**REACT JS Interview question**

**1. What is React?**

React is a front-end and open-source JavaScript library which is useful in developing user interfaces specifically for applications with a single page. It is helpful in building complex and reusable user interface(UI) components of mobile and web applications as it follows the component-based approach.

The important features of React are:

* It supports server-side rendering.
* It will make use of the virtual DOM rather than real DOM (Data Object Model) as RealDOM manipulations are expensive.
* It follows unidirectional data binding or data flow.
* It uses reusable or composable UI components for developing the view.

What is main Diffrence and why should be use react instead js ?

React (Declarative Approach):

 **What to Do**: In React, you focus on describing *what* the UI should look like or *what* it should do, and React handles the details of *how* to update the DOM and manage the application state.

 **Abstraction**: React abstracts away the complexity of directly manipulating the DOM. You define your UI components and their state, and React takes care of the underlying implementation details.

**Plain JavaScript (Imperative Approach):**

* **How to Do It**: In plain JavaScript, you need to explicitly manage the DOM and specify the sequence of operations to achieve the desired outcome. This involves directly interacting with DOM elements and handling updates manually.

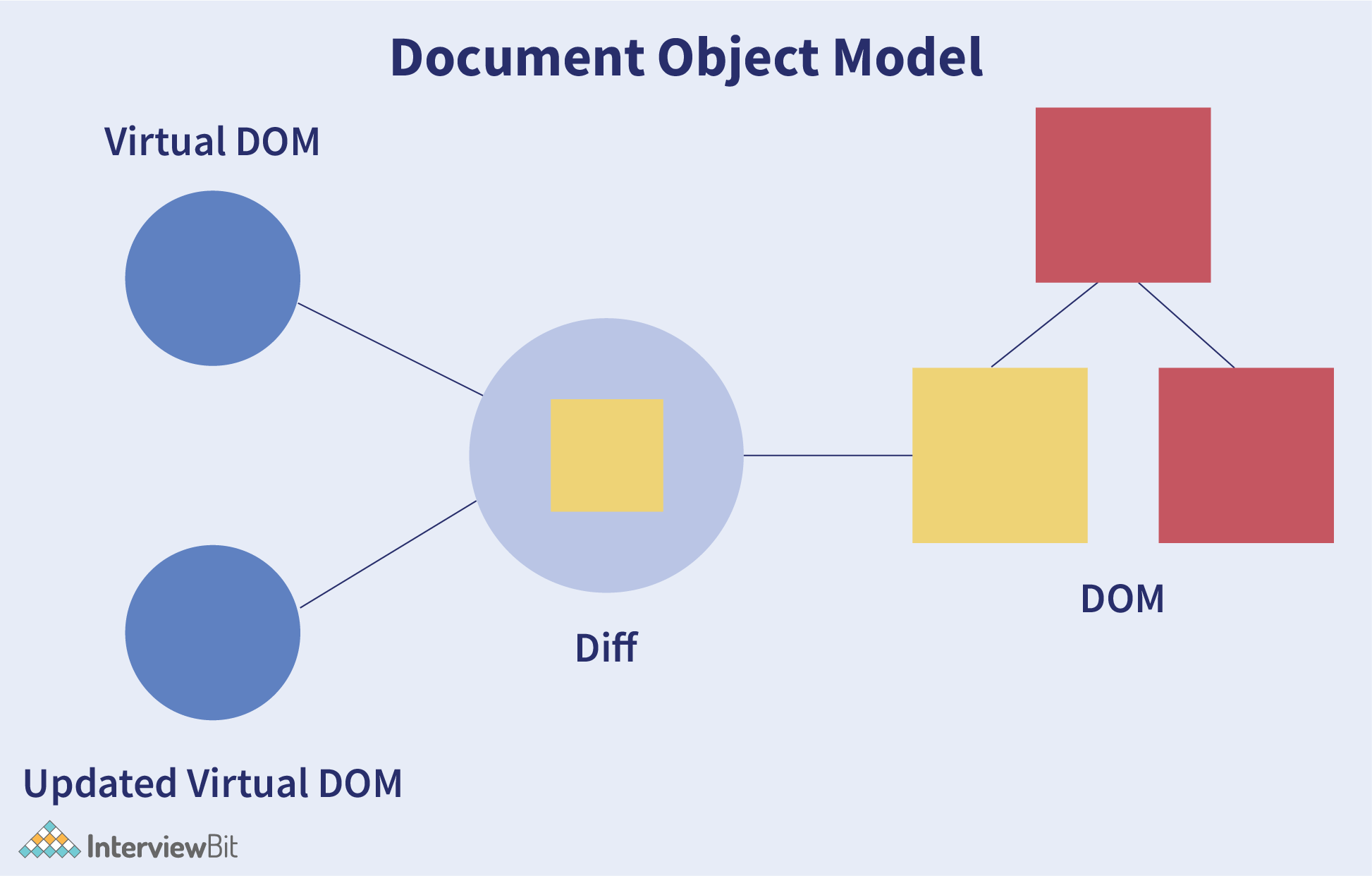
**Why React's Declarative Approach Is Beneficial:**

1. **Simplicity**:
   * You describe *what* you want to achieve without worrying about the step-by-step process of manipulating the DOM.
2. **Readability**:
   * Code is often more readable and maintainable because it abstracts away the complex details of DOM updates.
3. **Maintainability**:
   * As the application grows, maintaining and updating the UI becomes easier with React's declarative approach.
4. **Efficiency**:
   * React's virtual DOM optimizes updates, minimizing the number of changes to the real DOM and improving performance.
5. **Predictability**:
   * React's component-based architecture and state management make it easier to predict and manage UI changes.
6. **Virtual Dom:**
   * React’s virtual DOM and reconciliation algorithm optimize performance by reducing the number of direct DOM manipulations.

In React, you focus on describing the UI and its behavior in a declarative manner, while React handles the low-level details of updating the DOM. In contrast, traditional JavaScript requires you to manually specify how to manipulate the DOM, which can be more complex and error-prone. React's approach simplifies development and enhances maintainability by abstracting away these implementation details.

**2. What is virtual Dom?**

As stated by the react team, virtual DOM is a concept where a virtual representation of the real DOM is kept inside the memory and is synced with the real DOM by a library such as ReactDOM.



**Why was virtual DOM introduced?**

DOM manipulation is an integral part of any web application, but DOM manipulation is quite slow when compared to other operations in JavaScript. The efficiency of the application gets affected when several DOM manipulations are being done. Most JavaScript frameworks update the entire DOM even when a small part of the DOM changes.

For example, consider a list that is being rendered inside the DOM. If one of the items in the list changes, the entire list gets rendered again instead of just rendering the item that was changed/updated. This is called inefficient updating.

To address the problem of inefficient updating, the react team introduced the concept of virtual DOM.

**What is Reconciliation ?**

React uses reconciliation to efficiently update the virtual DOM and sync it with the real DOM, ensuring that only the necessary changes are applied. This process helps in optimizing performance and maintaining a smooth user experience.

**How Reconciliation Works**

1. **Virtual DOM Representation**:
   * React maintains a virtual DOM, a lightweight in-memory representation of the real DOM. When a component’s state or props change, React creates a new virtual DOM tree that reflects these updates.
2. **Diffing Algorithm**:
   * React uses a diffing algorithm to compare the new virtual DOM tree with the previous one. This algorithm identifies which parts of the virtual DOM have changed and need to be updated in the real DOM.
3. **Updating the Real DOM**:
   * Based on the diffing results, React updates only the changed parts of the real DOM. This selective update minimizes the number of direct manipulations to the DOM, which can be costly in terms of performance.

**What is Diffing ?**

React compares the newly created virtual DOM tree with the previous one to determine which elements have changed. It uses a heuristic approach to efficiently perform this comparison.

* **Reconciliation**: The entire process of updating the user interface, including rendering new virtual DOM trees, diffing, and updating the real DOM.
* **Diffing**: A specific algorithm used during reconciliation to compare virtual DOM trees and determine what has changed.

**What is React Elements ?**

A React element is a simple object that contains information about what to render. It includes the type of element (e.g., div, span, MyComponent), its properties (props), and any children elements.

React elements are the smallest building blocks of React apps. They describe what the UI should look like at any point in time.

**Why we need to wrap the elements into parent element ?**

JSX, the syntax used in React to describe what the UI should look like, requires that a component return a single root element. This means that all the elements you want to render must be wrapped in a single parent element.

By doing this we ensure that the component rendered is a single HTML element, as React expects its components to return a single element.

**Why React component is returning only one ?**

**Component as a Function**: React components can be thought of as functions that return UI elements. Just as a function in programming has a single return value, a React component must return a single root element.

JSX, the syntax used in React, is transpiled into JavaScript code that calls React.createElement. This function creates React elements, and it expects a single root element for each component.

It helps in Recoiliation and effective performance.

**Without Using Div and Fragement can return multiple elements ?**

**1. Arrays**

You can return an array of elements directly by using comma Saparated. This approach is less common and may not be as clean as using Fragments, but it is a valid method.

function MyComponent() {

return [

<h1 key="1">Hello</h1>,

<p key="2">World</p>

];

}

* **Note**: Each element in the array must have a unique key prop to help React identify and manage the elements properly.
* **Since it is array it is not shown on the Dom**.

**What is Components ?**

a **component** is a fundamental building block that defines a piece of the user interface. Components are indeed functions (or classes) that return a collection of HTML elements (or more accurately, React elements) to describe what the UI should look like.

## 3. **Differences between props and state**

| **PROPS** |  | **STATE** |
| --- | --- | --- |
| The Data is passed from one component to another. |  | The Data is passed within the component only. |
| It is Immutable (cannot be modified). |  | It is Mutable ( can be modified). |
| Props can be used with state and functional components. |  | The state can be used only with the state components/class component (Before 16.0). |
| Props are read-only. |  | The state is both read and write. |

Props :-

Props, short for properties, are a mechanism for passing read-only data from a parent component to a child component in React.js. They are immutable, meaning they cannot be modified within the child component. Props are set by the parent component and are received as parameters by the child component. They allow components to be customizable and reusable by passing different data to them based on their usage.

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.

State in React.js is like a container that holds information specific to a component. It's like a snapshot of how things are inside that component at a given moment. You can think of it as the memory of the component. When something changes in a component, like a user clicking a button or typing in a form, we can update the state to reflect that change. React then automatically re-renders the component to show the new state. So, state is basically what makes a component dynamic and interactive.

## **4) What is React Lifecycle ?**

**React Lifecycle** is defined as the series of methods that are invoked in different stages of the component’s existence.

## Lifecycle of React Components:

Each React Component go though the given Phases.

### ****1. Initialization phase****

This is the stage where the component is constructed with the given Props and default state. This is done in the constructor of a Component Class.

### ****2. Mounting Phase****

Mounting is the stage of rendering the JSX returned by the render method itself.

In this phase the component is created and mounted to the DOM.

* **constructor():** This is the first method called when a component is instantiated. It's used for initializing state and binding event handlers.
* **render():** This method is responsible for rendering the component's JSX or returning null. It's a required method in class components.
* **componentDidMount():** Invoked after the component is mounted to the DOM. It's commonly used for fetching data from a server or setting up subscriptions.

### ****3. Updating****

Updating is the stage when the state of a component is updated and the application is repainted.

* **shouldComponentUpdate(nextProps, nextState):** This method determines whether the component should re-render when its props or state change. It returns a boolean value indicating whether the update should proceed. By default, it returns true.
* **render():** Re-renders the component when its props or state change.
* **componentDidUpdate(prevProps, prevState):** Invoked after the component's update has been flushed to the DOM. It's used for performing tasks based on the changes to props or state.

### ****4. Unmounting****

As the name suggests Unmounting is the final step of the component lifecycle where the component is removed from the page.

* **componentWillUnmount**(): Called just before the component is unmounted from the DOM. It's used for cleanup tasks such as unsubscribing from event listeners or cancelling network requests.

**5) What is Keys in ReactJs and Why It is important ?**

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.

In React, the key prop is a special attribute that you need to include when creating lists of elements. Each child in a list should have a unique key prop assigned to it. The purpose of the key prop is to help React identify which items have changed, are added, or are removed.

Real-life Example: To-Do List

It is important because :

* Preventing Unintended Behavior:

Without keys, React may not be able to distinguish between different items in the list, leading to unintended behavior.

For example, if a key is not unique, React may incorrectly identify two items as the same and apply changes incorrectly.

* Optimizing Performance:

Using keys optimizes the performance of your React application by minimizing unnecessary DOM manipulations.

**6) What is Context API ?**

It can be annoying to pass props when you have to send the same data to lots of components or when desired components are far away from each other or from parent component. This can make an application slower and harder to work with.

Fortunately, React provides a built-in feature known as the context API that helps  “teleport” data to the components that need it without passing props to intermediate components.

Basically, Context API consists of two main components: the context provider and the context consumer. The provider is responsible for creating and managing the context, which holds the data to be shared between components. On the other hand, the consumer is used to access the context and its data from within a component. In the example given, the provider will create the context that holds the user's shopping information, while the consumer components (shopping cart and order history) will access that context to retrieve the data they need.

Example :- With Context API, you can create a “context” that holds the user’s shopping information, like their cart and order history. Then, you can use that context in both the shopping cart and the order history component, without having to pass the information down through props.

It’s like having a big box that holds all the things you need for your shopping trip. You can take things out of the box when you need them, and put them back in when you’re done.

**7) What is children prop?**

*Children* is a prop 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.

A simple usage of children prop looks as below,

function MyDiv({ children }){

return (

<div>

{children}

</div>;

);

}

export default function Greeting() {

return (

<MyDiv>

<span>{"Hello"}</span>

<span>{"World"}</span>

</MyDiv>

);

}

**8) What is fragments in reactJS ? Why it is used ?**

React Fragments provide a way to group multiple React elements without adding an extra DOM element. They allow us to return multiple elements from a component's render method without needing to wrap them in a container element like a div. This is particularly useful when we want to maintain a clean and semantic structure in our JSX code.

In essence, Fragments are lightweight and don't add any extra nodes to the DOM, which can improve performance and keep the rendered output cleaner. They're especially handy when we need to render lists or collections of elements.

In React, the render method is essential for displaying content on the screen. When rendering multiple elements, we encounter a restriction: the render method accepts only a single root node. To circumvent this, we typically wrap multiple elements in a parent tag / div tag, ensuring compliance with this requirement. In this Case div takes extra space by creating new node in Dom which is not space efficient.

However, an optimized approach involves omitting the div tag entirely. By using <></>, or its shorthand form <> and </>, known as React.Fragment, we achieve the same result without introducing additional DOM nodes. This strategy enhances performance marginally by reducing memory overhead and rendering speed.

**9) What is stateless Components ?**

Stateless components are JavaScript functions that return JSX (JavaScript XML) to describe the structure and appearance of the component. Unlike stateful components, they do not have an internal state and do not manage any data. Instead, they receive data and behavior through props (properties) passed down from parent components.

The primary advantage of stateless components is their simplicity and reusability. They are primarily responsible for presenting data and rendering the UI based on the received props. Stateless components focus on the presentational aspect of the application, without managing any state or handling lifecycle methods.

Stateless components are a fundamental concept in React, providing a simple and efficient way to create UI components that are purely based on props. By following best practices and leveraging the power of React Hooks, stateless components can be used to build robust and maintainable React applications.

Sure, let's create a detailed example of a stateless component in React. We'll create a simple UserCard component that receives user data as props and renders a card displaying the user's name, email, and avatar.

Here's how we can implement it:

// UserCard.jsx

In this example:

import React from 'react';

const UserCard = ({ user }) => {

  return (

    <div className="user-card">

      <img src={user.avatar} alt={user.name} />

      <div className="user-info">

        <h2>{user.name}</h2>

        <p>{user.email}</p>

      </div>

    </div>

  );

};

export default UserCard;

We define a functional component named UserCard.

It receives a user object as a prop.

Inside the component, we use JSX to render the user's avatar, name, and email.

We return a <div> element containing the user's information.

Now, let's use this UserCard component in another component:

In this example:

import React from 'react';

import UserCard from './UserCard';

import './App.css'; // Assuming we have some CSS for styling

const App = () => {

  // Sample user data

  const user = {

    name: 'John Doe',

    email: 'john@example.com',

    avatar: 'https://example.com/avatar.jpg'

  };

  return (

    <div className="app">

      <h1>User Profile</h1>

      <UserCard user={user} />

    </div>

  );

};

export default App;

We import the UserCard component.

We define a functional component named App.

Inside the App component, we create a sample user object with name, email, and avatar properties.

We render the UserCard component passing the user object as a prop.

Now, when App component renders, it will display the user's profile using the UserCard component. This UserCard component is stateless because it doesn't manage any internal state; it simply receives data via props and renders UI based on that data.

**10) What are the advantages in ReactJS ?**

* Virtual DOM: React uses a virtual DOM, a lightweight representation of the actual DOM. This allows React to efficiently update and render UI components by minimizing direct manipulation of the DOM, resulting in improved performance.
* Component-Based Architecture: React follows a component-based architecture, where UIs are broken down into reusable and independent components. This modular approach simplifies development, encourages code reusability, and enhances maintainability.
* It provides Single page application

**11) Explain the difference between functional and class component in React?**

| **Functional Components** | **Class Components** |
| --- | --- |
| A functional component is just a plain JavaScript pure function that accepts props as an argument | A class component requires you to extend from React. Component and create a render function |
| No render method used | It must have the render() method returning JSX |
| Also known as Stateless components | Also known as Stateful components |
| React lifecycle methods (for example, componentDidMount) cannot be used in functional components. | React lifecycle methods can be used inside class components (for example, componentDidMount). |
| Constructors are not used. | Constructor is used as it needs to store state. |

### ****12) Explain one way data binding in React?****

ReactJS uses one-way[data binding](https://www.geeksforgeeks.org/reactjs-data-binding/) which can be Component to View or View to Component. It is also known as one-way data flow, which means the data has one, and only one way to be transferred to other parts of the application. In essence, this means child components are not able to update the data that is coming from the parent component. It is easy to debug and less prone to errors.

### ****13) What is react router?****

[React Router](https://www.geeksforgeeks.org/reactjs-router/) is a standard library for routing in React. It enables the navigation among views of various components in a React Application, allows changing the browser URL, and keeps the UI in sync with the URL.

**14) 1What is Conditional Rendering :-**

React allows us to conditionally render components which means that the developer can decide which component to render on the screen on on the basis of some predefined conditions. This is known as **conditional rendering**.

import React, { useState } from 'react';

const ConditionalComponent = () => {

  // Define a state variable to store the condition

  const [condition, setCondition] = useState(true);

  // Function to toggle the condition

  const toggleCondition = () => {

    setCondition(!condition);

  };

  return (

    <div>

      <button onClick={toggleCondition}>Toggle Condition</button>

      {condition ? <p>Condition is true</p> : <p>Condition is false</p>}

    </div>

  );

};

export default ConditionalComponent;

There may arise a situation when we want to render something based on some condition. For example, consider the situation of handling a login/logout button. Both the buttons have different functions so they will be separate components. Now, the task is if a user is logged in then we will have to render the Logout component to display the logout button and if the user is not logged in then we will have to render the Login component to display the login button.

**15) What is Hooks ?**

React Hooks are functions that enable functional components to use state, lifecycle methods, and other React features without writing a class. They were introduced in React 16.8 as a way to simplify the development of functional components and promote code reuse.

* UseStatew
* UseEffect
* Usecontext
* useCallback
* useMemo
* useRef

### ****16) What is custom hooks in React?****

[Custom hooks](https://www.geeksforgeeks.org/reactjs-custom-hooks/) are normal JavaScript functions whose names start with “use” and they may call other hooks. We use custom hooks to maintain the DRY concept that is Don’t Repeat Yourself. It helps us to write a logic once and use it anywhere in the code.

**17) Explain CORS in React?**

In ReactJS, [Cross-Origin Resource Sharing (CORS)](https://www.geeksforgeeks.org/reactjs-cors-options/) refers to the method that allows you to make requests to the server deployed at a different domain. As a reference, if the frontend and backend are at two different domains, we need CORS there.

We can setup CORS evironment in frontend using two methods:

* axios
* fetch

**18) What is Axios :-**

Axios is a popular JavaScript library used for making HTTP requests from both the browser and Node.js. It provides an easy-to-use API with features like request and response interception, handling request and response data in various formats, and support for promises.

Axios is used to make GET request in react components.

**19) What is babel?**

Babel is a very famous transpiler that basically allows us to use future JavaScript in today’s browsers. In simple words, it can convert the latest version of JavaScript code into the one that the browser understands.

**20) What is Transpiler ?**

Transpiler is a tool that is used to convert source code into another source code that is of the same level. The latest standard version that JavaScript follows is ES2020 which is not fully supported by all browsers hence we make use of a tool such as ‘babel’ so that we can convert it into the code that today’s browser understands.

**21) How will you update props in React?**

No React props are only read-only.

**22) Can browsers understand JSX code? Explain.**

No. A transpiler is needed to convert JSX to JavaScript. Web browsers are built to understand and read original JavaScript or JS objects. Since JSX is not a regular JS object, web browsers are unable to read it. For web browsers to read JSX, it must be made compatible by first converting it into regular JS objects. This is achieved by using a transcompiler, such as Babel. While this may seem like a simple ReactJS interview question, knowing answers to such questions can be critical.

**23) Why fragments are better than container divs?**

Below are the list of reasons to prefer fragments over container DOM elements,

1. 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.
2. 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.
3. The DOM Inspector is less cluttered.

### 24) What is reconciliation?

Reconciliation is the process through which React updates the Browser DOM and makes React work faster. React use a diffing algorithm so that component updates are predictable and faster. React would first calculate the difference between the real DOM and the copy of DOM (Virtual DOM) when there's an update of components. React stores a copy of Browser DOM which is called Virtual DOM. When we make changes or add data, React creates a new Virtual DOM and compares it with the previous one. This comparison is done by Diffing Algorithm. Now React compares the Virtual DOM with Real DOM. It finds out the changed nodes and updates only the changed nodes in Real DOM leaving the rest nodes as it is. This process is called Reconciliation.

### 25) What is children prop?

Children is a prop 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.

### What are controlled components?

**Controlled Components**

A controlled component is an element whose value is controlled by the state of a React component. This means the state of the form element is kept in the React component, and changes to the element are handled by React through event handlers.

**Characteristics:**

* The value of the input is set by the component's state.
* Changes to the input value trigger an event handler that updates the state.
* The component's state is the single source of truth.

**Example:**

jsx

Copy code

import React, { useState } from 'react';

function ControlledComponent() {

const [inputValue, setInputValue] = useState('');

const handleChange = (event) => {

setInputValue(event.target.value);

};

const handleSubmit = (event) => {

event.preventDefault();

alert(`Submitted value: ${inputValue}`);

};

return (

<form onSubmit={handleSubmit}>

<input

type="text"

value={inputValue}

onChange={handleChange}

/>

<button type="submit">Submit</button>

</form>

);

}

export default ControlledComponent;

**Key Points:**

* The input value is controlled by the state variable inputValue.
* handleChange updates the state whenever the input value changes.
* The form submission reads the value directly from the state.

### What are uncontrolled components?

**Uncontrolled Components**

An uncontrolled component is an element that manages its own state internally. This means the form data is handled by the DOM itself rather than by React state.

**Characteristics:**

* The value of the input is not directly controlled by the React state.
* You use refs to access the input's value from the DOM when needed.
* Useful for simple forms or when integrating with non-React code.

**Example:**

jsx

Copy code

import React, { useRef } from 'react';

function UncontrolledComponent() {

const inputRef = useRef(null);

const handleSubmit = (event) => {

event.preventDefault();

alert(`Submitted value: ${inputRef.current.value}`);

};

return (

<form onSubmit={handleSubmit}>

<input

type="text"

ref={inputRef}

/>

<button type="submit">Submit</button>

</form>

);

}

export default UncontrolledComponent;

**Key Points:**

* The input value is managed by the DOM.
* inputRef is used to get the current value of the input when the form is submitted.
* The form submission reads the value directly from the DOM.

**When to Use Each Approach**

* **Controlled Components:**
  + When you need to immediately validate or format input.
  + When the form state needs to be synchronized with other UI elements or logic.
  + When you need fine-grained control over form behavior, such as disabling the submit button until all fields are valid.
* **Uncontrolled Components:**
  + When you need simple form behavior without much interaction.
  + When integrating with libraries or non-React code that expects DOM elements to manage their own state.
  + When you need better performance for a large number of form elements, as uncontrolled components can reduce the number of re-renders.

### Skipping Elements During Destructuring

If you want to skip certain elements while destructuring an array, you can use empty commas:

javascript

Copy code

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

// Skip the first two elements

const [ , , thirdElement] = array;

console.log(thirdElement); // Output: 3

### Combining Skipping and Rest Parameters

You can combine both skipping elements and using the rest syntax in array destructuring:

javascript

Copy code

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

// Skip the first two elements and capture the rest

const [ , , ...restOfArray] = array;

console.log(restOfArray); // Output: [3, 4, 5]

1. **How can we have good optimized react app ?**

### Memoization

Memoization in React is a technique used to optimize the performance of functional components by caching the results of expensive computations or function calls. It's particularly useful when dealing with computationally intensive or frequently called functions with the same input values, as it helps avoid redundant calculations and improves the overall efficiency of the application.

In React, there are three techniques for memoization: React.memo(), useMemo(), and useCallback(). Let's delve into the details for each:

The useMemo() hook optimizes performance by memoizing the result of a function call or an expensive computation. It caches the result and recalculates it only when the input values change. Below is an example on how to use the useMemo hook in functional component:

The useCallback() hook in React is used to memoize a function instead of memoizing the function result. It is particularly useful when passing events as props to child components to prevent unnecessary re-renders.

useCallback() memoizes the function, ensuring it remains the same across re-renders as long as the dependencies haven't changed.

### Throttling and Debouncing Events ?

### Debounce is when user click stops clicking for some time of gap of give certain amount of gap.

### Example: gun when user clicks reapetedly it willnot shoot until user give certain miliseconds of gap.

### Throttling:

### No mater how many times a user click under certain miliseconds it will shoots.

Throttling in React is a technique used to limit the number of times a function or an event handler is invoked. It ensures that the function is called at a specified interval, preventing it from being executed too frequently.

Throttling allows you to control the rate at which the function is called by setting up a minimum time interval between each function invocation. If the function is called multiple times within that interval, only the first invocation is executed, and subsequent invocations are ignored until the interval elapses

Now, let's illustrate throttling with a code example. First, without throttling:

// Without throttling, this function will be called every time the event is triggered

function handleResize() {

console.log('Window resized');

}

window.addEventListener('resize', handleResize);

With throttling, we can limit how often the handleResize function is called:

// Throttling function

function throttle(func, delay) {

let lastCall = 0;

return function(...args) {

const now = new Date().getTime();

if (now - lastCall < delay) {

return;

}

lastCall = now;

func(...args);

};

}

// Throttled event handler

const throttledHandleResize = throttle(handleResize, 200);

window.addEventListener('resize', throttledHandleResize)

In this example, the throttle function wraps handleResize and ensures it's not called more often than every 200 milliseconds. If the resize event fires more frequently than that, the handleResize function will only be executed once every 200 milliseconds, reducing the potential for performance issues caused by rapid, repeated function calls.

Debouncing, on the other hand, is also used to limit the number of times a function or an event handler is invoked. It ensures that the function is called only after a certain period of inactivity. Debouncing allows you to postpone the function call until the user has finished typing or a specific time has elapsed since the last event.

For example, imagine you have a search input field and want to trigger a search API request only when the user has finished typing for a certain duration, like 300ms.

With debouncing, the search function will only be invoked after the user stops typing for 300ms. If the user continues typing within that interval, the function call will be delayed until the pause occurs. Without debouncing, the function will be called for every keystroke, potentially leading to excessive function calls and unnecessary computation. let's demonstrate with a code example:

**Real Life Use Case :**

If the phone rings after 10 seconds and you receive it:

* **Debouncing**: You would ensure that you respond to the phone call only after it stops ringing for a specified period, like 10 seconds without additional rings.
* **Throttling**: You would limit how often you respond to incoming calls, perhaps by setting a minimum time interval between responses, regardless of how frequently the phone rings.

1. **Why fragments are better than container divs?**

Below are the list of reasons to prefer fragments over container DOM elements,

1. 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.
2. 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.
3. The DOM Inspector is less cluttered.
4. **Can we pass data from child to parent components ?**

Yes, it is possible to pass data from a child component to a parent component in React. This is typically done using callback functions defined in the parent component, which are then passed down to the child component as props. When the child component needs to send data to the parent, it calls this callback function with the data as an argument.

Here is an example demonstrating this:

**Example**

**Parent Component (Parent.js)**:

import React, { useState } from "react";

import Child from "./Child";

function Parent() {

// State in the parent component

const [parentData, setParentData] = useState("");

// Callback function to update state

const handleDataFromChild = (data) => {

setParentData(data);

};

return (

<div>

<h1>Parent Component</h1>

<p>Data from child: {parentData}</p>

{/\* Pass the callback function as a prop to the child component \*/}

<Child onDataFromChild={handleDataFromChild} />

</div>

);

}

export default Parent;

**Child Component (Child.js)**:

import React, { useState } from "react";

function Child({ onDataFromChild }) {

// Local state in the child component

const [childData, setChildData] = useState("");

// Handle input change

const handleChange = (e) => {

setChildData(e.target.value);

};

// Handle button click

const handleClick = () => {

// Call the parent callback function with the data

onDataFromChild(childData);

};

return (

<div>

<h2>Child Component</h2>

<input type="text" value={childData} onChange={handleChange} />

<button onClick={handleClick}>Send Data to Parent</button>

</div>

);

}

export default Child;

1. **Dom and Virtual dom ?**

The Actual DOM is the real, live structure of a webpage that the browser reads and displays. It’s how the browser understands and manipulates the content and structure of a webpage.

 **How it works**:

* When you open a webpage, the browser creates the Actual DOM from the HTML.
* If you change the HTML or use JavaScript to manipulate the page (like changing the text in a <div>), you're directly manipulating the Actual DOM.

 **Performance**:

* Direct changes to the Actual DOM can be slow because every change can cause the browser to re-draw the page (or parts of it), which is computationally expensive.

 The Virtual DOM is a lightweight, in-memory representation of the Actual DOM used by libraries like React. It’s a virtual copy of the Actual DOM that React keeps in memory.

 **How it works**:

* When a React component is rendered, React creates a Virtual DOM representation of the component.
* When the state or props of a component change, React updates the Virtual DOM first.
* React then compares the updated Virtual DOM with the previous version (this process is called “diffing”).
* After finding the differences, React updates only the parts of the Actual DOM that changed.

Performance:

* + Updating the Virtual DOM is very fast because it’s just JavaScript objects in memory.
  + After finding the differences, React updates the Actual DOM in the most efficient way, minimizing the number of changes and improving performance.

When using the Virtual DOM in frameworks like React, updates are optimized to only modify the specific parts of the DOM that have changed. In your example, if you update only item 2 in a list using React and the Virtual DOM, React will efficiently update only that specific item in the Actual DOM without re-rendering the entire list.

On the other hand, without using the Virtual DOM, if you directly manipulate the DOM to update item 2 in the list, the browser will typically re-render the entire list. This is because direct DOM manipulation doesn't have the intelligence to identify and update only the specific part of the DOM that has changed.

So, using the Virtual DOM ensures that updates are handled efficiently and only affect the necessary parts of the DOM, leading to better performance and a smoother user experience compared to manual DOM manipulation without the Virtual DOM abstraction.

1. **Why while Editing and updating we need the Immutibility?**

React components rely on detecting changes to determine when to re-render. With immutable data, React can easily detect changes through reference checks, making updates more efficient.

Immutable data ensures that other parts of the code or components do not unintentionally modify the data. This reduces the risk of unintended side effects and ensures that the data remains consistent throughout the application.

Immutable data structures help ensure that state changes are predictable. When data is immutable, it’s easier to trace how and when state changes occur, which simplifies debugging and tracking down issues.

1. **What is Fetch method and what it returns ?**

The fetch method is a modern JavaScript API used to make network requests. It's part of the Fetch API, which provides a more flexible and powerful way to handle HTTP requests and responses compared to older methods like XMLHttpRequest.

 **Syntax**: fetch(url, options)

* url: The URL to which the request is sent.
* options: An optional object to configure the request (method, headers, body, etc.).

 **Promises**: fetch returns a Promise that resolves to the Response object representing the response to the request.

 **Response Handling**: You need to handle the response using methods like .json(), .text(), or .blob() to extract the data.

 **Error Handling**: fetch won't reject the Promise on HTTP errors like 404 or 500; you need to manually check response.ok to handle HTTP errors.

1. **What is Sythetic Events?**

It provides cross browser wrapper around native browser’s event system which ensure that your event handling code works consistently across all supported browsers by normalizing the event properties and methods.

It provides a consistent, normalized interface for handling events across different browsers.

For Example:

The e.preventDefault is used to remove the default behaviour of form submit it show question mark on url.

      import React from 'react';

      function App() {

        const handleSubmit = (event) => {

          // Prevents the form from submitting and causing a page reload

          event.preventDefault();

          // Custom logic for handling form data

          console.log('Form submission prevented. Handle form data here.');

        };

        return (

          <form onSubmit={handleSubmit}>

            <input type="text" name="username" placeholder="Enter your username" required />

            <button type="submit">Submit</button>

          </form>

        );

      }

      export default App;

In the handleSubmit function, the event parameter is a synthetic event created by React. This synthetic event object wraps the native DOM event, providing a consistent interface.

**Cross-Browser Consistency:** Synthetic events ensure that preventDefault() and other event methods work uniformly across different browsers.

**Why Synthetic Events are Used**

1. **Cross-Browser Compatibility**:
   * Different browsers can have slight variations in how they handle events. Synthetic events ensure that your event handling code works consistently across all supported browsers by normalizing the event properties and methods. This abstraction eliminates the need to handle browser-specific quirks and inconsistencies.

**What is Event Pooling?**

* **Definition**: Event pooling refers to React's practice of reusing synthetic event objects rather than creating new ones for each event. This is achieved by pooling event objects in memory and reusing them to reduce the overhead of object creation and garbage collection.

**How It Works**

1. **Synthetic Events**: When an event occurs, React creates a synthetic event object that wraps the native DOM event. This object contains normalized properties for consistent behavior across different browsers.
2. **Event Pooling**: After the event handler finishes executing, React does not immediately destroy the event object. Instead, it returns the event object to a pool for reuse in future events.
3. **Event Persistency**: If you need to access the event object asynchronously (e.g., in a setTimeout or Promise), you must call event.persist() to prevent React from reusing or resetting the event object.

**Benefits**

* **Performance**: Reduces the overhead associated with creating and disposing of event objects, leading to better performance and lower memory usage.
* **Efficiency**: Helps manage large numbers of events more efficiently, especially in applications with many interactive components.

import React from 'react';

function App() {

const handleClick = (event) => {

event.persist(); // Prevents event pooling for this event object

setTimeout(() => {

console.log(event.type); // Accesses the event object asynchronously

}, 1000);

};

return <button onClick={handleClick}>Click me!</button>;

}

export default App;

**Who has more priority Parent prop or default props?**

When passing both **props** from a parent component and **default props** in a child component, the **props from the parent** have higher priority. This is because if a prop is passed from the parent, it will override the default prop value set in the child component.

**Default Props vs Undefined Props**

If a prop is **explicitly passed as undefined**, React will treat it differently than if it is **not passed at all**. Default props will only apply if the prop is undefined **because it wasn't provided**.

* **Trick:** Passing undefined explicitly overrides the default prop.

**Event Handlers vs Synthetic Events: Which event takes priority?**

React uses **synthetic events** to normalize event handling across different browsers. If you mix native DOM events and React’s synthetic events, they may not interact as expected.

* **Trick:** **Synthetic events** are handled first, and native DOM events follow. Be cautious when using both in the same component.

const Button = () => {

const handleClick = (e) => {

e.preventDefault();

console.log("React Synthetic Event");

};

useEffect(() => {

document.getElementById('button').addEventListener('click', () => {

console.log("Native DOM Event");

});

}, []);

return <button id="button" onClick={handleClick}>Click Me</button>;

};

**Priority Insight:** In this example, **"React Synthetic Event"** logs before **"Native DOM Event"** because React handles synthetic events first.

**Component Re-render vs Memoization: Which takes priority?**

React re-renders a component when its **state or props** change. However, using React.memo or useMemo can prevent unnecessary re-renders.

* **Trick:** React.memo only prevents re-renders due to **prop** changes. It doesn't prevent re-renders due to **state** changes in the same component.

**When the UseEffect runs twice?**

In strinct mode

**What is Synthetic Events ?**

React uses **Synthetic Events**, which are cross-browser wrappers around the native events. Synthetic events standardize event behavior across different browsers, ensuring compatibility. When you call e.preventDefault() in React, you're actually interacting with this synthetic event.