WAD-MODULE-4
PART-B

1 What are the features of React?

ANSWER:

React is a JavaScript Library created by Facebook for creating dynamic and interactive applications and building better UI/UX design for web and mobile applications. React is an open-source and component-based front-end library. React is responsible for the UI design. React makes code easier to debug by dividing them into components.

Features of React:

- JSX (JavaScript Syntax Extension)
- Virtual DOM
- One-way data binding
- Performance
- Extensions
- Conditional statements
- Components
- Simplicity

- **1. JSX(JavaScript Syntax Extension):** JSX is a combination of HTML and JavaScript. You can embed JavaScript objects inside the HTML elements. JSX is not supported by the browsers, as a result Babel compiler transcompile the code into JavaScript code. JSX makes codes easy and understandable. It is easy to learn if you know HTML and JavaScript.

```
const name="GeekforGeeks";
const ele = <h1>Welcome to {name}</h1>;
```

- **2. Virtual DOM:** DOM stands for Document Object Model. It is the most important part of the web as it divides into modules and executes the code. Usually, JavaScript Frameworks updates the whole DOM at once, which makes the web application slow. But react uses virtual DOM which is an exact copy of real DOM. Whenever there is a modification in the web application, the whole virtual DOM is updated first and finds the difference between real DOM and Virtual DOM. Once it finds the difference, then DOM updates only the part that has changed recently and everything remains the same.

- In the above-shown figure, when the whole virtual DOM has updated there is a change in the child components. So, now DOM finds the difference and updates only the changed part.
- **3. One-way Data Binding:** One-way data binding, the name itself says that it is a one-direction flow. The data in react flows only in one direction i.e. the data is transferred from top to bottom i.e. from parent components to child components. The properties(props) in the child component cannot return the data to its parent component but it can have communication with the parent components to modify the states according to the provided inputs. This is the working process of one-way data binding. This keeps everything modular and fast.
- One-way Data Binding
- As shown in the above diagram, data can flow only from top to bottom.
- **4. Performance:** As we discussed earlier, react uses virtual DOM and updates only the modified parts. So , this makes the DOM to run faster. DOM executes in memory so we can create separate components which makes the DOM run faster.
- **5. Extension:** React has many extensions that we can use to create full-fledged UI applications. It supports mobile app development and provides server-side rendering. React is extended with Flux, Redux, React Native, etc. which helps us to create good-looking UI.
- **6. Conditional Statements:** JSX allows us to write conditional statements. The data in the browser is displayed according to the conditions provided inside the JSX.
- **Syntax:**

```
const age = 12;
if (age >= 10)
{
    <p> Greater than { age } </p>;
}
else
{
    <p> { age } </p>;
}
```

- **7. Components:** React.js divides the web page into multiple components as it is component-based. Each component is a part of the UI design which has its own logic and design as shown in the below image. So the component logic which is written in JavaScript makes it easy and run faster and can be reusable.
- Multiple components
- **8. Simplicity:** React.js is a component-based which makes the code reusable and React.js uses JSX which is a combination of HTML and JavaScript. This makes code easy to understand and easy to debug and has less code

2. What is JSX?

ANSWER:

As we have already seen that, all of the React components have a **render** function. The render function specifies the HTML output of a React component. JSX(JavaScript Extension), is a React extension which allows writing JavaScript code that looks like HTML. In other words, JSX is an HTML-like syntax used by React that extends ECMAScript so that **HTML-like** syntax can co-exist with JavaScript/React code. The syntax is used by **preprocessors** (i.e., transpilers like babel) to transform HTML-like syntax into standard JavaScript objects that a JavaScript engine will parse.

JSX provides you to write HTML/XML-like structures (e.g., DOM-like tree structures) in the same file where you write JavaScript code, then preprocessor will transform these expressions into actual JavaScript code. Just like XML/HTML, JSX tags have a tag name, attributes, and children.

Why use JSX?

- It is faster than regular JavaScript because it performs optimization while translating the code to JavaScript.
- Instead of separating technologies by putting markup and logic in separate files, React uses components that contain both. We will learn components in a further section.
- It is type-safe, and most of the errors can be found at compilation time.
- It makes easier to create templates.

JSX Attributes

JSX use attributes with the HTML elements same as regular HTML. JSX uses camelcase naming convention for attributes rather than standard naming convention of HTML such as a class in HTML becomes className in JSX because the class is the reserved keyword in JavaScript. We can also use our own custom attributes in JSX. For custom attributes, we need to use data- prefix. In the below example, we have used a custom attribute data-demoAttribute as an attribute for the <p> tag.

3 Why use React instead of other frameworks, like Angular?

ANSWER:

In summary, React JS is better than Angular or Vue JS because of its superior Virtual DOM capabilities, its robust community support, rich documentation, its light-weight attributes, manageable learning curve, and its flexibility to allow mobile functionality with React Native's.

JavaScript: The Glue Language

To set a precedence for our case justifying React's dominance, we'll recap some basics. JavaScript has often been identified as a glue language, used for assembly of other components. Out of the three web building blocks, HTML and CSS are less complex (though mainly used for static design purposes). However, for developers who create dynamic web content, JavaScript is a priority, which indicates why it's much larger and more complicated than its contemporaries. Essentially, becoming the de facto

standard employed on most web pages.
Consequently, over the last decade, multiple front-end frameworks have been engineered on top of JavaScript to ease the development and maintenance of websites. Resultantly, as of 2020, the most popular of these front-end JavaScript frameworks were React and Vue, then Angular.

Why React is the Dominant Alternative!

In hindsight, React was developed as a JavaScript port of XHP (a PHP library created by Facebook). Generally, XHP was a modification of PHP that allowed for custom component creation, and aimed to prevent malicious user attacks. Later out of this JavaScript porting project grew JSX (JavaScript XML), which became a common standard language for React.

All things considered, React has a narrower scope than its peers, only rendering the application user interface. However, it maintains the benefit of a lightweight structure, and hence is less costly to learn and use. That being said, it can still be considered a framework, as it is used for the same purpose as Vue and Angular

Why React Is Better Than Angular

Let's do a case by case comparison for more context, shall we? Intended for larger application development, Angular is a more full-featured JavaScript framework, in terms of programming features and file size.

However, in comparison to React, Angular has negative bloatedness, complexity and heavy style of development traits. It is recommended for smaller development projects and cited as having a steep learning curve. To exacerbate things, AngularJS is no longer under active development.

Angular is built entirely in TypeScript and requires lots of code which makes it a bit complicated. On the contrary, React is quite "simple" and its original orientation is focused on controlling and manipulating the view. Essentially, it allows the developer to select exactly what they need, but also it avails the ability to use a large number of third-party packages. So, consequently, React doesn't include a lot of overhead in comparison to Angular.

4 What are synthetic events in React?

ANSWER:

In ReactJS, there are events by which user uses to interact with an application UI. React listens to events at the document level, after receiving events from the browser, React wraps these events with a wrapper that has a similar interface as the local browser event, which helps us to use methods like preventDefault().
Why use such a wrapper?
So, often we use different browsers where the same event has different names. Here wrapper does is triggering all

the different names for the same event effect. Therefore, whenever we are triggering an event in a ReactJS, we are not actually trying to trigger the real DOM event, instead, we are using the ReactJS custom event type, which is the synthetic event.
The examples of the synthetic events are onClick(), onBlur() and onChange(). These all are not real DOM events but react synthetic events.
Benefits of using synthetic events:
Cross browsers applications are easy to implement.
Synthetic events are that ReactJS reuses these events objects, by pooling them, which increase the performance.
Your event handlers will be passed instances of `SyntheticEvent`, a cross-browser wrapper around the browser's native event. It has the same interface as the browser's native event, including `stopPropagation()` and `preventDefault()`, except the events work identically across all browsers.
If you find that you need the underlying browser event for some reason, simply use the `nativeEvent` attribute to get it. The synthetic events are different from, and do not map directly to, the browser's native events. For example in `onMouseLeave` `event.nativeEvent` will point to a `mouseout` event. The specific mapping is not part of the public API and may change at any time.
Every `SyntheticEvent` object has the following attributes:
boolean bubbles
boolean cancelable
DOMEventTarget currentTarget
boolean defaultPrevented
number eventPhase
boolean isTrusted
DOMEvent nativeEvent
void preventDefault()
boolean isDefaultPrevented()
void stopPropagation()
boolean isPropagationStopped()
void persist()
DOMEventTarget target
number timeStamp
string type

5 What are keys in React?

ANSWER:

React Keys

A key is a unique identifier. In React, it is used to identify which items have changed, updated, or deleted from the Lists. It is useful when we dynamically created components or when the users alter the lists. It also helps to determine which components in a collection needs to be re-rendered instead of re-rendering the entire set of components every time.
Keys should be given inside the array to give the elements a stable identity. The best way to pick a key as a string that uniquely identifies the items in the list. It can be understood with the below example.

Example

```
const stringLists = [ 'Peter', 'Sachin', 'Kevin', 'Dhoni', 'Alisa' ];

const updatedLists = stringLists.map((strList)=>{
```

- <li key={strList.id}> {strList} ;
- });

Using Keys with Components

Consider a situation where you have created a separate component for list items and you are extracting list items from that component. In that case, you will have to assign keys to the component you are returning from the iterator and not to the list items. That is you should assign keys to <Component /> and not to <li> A good practice to avoid mistakes is to keep in mind that anything you are returning from inside of the map() function is needed to be assigned key.

Below code shows **incorrect usage** of keys:

```
import React from "react";
import ReactDOM from "react-dom";
// Component to be extracted
function MenuItems(props) {
	const item = props.item;
	return <li>{item}</li>;
}

// Component that will return an
// unordered list
function Navmenu(props) {
	const list = props.menuitems;
	const updatedList = list.map((listItems) => {
		return <MenuItems key={listItems.toString()} item={listItems} />;
	});

	return <ul>{updatedList}</ul>;
}

const menuItems = [1, 2, 3, 4, 5];

ReactDOM.render(
	<Navmenu menuitems={menuItems} />,
	document.getElementById("root")
);
```

Uniqueness of Keys

We have told many times while discussing about keys that keys assigned to the array elements must be unique. By this, we did not mean that the keys should be globally unique. All the elements in a particular array should have unique keys. That is, two different arrays can have the same set of keys.

In the below code we have created two different arrays menuItems1 and menuItems2. You can see in the below code that the keys for the first 5 items for both arrays are the same still the code runs successfully without any warning.

```
import React from "react";
import ReactDOM from "react-dom";
// Component to be extracted
function MenuItems(props) {
	const item = props.item;
	return <li>{item}</li>;
}

// Component that will return an
// unordered list
```

```
function Navmenu(props) {
	const list = props.menuitems;
	const updatedList = list.map((listItems) => {
		return <MenuItems key={listItems.toString()} item={listItems} />;
	});

	return <ul>{updatedList}</ul>;
}

const menuItems1 = [1, 2, 3, 4, 5];
const menuItems2 = [1, 2, 3, 4, 5, 6];

ReactDOM.render(
	<div>
		<Navmenu menuitems={menuItems1} />
		<Navmenu menuitems={menuItems2} />
	</div>,
	document.getElementById("root")
);
```

6 What are the differences between functional and class components?

ANSWER:

Functional Components: Functional components are some of the more common components that will come across while working in React. These are simply JavaScript functions. We can create a functional component to React by writing a JavaScript function.

```
import React, { useState } from "react";

const FunctionalComponent=()=>{
	const [count, setCount] = useState(0);

	const increase = () => {
		setCount(count+1);
	}

	return (
		<div style={{margin:'50px'}}>
			<h1>Welcome to Geeks for Geeks </h1>
			<h3>Counter App using Functional Component : </h3>
		<h2>{count}</h2>
			<button onClick={increase}>Add</button>
		</div>
	)
}


export default FunctionalComponent;
```

Class Component: This is the bread and butter of most modern web apps built in ReactJS. These components are simple classes (made up of multiple functions that add functionality to the application).

```
import React, { Component } from "react";

class ClassComponent extends React.Component{
```

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

increase(){
        this.setState({count : this.state.count +1});
}

        render(){
                return (
                        <div style={{margin:'50px'}}>
                        <h1>Welcome to Geeks for Geeks </h1>
                        <h3>Counter App using Class Component : </h3>
                        <h2> {this.state.count}</h2>
                        <button onClick={this.increase}> Add</button>

                        </div>
                )
        }
}

export default ClassComponent;
```

Functional Components	Class Components
A functional component is just a plain JavaScript pure function that accepts props as an argument and returns a React element(JSX).	A class component requires you to extend from React. Component and create a render function which returns a React element.
There is no render method used in functional components.	It must have the render() method returning JSX (which is syntactically similar to HTML)
Functional component run from top to bottom and once the function is returned it cant be kept alive.	Class component is instantiated and different life cycle method is kept alive and being run and invoked depending on phase of class component.
Also known as Stateless components as they simply accept data and display them in some form, that they are mainly responsible for rendering UI.	Also known as Stateful components because they implement logic and state.
React lifecycle methods (for example, componentDidMount) cannot be used in functional components.	React lifecycle methods can be used inside class components (for example, componentDidMount).

Hooks can be easily used in functional components to make them Stateful.
example: const [name,SetName]= React.useState(‘ ‘)
	It requires different syntax inside a class component to implement hooks.
	example: constructor(props) {
	super(props);
	this.state = {name: ‘ ‘}
	}

Constructors are not used.	Constructor are used as it needs to store state.

7 What is the virtual DOM? How does react use the virtual DOM to render the UI?

ANSWER:

DOM: DOM stands for ‘Document Object Model’. In simple terms, it is a structured representation of the HTML elements that are present in a webpage or web-app. DOM represents the entire UI of your application. The DOM is represented as a tree data structure. It contains a node for each UI element present in the web document. It is very useful as it allows web developers to modify content through JavaScript, also it being in structured format helps a lot as we can choose specific targets and all the code becomes much easier to work with. If you want to learn more about DOM, visit this link.
Virtual DOM: React uses Virtual DOM exists which is like a lightweight copy of the actual DOM(a virtual representation of the DOM). So for every object that exists in the original DOM, there is an object for that in React Virtual DOM. It is exactly the same, but it does not have the power to directly change the layout of the document. Manipulating DOM is slow, but manipulating Virtual DOM is fast as nothing gets drawn on the screen. So each time there is a change in the state of our application, the virtual DOM gets updated first instead of the real DOM. You may still wonder, “Aren’t we doing the same thing again and doubling our work? How can this be faster?” Read below to understand how things will be faster using virtual DOM.

How does React use Virtual DOM

Now that you have a fair understanding of what a Virtual DOM is, and how it can help with performance of your app, lets look into how React leverages the virtual DOM.

In React every UI piece is a component, and each component has a state. React follows the observable pattern and listens for state changes. When the state of a component changes, React updates the virtual DOM tree. Once the virtual DOM has been updated, React then compares the current version of the virtual DOM with the previous version of the virtual DOM. This process is called “diffing”.

Once React knows which virtual DOM objects have changed, then React updates only those objects, in the real DOM. This makes the performance far better when compared to manipulating the real DOM directly. This makes React standout as a high performance JavaScript library.

React render() function

render() is where the UI gets updated and rendered. render() is the required lifecycle method in React. You can learn more about React lifecycle methods in detail from my blog post.
render() function is the point of entry where the tree of React elements are created. When

a state or prop within the component is updated, the render() will return a different tree of React elements. If you use setState() within the component, React immediately detects the state change and re-renders the component.
React then figures out how to efficiently update the UI to match the most recent tree changes.

This is when React updates its virtual DOM first and updates only the object that have changed in the real DOM.

- React uses virtual DOM to enhance its performance.
- It uses the observable to detect state and prop changes.
- React uses an efficient diff algorithm to compare the versions of virtual DOM.
- It then makes sure that batched updates are sent to the real DOM for repainting or re-rendering of the UI.

8 What are the differences between controlled and uncontrolled components?
ANSWERS:

Controlled Component

A controlled component is bound to a value, and its changes will be handled in code by using event-based callbacks. Here, the input form element is handled by the react itself rather than the DOM. In this, the mutable state is kept in the state property and will be updated only with setState() method.
Controlled components have functions that govern the data passing into them on every onChange event occurs. This data is then saved to state and updated with setState() method. It makes component have better control over the form elements and data.

Uncontrolled Component

It is similar to the traditional HTML form inputs. Here, the form data is handled by the DOM itself. It maintains their own state and will be updated when the input value changes. To write an uncontrolled component, there is no need to write an event handler for every state update, and you can use a ref to access the value of the form from the DOM.

Difference table between controlled and uncontrolled component

SN	Controlled	Uncontrolled
1.	It does not maintain its internal state.	It maintains its internal states.
2.	Here, data is controlled by the parent component.	Here, data is controlled by the DOM itself.
3.	It accepts its current value as a prop.	It uses a ref for their current values.
4.	It allows validation control.	It does not allow validation control.
5.	It has better control over the form elements and data.	It has limited control over the form elements and data.

9 Explain React state and props.
ANSWER:

- Props are used to pass data, whereas state is for managing data

- Data from props is read-only, and cannot be modified by a component that is receiving it from outside
- State data can be modified by its own component, but is private (cannot be accessed from outside)

State

The state is an updatable structure that is used to contain data or information about the component and can change over time. The change in state can happen as a response to user action or system event. It is the heart of the react component which determines the behavior of the component and how it will render. A state must be kept as simple as possible. It represents the component's local state or information. It can only be accessed or modified inside the component or by the component directly.

Props

Props are read-only components. It is an object which stores the value of attributes of a tag and work similar to the HTML attributes. It allows passing data from one component to other components. It is similar to function arguments and can be passed to the component the same way as arguments passed in a function. Props are immutable so we cannot modify the props from inside the component.

Difference between State and Props

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

10 What is prop drilling in React?

ANSWER:

What are props?

Components in React can be passed some parameters. These parameters are generally named props. There is no hard and fast rule that they should be mentioned as props, but it is convenient to use the same convention.

What is Prop Drilling?

Anyone who has worked in React would have faced this and if not then will face it definitely. Prop drilling is basically a situation when the same data is being sent at almost every level due to requirements in the final level. Here is a diagram to demonstrate it better. Data needed to be sent from Parent to ChildC. In this article different ways to do that are discussed.

Why is Prop Drilling a Problem?

Prop drilling doesn't have to be a problem. If we are passing data only between 2 or 3 levels, we are fine. It will be easy to trace the flow of data. But imagine, we are drilling 5 levels, or 10 levels, or 15.

How to Solve the Problem

Prop drilling is not a new problem in React (quite obviously), and there have been many solutions that let us pass data down to deeply nested Components.

One of which is Redux: You create a data store and connect any component to the store and voila, no matter where the component is positioned in the Component Tree it has access to the store.
React also has the concept of Context which lets you create something like a global data store and any Component in 'context' can have access to the data store.
If you however want to solve this problem without using context, you can use Component Composition as suggested by the **React Docs**: If you only want to avoid passing some props through many levels, component composition is often a simpler solution than context.
You can learn more here **Before You Use Context** and also, check out **Michael Jackson**'s thread on why you should avoid using the Context API.

11 What are error boundaries?
ANSWER:

Error Boundaries: Error Boundaries basically provide some sort of boundaries or checks on errors, They are React components that are used to handle JavaScript errors in their child component tree.
React components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI. It catches errors during rendering, in lifecycle methods, etc.
Reason to Use: Suppose there is an error in JavaScript inside component then it used to corrupt React's internal state and cause it to emit cryptic errors. Error boundaries help in removing these errors and display a Fallback UI instead(Which means a display of an error that something broke in the code).
Working Principle: Error Boundary works almost similar to catch in JavaScript. Suppose an error is encountered then what happens is as soon as there is a broken JavaScript part in Rendering or Lifecycle Methods, It tries to find the nearest Error Boundaries Tag.
Error boundaries do **not** catch errors for:

- Event handlers (learn more)
- Asynchronous code (e.g. `setTimeout` or `requestAnimationFrame` callbacks)
- Server side rendering
- Errors thrown in the error boundary itself (rather than its children)

12 What is React Hooks?expalin it?
ANSWER:

Hooks are the new feature introduced in the React 16.8 version. It allows you to use state and other React features without writing a class. Hooks are the functions which "hook into" React state and lifecycle features from function components. It does not work inside classes.
Hooks are backward-compatible, which means it does not contain any breaking changes. Also, it does not replace

your knowledge of React concepts.

When to use a Hooks

If you write a function component, and then you want to add some state to it, previously you do this by converting it to a class. But, now you can do it by using a Hook inside the existing function component.

Rules of Hooks

Hooks are similar to JavaScript functions, but you need to follow these two rules when using them. Hooks rule ensures that all the stateful logic in a component is visible in its source code. These rules are:

1. Only call Hooks at the top level

Do not call Hooks inside loops, conditions, or nested functions. Hooks should always be used at the top level of the React functions. This rule ensures that Hooks are called in the same order each time a components renders.

2. Only call Hooks from React functions

You cannot call Hooks from regular JavaScript functions. Instead, you can call Hooks from React function components. Hooks can also be called from custom Hooks.

```
import React, { useState } from "react";
import ReactDOM from "react-dom/client";

function FavoriteColor() {
  const [color, setColor] = useState("red");

  return (
    <>
      <h1>My favorite color is {color}!</h1>
      <button
        type="button"
        onClick={() => setColor("blue")}
      >Blue</button>
      <button
        type="button"
        onClick={() => setColor("red")}
      >Red</button>
      <button
        type="button"
        onClick={() => setColor("pink")}
      >Pink</button>
      <button
        type="button"
        onClick={() => setColor("green")}
      >Green</button>
    </>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<FavoriteColor />);
```

13 Explain Strict Mode in React.

ANSWER:

`StrictMode` currently helps with:

- Identifying components with unsafe lifecycles
- Warning about legacy string ref API usage
- Warning about deprecated findDOMNode usage
- Detecting unexpected side effects
- Detecting legacy context API
- Ensuring reusable state

StrictMode is a React Developer Tool primarily used for highlighting possible problems in a web application. It activates additional deprecation checks and warnings for its child components. One of the reasons for its popularity is the fact that it provides visual feedback (warning/error messages) whenever the React guidelines and recommended practices are not followed. Just like the React Fragment, the React StrictMode Component does not render any visible UI.

The React StrictMode can be viewed as a helper component that allows developers to code efficiently and brings to their attention any suspicious code which might have been accidentally added to the application. The StrictMode can be applied to any section of the application, not necessarily to the entire application. It is especially helpful to use while developing new codes or debugging the application.

Example:

Javascript

```
import React from 'react';

function StictModeDemo() {
  return (
    <div>
      <Component1 />
      <React.StrictMode>
        <React.Fragment>
          <Component2 />
          <Component3 />
        </React.Fragment>
      </React.StrictMode>
      <Component4 />
    </div>
  );
}
```

Explanation: In the above example, the StrictMode checks will be applicable only on Component2 and Component3 (as they the child components of React.StrictMode). Contrary to this, Component1 and Component4 will not have any checks.

Advantages: The React StrictMode helps to identify and detect various warnings/errors during the development phase, namely-

Helps to identify those components having unsafe lifecycles: Some of the legacy component lifecycle methods are considered to be unsafe to use in async applications. The React StrictMode helps to detect the use of such unsafe methods. Once enabled, it displays a list of all components using unsafe lifecycle methods as warning messages.

Warns about the usage of the legacy string ref API: Initially, there were two methods to manage refs- legacy string ref API and the callback API. Later, a third alternate method, the createRef API was added, replacing the string refs with object refs, which allowed the StrictMode to give warning messages whenever string refs are used.

Warns about the usage of the deprecated findDOMNode: Since the findDOMNode is only a one-time read API, it is not possible to handle changes when a child component attempts to render a different node (other than the one rendered by the parent component). These issues were detected by the React StrictMode and displayed as warning messages.

14 How to prevent re-renders in React?
ANSWER:

If you're using a React class component you can use the shouldComponentUpdate method or a React.PureComponent class extension to prevent a component from re-rendering. But, is there an option to prevent re-rendering with functional components?

The answer is yes! Use `React.memo()` to prevent re-rendering on React function components.

React `shouldComponentUpdate` is a performance optimization method, and it tells React to avoid re-rendering a component, even if state or prop values may have changed.Only use this method if when a component will stay static or pure.The React `shouldComponentUpdate` method requires you to return a boolean value.Return **true** if it needs to re-render or **false** to avoid being re-render.
These are some tips to avoid too many re-renders errors in React:
Don't change the state in the main body of the component.
Use the useEffect hook very cautiously. The second parameter of useEffect is an array of states based on the useEffect will call. So don't update those states in useEffect otherwise, it will rerender the component again and again.
Use React shouldComponentUpdate: React shouldComponentUpdate is a method for optimizing performance, which tells React to stop re-rendering a component, even though it might have changed the state or prop values. Using this approach only if a part stays unchanged or pure while it is used. You are expected to return a Boolean value with the React shouldComponentUpdate method. Return true if it needs to re-render or false to avoid being re-render.
Filename- App.js: Below is an example of how to use React shouldComponentUpdate. I've built 2 components of React in this code. One is a part of the greeting, and the other is the app component. During the render lifecycle, each React component is a console logging a message.
Javascript

```
import React from "react";
class Greeting extends React.Component {
  render() {
    console.log("Greeting - Render lifecycle");

    return <h1>Geeksforgeeks</h1>;
  }
}

class App extends React.Component {
  render() {
    console.log("App - Render lifecycle");

    return <Greeting />;
  }
}

export default App;
```

Output:

Filename- App.js: Next, in the componentDidMount React lifecycle, I will add the React state, and update the state value.

```
import React from "react";

class Greeting extends React.Component {
  render() {
    console.log("Greeting - Render lifecycle");

    return <h1>Geeksforgeeks</h1>;
  }
}

class App extends React.Component {
  state = {
    greeted: false,
  };

  componentDidMount() {
    this.setState({ greeted: true });
  }

  render() {
    console.log("App - Render lifecycle");

    return <Greeting />;
  }
}

export default App;
```

Output: You can see in the output that the render lifecycle was triggered more than once on both the app and greeting components. This is because the React app component was re-rendered, after updating the state values, and its child components were also re-rendered. We should assume that the greeting portion is unchanged and that it won't ever change.

Filename- App.js: Use the shouldComponentUpdate hook when it is clear that at all times a component we are creating will be static.

```
import React from "react";
class Greeting extends React.Component {
  shouldComponentUpdate() {
    console.log("Greeting - shouldComponentUpdate lifecycle");

    return false;
  }

  render() {
    console.log("Greeting - Render lifecycle");

    return <h1>Geeksforgeeks</h1>;
  }
}

class App extends React.Component {
  state = {
    greeted: false,
  };

  componentDidMount() {
    this.setState({ greeted: true });
  }

  render() {
    console.log("App - Render lifecycle");

    return <Greeting />;
  }
}

export default App;
```

Output: You can see that the app and greeting component went through a round of the rendering lifecycle. After the state values were changed, the App component went through the rendering lifecycle again. But the Greeting component did not console log the Lifecycle Render message.

15 Name a few techniques to optimize React app performance.

ANSWER:

FIVE IMPORTANT WAYS TO OPTIMIZE THE PERFORMANCE OF A REACT APPLICATION, INCLUDING PRE-OPTIMIZATION TECHNIQUES:

- Keeping component state local where necessary

- Memoizing React components to prevent unnecessary re-renders
- Code-splitting in React using dynamic `import()`
- Windowing or list virtualization in React
- Lazy loading images in React

The operations involved in keeping the DOM updates are costly but react uses several techniques to minimize the no. of operations which leads to natively faster UI for many cases. Still, we can use several techniques to speed up the application:

1. Use binding functions in constructors: By adding an arrow function in a class, we add it as an object and not as the prototype property of the class. And if we use the component multiple time, there will be various instances of these functions within each object of the component. The most reliable way to use functions is to bind them with the constructor.

2. Avoid inline style attributes: The browser often invests a lot of time rendering, when styles are implied inline. Scripting and rendering take time because the browser has to plan all the React style rules to the CSS properties. Creating a separate style.js file and importing it into the component is a faster method.

Example: Creating a separate *style.js* file and importing in the component instead of using inline style attribute:

```
import React from "react";
import styles from "./styles.css"

export default class StyleExample extends React.Component {
    render() {
        return (
            <>
                <h1 style=classNameName={styles.head1}>
                    GeeksforGeeks
                </h1>
            </>
        );
    }
}
```

```
.head1 {
    color: green;
    margin: 50;
}
```

3. Avoid extra tags by using React fragments: Using react fragments decreases the no. of additional tags and satisfies the necessity of having a single parent element in the component.

Example: Using react fragments

```
import React from "react";
const App = () => {
    return (
        <div>
            <h1 style={{ color: "green", margin: 50 }}>
                GeeksforGeeks: react fragments as root element
            </h1>
        </div>
    );
};
export default App;
```

4. Avoid inline function in the render method: If we use the inline function, the function will generate a new instance of the object in every render and there will be multiple instances of these functions which will lead to consuming more time in garbage collection. To optimize that we can define functions outside the render method and call them wherever required.
Example: Creating functions outside the render method

```
import React, { useState } from "react";
const App = () => {
    const [isClicked, setIsclicked] = useState(false);
    const handleClicked = () => {
        setIsclicked(true);
        console.log(`clicked ${isClicked}`);
    };
    return (
        <div style={{ margin: 50 }}>
            <h1 style={{ color: "green" }}>
                GeeksforGeeks: function outside render example
            </h1>
            <button onClick={handleClicked}>Click me!</button>
        </div>
    );
};
export default App;
```

- **Avoid bundling all of the front end code in a single file:** By splitting the files into resource and on-demand code files we can reduce the time consumed in presenting bundled files to the browser transformers.

16 What are the different phases of the component lifecycle?

ANSWER:

Lifecycle of Components

Each component in React has a lifecycle which you can monitor and manipulate during its three

main phases.

The three phases are: Mounting, Updating, and Unmounting.

Lifecycle methods: In ReactJS, the development of each component involves the use of different lifecycle methods. All the lifecycle methods used to create such components, together constitute the component's lifecycle. They are defined as a series of functions invoked in various stages of a component. Each phase of the lifecycle components includes some specific lifecycle methods related to that particular phase. There are primarily 4 phases involved in the lifecycle of a reactive component, as follows.
Initializing
Mounting
Updating
Unmounting
Phase 1: Initializing
This is the initial phase of the React component lifecycle. As the name suggests, this phase involves all the declarations, definitions, and initialization of properties, default props as well as the initial state of the component required by the developer. In a class-based component, this is implemented in the constructor of the component. This phase occurs only once throughout the entire lifecycle. The methods included in this phase are:
getDefaultProps(): The method invoked immediately before the component is created or any props from the parent component are passed into the said (child) component. It is used to specify the default value of the props.
getInitialState(): The method invoked immediately before the component is created and used to specify the default value of the state.
Phase 2: Mounting
The second phase of the React component lifecycle, followed by the initialization phase, is the mounting phase. It commences when the component is positioned over the DOM container(meaning, an instance of the component is created and inserted into the DOM) and rendered on a webpage. It consists of 2 methods, namely:
componentWillMount(): The method invoked immediately before the component is positioned on the DOM, i.e. right before the component is rendered on the screen for the very first time.
componentDidMount(): The method invoked immediately after the component is positioned on the DOM, i.e. right after the component is rendered on the screen for the very first time.
Phase 3: Updating
The third phase of the ReactJS Component Lifecycle is the Updation phase. Followed by the mounting phase, it updates the states and properties that were declared and initialized during the initialization phase (if at all any changes are required). It is also responsible for handling user interaction and passing data within the component hierarchy. Unlike the initialization phase, this phase can be repeated multiple times. Some of the lifecycle methods falling into this category are as follows:
componentWillReceiveProps(): The method invoked immediately before the props of a mounted component get reassigned. It accepts new props which may/may not be the same as the original props.
shouldComponentUpdate(): The method invoked before deciding whether the newly rendered props are required to be displayed on the webpage or not. It is useful in those scenarios when there is such a requirement not to display the new props on the screen.
componentWillUpdate(): The method invoked immediately before the component is re-rendered after updating the states and/or properties.
componentDidUpdate(): The method invoked immediately after the component is re-rendered after updating the states and/or properties.
Phase 4: Unmounting
Unmounting is the last phase of the ReactJS component lifecycle. This phase includes those lifecycle methods which are used when a component is getting detached from the DOM container(meaning, the instance of the component being destroyed and unmounted from the DOM). It is also responsible for performing the required cleanup tasks.

Once unmounted, a component can not be re-mounted again.
componentWillUnmount(): The method invoked immediately before the component is removed from the DOM at last, i.e. right when the component is completely removed from the page and this shows the end of its lifecycle

17 What are the lifecycle methods of React?

ANSWER:

Refer Q16.Part-B

18 What is React Router?

ANSWER:

React Router

Routing is a process in which a user is directed to different pages based on their action or request. ReactJS Router is mainly used for developing Single Page Web Applications. React Router is used to define multiple routes in the application. When a user types a specific URL into the browser, and if this URL path matches any 'route' inside the router file, the user will be redirected to that particular route.
React Router is a standard library system built on top of the React and used to create routing in the React application using React Router Package. It provides the synchronous URL on the browser with data that will be displayed on the web page. It maintains the standard structure and behavior of the application and mainly used for developing single page web applications.

Need of React Router

React Router plays an important role to display multiple views in a single page application. Without React Router, it is not possible to display multiple views in React applications. Most of the social media websites like Facebook, Instagram uses React Router for rendering multiple views.

React Router Installation

React contains three different packages for routing. These are:

- react-router: It provides the core routing components and functions for the React Router applications.
- react-router-native: It is used for mobile applications.
- react-router-dom: It is used for web applications design.

It is not possible to install react-router directly in your application. To use react routing, first, you need to install react-router-dom modules in your application. The below command is used to install react router dom.

- $ npm install react-router-dom --save

Components in React Router

There are two types of router components:

- <BrowserRouter>: It is used for handling the dynamic URL.
- <HashRouter>: It is used for handling the static request.

Let us add some code to our 3 components:

```
import React from 'react';

function Home (){
    return <h1>Welcome to the world of Geeks!</h1>
}

export default Home;
```

```
import React from 'react';

function About () {
    return <div>
        <h2>GeeksforGeeks is a computer science portal for geeks!</h2>

        Read more about us at :
        <a href="https://www.geeksforgeeks.org/about/">
            https://www.geeksforgeeks.org/about/
        </a>
    </div>
}
export default About;
```

```
import React from 'react';

function Contact (){
 return <address>
            You can find us here:<br />
            GeeksforGeeks<br />
            5th & 6th Floor, Royal Kapsons, A- 118, <br />
            Sector- 136, Noida, Uttar Pradesh (201305)
        </address>
}

export default Contact;
```

Now, let us include React Router components to the application:

BrowserRouter: Add BrowserRouter aliased as Router to your app.js file in order to wrap all the other components. BrowserRouter is a parent component and can have only single child.

```
class App extends Component {
  render() {
    return (
        <Router>
            <div className="App">
            </div>
        </Router>
    );
  }
}
```

Link: Let us now create links to our components. Link component uses the **to** prop to describe the location where the links should navigate to.

```
<div className="App">
    <ul>
      <li>
        <Link to="/">Home</Link>
      </li>
      <li>
        <Link to="/about">About Us</Link>
      </li>
      <li>
        <Link to="/contact">Contact Us</Link>
      </li>
    </ul>
</div>
```

Now, run your application on the local host and click on the links you created. You will notice the url changing according the value in **to** props of the Link component.

Route: Route component will now help us to establish the link between component's UI and the URL. To include routes to the application, add the code give below to your app.js.

```
<Route exact path='/' element={< Home />}></Route>
<Route exact path='/about' element={< About />}></Route>
<Route exact path='/contact' element={< Contact />}></Route>
```

19 Can React Hook replaces Redux?

ANSWER:

Redux and **React hooks** should not be seen as the opposite of each other. They are different things, with distinct goals.When **React Hooks** was born, several developers seemed confused by the concepts introduced and how they will interpolate into the **Redux** library. The **useReducer** hook increased this confusion.

React Hooks is the new way of handling state and life cycle into React components, without relying on component classes. It was introduced in the 16.8 version of the library and has the intention to decrease the complexity of the components, by sharing logic between them.

Redux is a library for managing the global application state. In this library, we can find several tools that help us, developers, to be in touch with the state of the application and also transform it by giving the user the ability to emit actions.

Redux, as the documentation says, can be described in three fundamental principles:

- Single source of truth: the global state of your application is stored in an object tree within a single store.
- The State is read-only: The only way to change the store is by emitting actions.
- Changes are made with pure functions: To update the store, the reducer should be written as a pure function.

Redux even updated the library with its custom hooks. These can be used to integrate the components that use the React Hooks features to access data from the store and dispatch actions without relying on the components classes.

Can useReducer replace Redux?

The useReducer hook should be used in components that have complex logic behind it. It shows as the main confusion with the Redux library, because developers tend to think that useReducer could replace the state manager library. But in fact, its use should be restricted to components. This is mainly because the global use of useReducer demands rewrites of new code to features like the control of components update.

20 How to pass data between sibling components using React router?

ANSWER:

React is a JavaScript library created by Facebook. Data handling in React could be a bit tricky, but not as complicated as it might seem. I have currently compiled three methods of Data Handling in React :-

- From Parent to Child using Props
- From Child to Parent using Callbacks
- Between Siblings :
  (i) Combine above two methods
  (ii) Using Redux
  (iii) Using React's Context API

- First we need to import `useContext` from react.
- `import React, { useContext } from 'react';`

- Now we need to import AppContext from App component
- `import { AppContext } from '../App'`

- To use our state values inside component we need to add
- `const {state, dispatch} = useContext(AppContext);`

- Next add function, which will update state `inputText` value using `dispatch`. This function will be called each time, when we type something in any input.
- `const changeInputValue = (newValue) => {`

```
    dispatch({ type: 'UPDATE_INPUT', data: newValue,});
};
```

- Now we need to set up *value* to *state.inputText* and *onChange* event should call function `changeInputValue`.

- Now `./src/components/Input_one.js` should looks like:
-
```
import React, { useContext } from 'react';
import {makeStyles} from "@material-ui/core/styles/index";

//Material UI components
import Grid from '@material-ui/core/Grid';
import Paper from '@material-ui/core/Paper';
import InputBase from '@material-ui/core/InputBase';

//Material UI Icons
import IconButton from '@material-ui/core/IconButton';
import SearchIcon from '@material-ui/icons/Search';

// Import Context
import { AppContext } from '../App'

const useStyles = makeStyles({
    root: {
        padding: '2px 4px',
        display: 'flex',
        alignItems: 'center',
        width: '100%',
    },
    input: {
        marginLeft: '8px',
        flex: 1,
    },
    iconButton: {
        padding: 10,
    }
});

export default function Input_one() {

    const classes = useStyles();

    const {state, dispatch} = useContext(AppContext);

    const changeInputValue = (newValue) => {

        dispatch({ type: 'UPDATE_INPUT', data: newValue,});
    };

    return(
        <React.Fragment>
```

```
            <Grid item xs={12} md={6}>
                <Paper className={classes.root}>
                    <InputBase
                        className={classes.input}
                        placeholder="Input one"
                        value={state.inputText}
                        onChange={e => changeInputValue(e.target.value)}
                    />
                    <IconButton className={classes.iconButton} aria-
label="search">
                        <SearchIcon />
                    </IconButton>
                </Paper>
            </Grid>
        </React.Fragment>
    )
}
```

- And `./src/components/Input_two.js` should looks like:
- ```
  import React, { useContext } from 'react';
  import {makeStyles} from "@material-ui/core/styles/index";

  //Material UI components
  import Grid from '@material-ui/core/Grid';
  import Paper from '@material-ui/core/Paper';
  import InputBase from '@material-ui/core/InputBase';

  //Material UI Icons
  import IconButton from '@material-ui/core/IconButton';
  import SearchIcon from '@material-ui/icons/Search';


  //Import Context
  import { AppContext } from '../App'

  const useStyles = makeStyles({
      root: {
          padding: '2px 4px',
          display: 'flex',
          alignItems: 'center',
          width: '100%',
      },
      input: {
          marginLeft: '8px',
          flex: 1,
      },
      iconButton: {
          padding: 10,
      }
  });
  ```
  ```

```
export default function Input_two() {

    const classes = useStyles();

    const {state, dispatch} = useContext(AppContext);

    const changeInputValue = (newValue) => {

        dispatch({ type: 'UPDATE_INPUT', data: newValue,});
    };

    return(
        <React.Fragment>
            <Grid item xs={12} md={6}>
                <Paper className={classes.root}>
                    <InputBase
                        className={classes.input}
                        placeholder="Input two"
                        value={state.inputText}
                        onChange={e => changeInputValue(e.target.value)}
                    />
                    <IconButton className={classes.iconButton} aria-
label="search">
                        <SearchIcon />
                    </IconButton>
                </Paper>
            </Grid>
        </React.Fragment>
    )
}
```

- Now, type something in any input fields and you will see that both inputs will be updated and will contain the same values.
-