WrappedComponent.staticMethod = function () {
xi. /*...*/
xii. };
xiii. // Now apply a HOC
xiv. const EnhancedComponent = enhance(WrappedComponent); xv.
xvi. // The enhanced component has no static method
typeof EnhancedComponent.staticMethod === "undefined"; // true
```

You can overcome this by copying the methods onto the container before returning it,

```
function enhance(WrappedComponent) { class Enhance extends React.Component {
/*...*/
}
// Must know exactly which method(s) to copy :( Enhance.staticMethod =
WrappedComponent.staticMethod; return Enhance;
}
```

xvii. Refs aren't passed through: For HOCs you need to pass through all props to the wrapped component but this does not work for refs. This is because ref is not really a prop similar to key. In this case you need to use the React.forwardRef API

255. How to debug forwardRefs in DevTools?
React.forwardRef accepts a render function as parameter and DevTools uses this function to determine what to display for the ref forwarding component.
For example, If you don't name the render function or not using displayName property then it will appear as "ForwardRef" in the DevTools,

```
const WrappedComponent = React.forwardRef((props, ref) => { return <LogProps {...props}
forwardedRef={ref} />;
});
```

But If you name the render function then it will appear as "ForwardRef(myFunction)"

```
const WrappedComponent = React.forwardRef(function myFunction(props, ref) {

return <LogProps {...props} forwardedRef={ref} />;
});
```

As an alternative, You can also set displayName property for forwardRef function,

```
function logProps(Component) {
class LogProps extends React.Component {
// ...
}

function forwardRef(props, ref) {
return <LogProps {...props} forwardedRef={ref} />;
}

// Give this component a more helpful display name in DevTools.
// e.g. "ForwardRef(logProps(MyComponent))"
const name = Component.displayName || Component.name; forwardRef.displayName =
`logProps(${name})`;

return React.forwardRef(forwardRef);
}
```

256. When component props defaults to true?
If you pass no value for a prop, it defaults to true. This behavior is available so that it matches the behavior of HTML.
For example, below expressions are equivalent,

```
<MyInput autocomplete />

<MyInput autocomplete={true} />
```

Note: It is not recommended to use this approach because it can be confused with the ES6 object shorthand (example, {name} which is short for {name: name})

257. What is NextJS and major features of it?
Next.js is a popular and lightweight framework for static and server-rendered applications built with React. It also provides styling and routing solutions. Below are the major features provided by NextJS,

i. Server-rendered by default
ii. Automatic code splitting for faster page loads
iii. Simple client-side routing (page based)
iv. Webpack-based dev environment which supports (HMR)

v. Able to implement with Express or any other Node.js HTTP server
vi. Customizable with your own Babel and Webpack configurations

258. How do you pass an event handler to a component?
You can pass event handlers and other functions as props to child components. It can be used in child component as below,

```
<button onClick="{this.handleClick}"></button>
```

259. Is it good to use arrow functions in render methods?
Yes, You can use. It is often the easiest way to pass parameters to callback functions. But you need to optimize the performance while using it.

```
class Foo extends Component { handleClick() {
console.log("Click happened");
}
render() {
return <button onClick={() => this.handleClick()}>Click Me</button>;
}
}
```

Note: Using an arrow function in render method creates a new function each time the component renders, which may have performance implications

260. How to prevent a function from being called multiple times?
If you use an event handler such as onClick or onScroll and want to prevent the callback from being fired too quickly, then you can limit the rate at which callback is executed. This can be achieved in the below possible ways,

i. Throttling: Changes based on a time based frequency. For example, it can be used using _.throttle lodash function
ii. Debouncing: Publish changes after a period of inactivity. For example, it can be used using _.debounce lodash function

iii. RequestAnimationFrame throttling: Changes based on requestAnimationFrame. For example, it can be used using raf-schd lodash function

261. How JSX prevents Injection Attacks?
React DOM escapes any values embedded in JSX before rendering them. Thus it ensures that you can never inject anything that's not explicitly written in your application. Everything is converted to a string before being rendered.
For example, you can embed user input as below,

```
const name = response.potentiallyMaliciousInput; const element = <h1>{name}</h1>;
```

This way you can prevent XSS(Cross-site-scripting) attacks in the application.

262. How do you update rendered elements?

You can update UI(represented by rendered element) by passing the newly created element to ReactDOM's render method.
For example, lets take a ticking clock example, where it updates the time by calling render method multiple times,

```
function tick() { const element = (
<div>
<h1>Hello, world!</h1>
<h2>It is {new Date().toLocaleTimeString()}.</h2>
</div>
);
ReactDOM.render(element, document.getElementById("root"));
}

setInterval(tick, 1000);
```

263. How do you say that props are readonly?
When you declare a component as a function or a class, it must never modify its own props.

Let us take a below capital function,

```
function capital(amount, interest) { return amount + interest;
}
```

The above function is called "pure" because it does not attempt to change their inputs, and always return the same result for the same inputs. Hence, React has a single rule saying "All React components must act like pure functions with respect to their props."

264. How do you say that state updates are merged?
When you call setState() in the component, React merges the object you provide into the current state.
For example, let us take a facebook user with posts and comments details as state variables,

```
constructor(props) { super(props); this.state = {
posts: [], comments: []
};
}
```

Now you can update them independently with separate setState() calls as below,

```
componentDidMount() { fetchPosts().then(response => {
this.setState({
posts: response.posts
});
});

fetchComments().then(response => { this.setState({
comments: response.comments
});
});
}
```

As mentioned in the above code snippets, this.setState({comments}) updates only comments variable without modifying or replacing posts variable.

265. How do you pass arguments to an event handler?

During iterations or loops, it is common to pass an extra parameter to an event handler. This can be achieved through arrow functions or bind method.
Let us take an example of user details updated in a grid,

```
<button onClick={(e) => this.updateUser(userId, e)}>Update User details</button>
<button onClick={this.updateUser.bind(this, userId)}>Update User details</button>
```

In the both approaches, the synthetic argument e is passed as a second argument. You need to pass it explicitly for arrow functions and it will be passed automatically for bind method.

266. How to prevent component from rendering?
You can prevent component from rendering by returning null based on specific condition. This way it can conditionally render component.

```
function Greeting(props) { if (!props.loggedIn) {
return null;
}

return <div className="greeting">welcome, {props.name}</div>;
}
class User extends React.Component { constructor(props) {
super(props);
this.state = {loggedIn: false, name: 'John'};
}

render() { return (
<div>
//Prevent component render if it is not loggedIn
<Greeting loggedIn={this.state.loggedIn} />
<UserDetails name={this.state.name}>
</div>
);
}
```

In the above example, the greeting component skips its rendering section by applying condition and returning null value.

267. What are the conditions to safely use the index as a key?
There are three conditions to make sure, it is safe use the index as a key.

i. The list and items are static– they are not computed and do not change
ii. The items in the list have no ids
iii. The list is never reordered or filtered.

268. Should keys be globally unique?
The keys used within arrays should be unique among their siblings but they don't need to be globally unique. i.e, You can use the same keys with two different arrays.
For example, the below Book component uses two arrays with different arrays,

```
function Book(props) { const index = (
<ul>
{props.pages.map((page) => (
```

```
<li key={page.id}>{page.title}</li>
))}
</ul>
);
const content = props.pages.map((page) => (
<div key={page.id}>
<h3>{page.title}</h3>
<p>{page.content}</p>
<p>{page.pageNumber}</p>
</div>
));
return (
<div>
{index}
<hr />
{content}
</div>
);
}
```

269. What is the popular choice for form handling?
Formik is a form library for react which provides solutions such as validation, keeping track of the visited fields, and handling form submission.

In detail, You can categorize them as follows,
i. Getting values in and out of form state
ii. Validation and error messages
iii. Handling form submission
It is used to create a scalable, performant, form helper with a minimal API to solve annoying stuff.

270. What are the advantages of formik over redux form library?
Below are the main reasons to recommend formik over redux form library,
i. The form state is inherently short-term and local, so tracking it in Redux (or any kind of Flux library) is unnecessary.
ii. Redux-Form calls your entire top-level Redux reducer multiple times ON EVERY SINGLE KEYSTROKE. This way it increases input latency for large apps.
iii. Redux-Form is 22.5 kB minified gzipped whereas Formik is 12.7 kB

271. Why are you not required to use inheritance?
In React, it is recommended to use composition over inheritance to reuse code between components. Both Props and composition give you all the flexibility you need to customize a component's look and behavior explicitly and safely.
Whereas, If you want to reuse non-UI functionality between components, it is suggested to extract it into a separate JavaScript module. Later components import it and use that function, object, or class, without extending it.

272. Can I use web components in react application?
Yes, you can use web components in a react application. Even though many developers won't use this combination, it may require especially if you are using third-party UI components that are written using Web Components.

For example, let us use Vaadin date picker web component as below,

```
import React, { Component } from "react"; import "./App.css";
import "@vaadin/vaadin-date-picker"; class App extends Component {
render() { return (
<div className="App">
<vaadin-date-picker label="When were you born?"></vaadin-date-picker>
</div>
);
}
}
export default App;
```

273. What is dynamic import?
You can achieve code-splitting in your app using dynamic import. Let's take an example of addition,
i. Normal Import

```
import { add } from "./math"; console.log(add(10, 20));
```

ii. Dynamic Import

```
import("./math").then((math) => { console.log(math.add(10, 20));
});
```

274. What are loadable components?
If you want to do code-splitting in a server rendered app, it is recommend to use Loadable Components because React.lazy and Suspense is not yet available for server-side rendering. Loadable lets you render a dynamic import as a regular component.
Lets take an example,

```
import loadable from "@loadable/component";

const OtherComponent = loadable(() => import("./OtherComponent")); function MyComponent() {

return (
<div>
<OtherComponent />
</div>
);
}
```

Now OtherComponent will be loaded in a separated bundle

275. What is suspense component?
If the module containing the dynamic import is not yet loaded by the time parent component renders, you

must show some fallback content while you're waiting for it to load using a loading indicator. This can be done
using Suspense component.
For example, the below code uses suspense component,

```
const OtherComponent = React.lazy(() => import("./OtherComponent")); function MyComponent() {
return (
<div>
<Suspense fallback={<div>Loading...</div>}>
<OtherComponent />
</Suspense>
</div>
);
}
```

As mentioned in the above code, Suspense is wrapped above the lazy component.

276. What is route based code splitting?
One of the best place to do code splitting is with routes. The entire page is going to re-render at once so users are unlikely to interact with other elements in the page at the same time. Due to this, the user experience won't be disturbed.
Let us take an example of route based website using libraries like React Router with React.lazy,

```
import { BrowserRouter as Router, Route, Switch } from "react-router-dom"; import React, { Suspense, lazy } from "react";

const Home = lazy(() => import("./routes/Home")); const About = lazy(() => import("./routes/About"));

const App = () => (
<Router>
<Suspense fallback={<div>Loading...</div>}>
<Switch>
<Route exact path="/" component={Home} />
<Route path="/about" component={About} />
</Switch>
</Suspense>
</Router>
);
```

In the above code, the code splitting will happen at each route level.

277. Give an example on How to use context?
Context is designed to share data that can be considered global for a tree of React components.
For example, in the code below lets manually thread through a "theme" prop in order to style the Button component.

```
//Lets create a context with a default theme value "luna" const ThemeContext =
React.createContext("luna");
// Create App component where it uses provider to pass theme value in the tree class App extends
React.Component {
render() { return (
<ThemeContext.Provider value="nova">
<Toolbar />
</ThemeContext.Provider>
);
}
}
// A middle component where you don't need to pass theme prop anymore function Toolbar(props) {
return (
<div>
<ThemedButton />
</div>
);
}
// Lets read theme value in the button component to use class ThemedButton extends React.Component {
static contextType = ThemeContext; render() {
return <Button theme={this.context} />;
}

}
```


278. What is the purpose of default value in context?
The defaultValue argument is only used when a component does not have a matching Provider above it in the tree. This can be helpful for testing components in isolation without wrapping them.
Below code snippet provides default theme value as Luna.

```
const MyContext = React.createContext(defaultValue);
```

279. How do you use contextType?
ContextType is used to consume the context object. The contextType property can be used in two ways,
i. contextType as property of class: The contextType property on a class can be assigned a Context object created by React.createContext(). After that, you can consume the nearest current value of that Context type using this.context in any of the lifecycle methods and render function.
Lets assign contextType property on MyClass as below,

```
class MyClass extends React.Component { componentDidMount() {
let value = this.context;
/* perform a side-effect at mount using the value of MyContext */
}
componentDidUpdate() {
let value = this.context;
/* ... */
}
componentWillUnmount() { let value = this.context;
/* ... */
}
render() {
let value = this.context;
/* render something based on the value of MyContext */
}
}
MyClass.contextType = MyContext;
```

ii. Static field You can use a static class field to initialize your contextType using public class field syntax.

```
iii.  class MyClass extends React.Component {
iv.   static contextType = MyContext;
v.    render() {
vi.   let value = this.context;
vii.  /* render something based on the value */
viii. }
}
```

280. What is a consumer?

A Consumer is a React component that subscribes to context changes. It requires a function as a child which receives current context value as argument and returns a react node. The value argument passed to the function will be equal to the value prop of the closest Provider for this context above in the tree.

Lets take a simple example,

```
<MyContext.Consumer>
{value => /* render something based on the context value */}
</MyContext.Consumer>
```

281. How do you solve performance corner cases while using context?

The context uses reference identity to determine when to re-render, there are some gotchas that could trigger unintentional renders in consumers when a provider's parent re-renders.

For example, the code below will re-render all consumers every time the Provider re-renders because a new object is always created for value.

```
class App extends React.Component { render() {
return (
<Provider value={{ something: "something" }}>
<Toolbar />
</Provider>
);
}
}
```

This can be solved by lifting up the value to parent state,

```
class App extends React.Component { constructor(props) {
super(props); this.state = {
value: { something: "something" },
};
}

render() { return (
<Provider value={this.state.value}>
<Toolbar />
</Provider>
);
}
}
```


282. What is the purpose of forward ref in HOCs?

Refs will not get passed through because ref is not a prop. It is handled differently by React just like key. If you add a ref to a HOC, the ref will refer to the outermost container component, not the wrapped component. In this case, you can use Forward Ref API. For example, we can explicitly forward refs to the inner FancyButton component using the React.forwardRef API.

The below HOC logs all props,

```
function logProps(Component) {
class LogProps extends React.Component { componentDidUpdate(prevProps) {
console.log("old props:", prevProps); console.log("new props:", this.props);
}

render() {
const { forwardedRef, ...rest } = this.props;

// Assign the custom prop "forwardedRef" as a ref return <Component ref={forwardedRef} {...rest} />;
}
}

return React.forwardRef((props, ref) => {
return <LogProps {...props} forwardedRef={ref} />;
});
}
```

Let's use this HOC to log all props that get passed to our "fancy button" component,

```
class FancyButton extends React.Component { focus() {
// ...
}

// ...
}
export default logProps(FancyButton);
```

Now let's create a ref and pass it to FancyButton component. In this case, you can set focus to button element.

```
import FancyButton from "./FancyButton";

const ref = React.createRef(); ref.current.focus();
<FancyButton label="Click Me" handleClick={handleClick} ref={ref} />;
```

283. Is ref argument available for all functions or class components?

Regular function or class components don't receive the ref argument, and ref is not available in props either. The second ref argument only exists when you define a component with React.forwardRef call.

284. Why do you need additional care for component libraries while using forward refs?

When you start using forwardRef in a component library, you should treat it as a breaking change and release a new major version of your library. This is because your library likely has a different behavior such as what refs get assigned to, and what types are exported. These changes can break apps and other

libraries that depend on the old behavior.

285. How to create react class components without ES6?

If you don't use ES6 then you may need to use the create-react-class module instead. For default props, you need to define getDefaultProps() as a function on the passed object. Whereas for initial state, you have to provide a separate getInitialState method that returns the initial state.

```
var Greeting = createReactClass({ getDefaultProps: function () {
return {
name: "Jhohn",
};
},
getInitialState: function () {
return { message: this.props.message };
},
handleClick: function () { console.log(this.state.message);
},
render: function () {
return <h1>Hello, {this.props.name}</h1>;
},
});
```

Note: If you use createReactClass then auto binding is available for all methods. i.e, You don't need to use .bind(this) with in constructor for event handlers.

286. Is it possible to use react without JSX?

Yes, JSX is not mandatory for using React. Actually it is convenient when you don't want to set up compilation in your build environment. Each JSX element is just syntactic sugar for calling React.createElement(component, props, ...children).

For example, let us take a greeting example with JSX,

```
class Greeting extends React.Component { render() {
return <div>Hello {this.props.message}</div>;
}
}

ReactDOM.render(
<Greeting message="World" />, document.getElementById("root")
);
```

You can write the same code without JSX as below,

```
class Greeting extends React.Component { render() {
return React.createElement("div", null, `Hello ${this.props.message}`);

}
}

ReactDOM.render(
React.createElement(Greeting, { message: "World" }, null), document.getElementById("root")
);
```


290. Must prop be named as render for render props?

Even though the pattern named render props, you don't have to use a prop named render to use this pattern. i.e, Any prop that is a function that a component uses to know what to render is technically a "render prop". Lets take an example with the children prop for render props,

```
<Mouse
children={(mouse) => (
<p>
The mouse position is {mouse.x}, {mouse.y}
</p>
)}
/>
```

Actually children prop doesn't need to be named in the list of "attributes" in JSX element. Instead, you can keep it directly inside element,

```
<Mouse>
{(mouse) => (
<p>
The mouse position is {mouse.x}, {mouse.y}
</p>
)}
</Mouse>
```

While using this above technique(without any name), explicitly state that children should be a function in your propTypes.

```
Mouse.propTypes = {
children: PropTypes.func.isRequired,
};
```

291. What are the problems of using render props with pure components?

If you create a function inside a render method, it negates the purpose of pure component. Because the shallow prop comparison will always return false for new props, and each render in this case will generate a new value for the render prop. You can solve this issue by defining the render function as instance method.

292. How do you create HOC using render props?

You can implement most higher-order components (HOC) using a regular component with a render prop. For example, if you would prefer to have a withMouse HOC instead of a component, you could easily create one using a regular with a render prop.

```
function withMouse(Component) {
return class extends React.Component { render() {
return (
<Mouse
render={(mouse) => <Component {...this.props} mouse={mouse} />}
/>
);
}
};
}
```

This way render props gives the flexibility of using either pattern.

293. What is windowing technique?
Windowing is a technique that only renders a small subset of your rows at any given time, and can dramatically reduce the time it takes to re-render the components as well as the number of DOM nodes created. If your application renders long lists of data then this technique is recommended. Both react-window and react-virtualized are popular windowing libraries which provides several reusable components for displaying lists, grids, and tabular data.

294. How do you print falsy values in JSX?

The falsy values such as false, null, undefined, and true are valid children but they don't render anything. If you still want to display them then you need to convert it to string. Let's take an example on how to convert to a string,

```
<div>My JavaScript variable is {String(myVariable)}.</div>
```

295. What is the typical use case of portals?
React portals are very useful when a parent component has overflow: hidden or has properties that affect the stacking context (e.g. z-index, position, opacity) and you need to visually "break out" of its container. For example, dialogs, global message notifications, hovercards, and tooltips.

296. How do you set default value for uncontrolled component?
In React, the value attribute on form elements will override the value in the DOM. With an uncontrolled component, you might want React to specify the initial value, but leave subsequent updates uncontrolled. To handle this case, you can specify a defaultValue attribute instead of value.

```
render() { return (
<form onSubmit={this.handleSubmit}>
<label> User Name:
<input
defaultValue="John" type="text" ref={this.input} />
</label>
<input type="submit" value="Submit" />
</form>
);
}
```

The same applies for select and textArea inputs. But you need to use defaultChecked for checkbox and radio inputs.

209. What is the difference between Real DOM and VirtualDOM?

Below are the main differences between Real DOM and Virtual DOM,

Real DOM	Virtual DOM
Updates are slow	Updates are fast
DOM manipulation is very expensive.	DOM manipulation is very easy
You can update HTML directly.	You Can't directly update HTML
It causes too much of memory wastage	There is no memory wastage
Creates a new DOM if element updates	It updates the JSX if element update

299. How to add Bootstrap to a react application?
Bootstrap can be added to your React app in a three possible ways,
i. Using the Bootstrap CDN: This is the easiest way to add bootstrap. Add both bootstrap CSS and JS resources in a head tag.
ii. Bootstrap as Dependency: If you are using a build tool or a module bundler such as Webpack, then this is the preferred option for adding Bootstrap to your React application
npm install bootstrap

iii. React Bootstrap Package: In this case, you can add Bootstrap to our React app is by using a package that has rebuilt Bootstrap components to work particularly as React components. Below packages are popular in this category,
a. react-bootstrap
b. reactstrap

300. Can you list down top websites or applications using react as front end framework?
Below are the top 10 websites using React as their front-end framework,
i. Facebook
ii. Uber
iii. Instagram
iv. WhatsApp
v. Khan Academy
vi. Airbnb
vii. Dropbox
viii. Flipboard
ix. Netflix
x. PayPal

301. Is it recommended to use CSS In JS technique in React?
React does not have any opinion about how styles are defined but if you are a beginner then good starting point is to define your styles in a separate *.css file as usual and refer to them using className. This functionality is not part of React but came from third-party libraries. But If you want to try a different approach(CSS-In-JS) then styled-components library is a good option.

302. Do I need to rewrite all my class components with hooks?

No. But you can try Hooks in a few components(or new components) without rewriting any existing code. Because there are no plans to remove classes in ReactJS.

303. How to fetch data with React Hooks?
The effect hook called useEffect can be used to fetch data from an API and to set the data in the local state of the component with the useState hook's update function.
Here is an example of fetching a list of react articles from an API using fetch.

```
import React from "react"; function App() {
const [data, setData] = React.useState({ hits: [] });

React.useEffect(() => { fetch("http://hn.algolia.com/api/v1/search?query=react")
.then(response => response.json())
.then(data => setData(data))
}, []);

return (
<ul>
{data.hits.map((item) => (
<li key={item.objectID}>
<a href={item.url}>{item.title}</a>
</li>
))}
</ul>
);
}

export default App;
```

A popular way to simplify this is by using the library axios.
We provided an empty array as second argument to the useEffect hook to avoid activating it on component updates. This way, it only fetches on component mount.

304. Is Hooks cover all use cases for classes?

Hooks doesn't cover all use cases of classes but there is a plan to add them soon.
Currently there are no Hook equivalents to the
uncommon getSnapshotBeforeUpdate and componentDidCatch lifecycles yet.

305. What is the stable release for hooks support?
React includes a stable implementation of React Hooks in 16.8 release for below packages
i. React DOM
ii. React DOM Server
iii. React Test Renderer
iv. React Shallow Renderer

306. Why do we use array destructuring (square brackets notation) in useState?
When we declare a state variable with useState, it returns a pair — an array with two items. The first item is the current value, and the second is a function that updates the value. Using [0] and [1] to access them is a bit confusing because they have a specific meaning. This is why we use array destructuring instead.
For example, the array index access would look as follows:

```
var userStateVariable = useState("userProfile"); // Returns an array pair var user = userStateVariable[0]; // Access first item
var setUser = userStateVariable[1]; // Access second item
```

Whereas with array destructuring the variables can be accessed as follows:

```
const [user, setUser] = useState("userProfile");
```

307. What are the sources used for introducing hooks?
Hooks got the ideas from several different sources. Below are some of them,
i. Previous experiments with functional APIs in the react-future repository
ii. Community experiments with render prop APIs such as Reactions Component
iii. State variables and state cells in DisplayScript.

iv. Subscriptions in Rx.
v. Reducer components in ReasonReact.

308. How do you access imperative API of web components?
Web Components often expose an imperative API to implement its functions. You will need to use a ref to interact with the DOM node directly if you want to access imperative API of a web component. But if you are using third-party Web Components, the best solution is to write a React component that behaves as a wrapper for your Web Component.

309. What is formik?
Formik is a small react form library that helps you with the three major problems,
i. Getting values in and out of form state
ii. Validation and error messages
iii. Handling form submission

310. What are typical middleware choices for handling asynchronous calls in Redux?
Some of the popular middleware choices for handling asynchronous calls in Redux eco system are Redux Thunk, Redux Promise, Redux Saga

311. Do browsers understand JSX code?
No, browsers can't understand JSX code. You need a transpiler to convert your JSX to regular Javascript that browsers can understand. The most widely used transpiler right now is Babel.

312. Describe about data flow in react?

React implements one-way reactive data flow using props which reduce boilerplate and is easier to understand than traditional two-way data binding.

313. What is react scripts?
The react-scripts package is a set of scripts from the create-react-app starter pack which helps you kick off projects without configuring. The react-scripts start command sets up the development environment and starts a server, as well as hot module reloading.

314. What are the features of create react app?
Below are the list of some of the features provided by create react app.
i. React, JSX, ES6, Typescript and Flow syntax support.
ii. Autoprefixed CSS
iii. CSS Reset/Normalize
iv. A live development server
v. A fast interactive unit test runner with built-in support for coverage reporting
vi. A build script to bundle JS, CSS, and images for production, with hashes and sourcemaps
vii. An offline-first service worker and a web app manifest, meeting all the Progressive Web App criteria.
332. What is the difference between useState and useRef hook?

i. useState causes components to re-render after state updates whereas useRef doesn't cause a component to re-render when the value or state changes. Essentially, useRef is like a "box" that can hold a mutable value in its (.current) property.
ii. useState allows us to update the state inside components. While useRef allows referencing DOM elements.

333. What is a wrapper component?
A wrapper in React is a component that wraps or surrounds another component or group of components. It can be used for a variety of purposes such as adding additional functionality, styling, or layout to the wrapped components.
For example, consider a simple component that displays a message:


```
const Message = ({ text }) => { return <p>{text}</p>;
};
```
We can create a wrapper component that will add a border to the message component:
```
const MessageWrapper = (props) => { return (
<div style={{ border: "1px solid black" }}>
<Message {...props} />
</div>
);
};
```
Now we can use the MessageWrapper component instead of the Message component and the message will be displayed with a border:
```
<MessageWrapper text="Hello World" />
```
Wrapper component can also accept its own props and pass them down to the wrapped component, for example, we can create a wrapper component that will add a title to the message component:
```
const MessageWrapperWithTitle = (props) => { return (
<div>
<h3>{props.title}</h3>
<Message {...props} />
</div>
);

};
```
Now we can use the MessageWrapperWithTitle component and pass title props:
```
<MessageWrapperWithTitle title="My Message" text="Hello World" />
```
This way, the wrapper component can add additional functionality, styling, or layout to the wrapped component while keeping the wrapped component simple and reusable.

334. What are the differences between useEffect and useLayoutEffect hooks?
useEffect and useLayoutEffect are both React hooks that can be used to synchronize a component with an external system, such as a browser API or a third-party library. However, there are some key differences between the two:
o Timing: useEffect runs after the browser has finished painting, while useLayoutEffect runs synchronously before the browser paints. This means that useLayoutEffect can be used to measure and update layout in a way that feels more synchronous to the user.
o Browser Paint: useEffect allows browser to paint the changes before running the effect, hence it may cause some visual flicker. useLayoutEffect synchronously runs the effect before browser paints and hence it will avoid visual flicker.
o Execution Order: The order in which multiple useEffect hooks are executed is determined by React and may not be predictable. However, the order in which multiple useLayoutEffect hooks are executed is determined by the order in which they were called.
o Error handling: useEffect has a built-in mechanism for handling errors that occur during the execution of the effect, so that it does not crash the entire application. useLayoutEffect does not have this mechanism, and errors that occur during the execution of the effect will crash the entire application.
In general, it's recommended to use useEffect as much as possible, because it is more performant and less prone to errors. useLayoutEffect should only be used

when you need to measure or update layout, and you can't achieve the same result using useEffect.

335. What are the differences between Functional and Class Components?
There are two different ways to create components in ReactJS. The main differences are listed down as below,

1. Syntax:

The classs components uses ES6 classes to create the components. It uses render function to display the HTML content in the webpage.
The syntax for class component looks like as below.

```
class App extends Reacts.Component { render(){
return <h1>This is a class component</h1>}
}
```

Note: The Pascal Case is the recommended approach to provide naming to a component.
Functional component has been improved over the years with some added features like Hooks. Here is a syntax for functional component.

```
function App(){
return <div className="App">
<h1>Hello, I'm a function component</h1>
</div>
}
```

2. State:

State contains information or data about a component which may change over time.
In class component, you can update the state when a user interacts with it or server updates the data using the setState() method. The initial state is going to be assigned in the Constructor( ) method using the the this.state object and it is possible to different data types in the this.state object such as string, boolean,

numbers, etc. A simple example showing how we use the setState() and constructor()

```
class App extends Component { constructor() {
super(); this.state = {
message: "This is a class component",
};
}
updateMessage() { this.setState({t
message: "Updating the class component",
});
}
render() { return (
<>
<h1>{this.state.message}</h1>
<button onClick={() => {
this.updateMessage();
}}>
Click!!
</button>
</>
);
}
}
```

You not use state in functional components because it was only supported in class components. But over the years hooks have been implemented in functional component which enable to use state in functional component too.

The useState() hook can used to implement state in funcitonal component. It returns an array with two items: the first item is current state and the next one is a function (setState) that updates the value of the current state.
Let's see an example to demonstrate the state in functional components,

```
function App() {
const [message, setMessage] = useState("This is a functional component"); const updateMessage = () => {
setCountry("Updating the functional component");
};
return (
<div className="App">
<h1>{message} </h1>
<button onClick={updateMessage}>Click me!!</button>
</div>
);
}
```

4. Props:

Props are referred to as "properties". The props are passed into react component just like arguments passed to a function. In otherwords, they are similar to HTML attributes.
The props are accessible in child class component using this.props as shown in below example,

```
class Child extends React.Component { render() {
return <h1> This is a functional component and component name is
{this.props.name} </h1>;
}
}

class Parent extends React.Component { render() {
return (
<div className="Parent">
<Child name="First child component" />
<Child name="Second child component" />
</div>
);
}
}
```

Props in functional components are similar to that of the class components but the difference is the absence of 'this' keyword.

```
function Child(props) {
return <h1>This is a child component and the component name is{props.name}</h1>;
}

function Parent() { return (
<div className="Parent">
<Child name="First child component" />
<Child name="Second child component" />
</div>
);
}
```