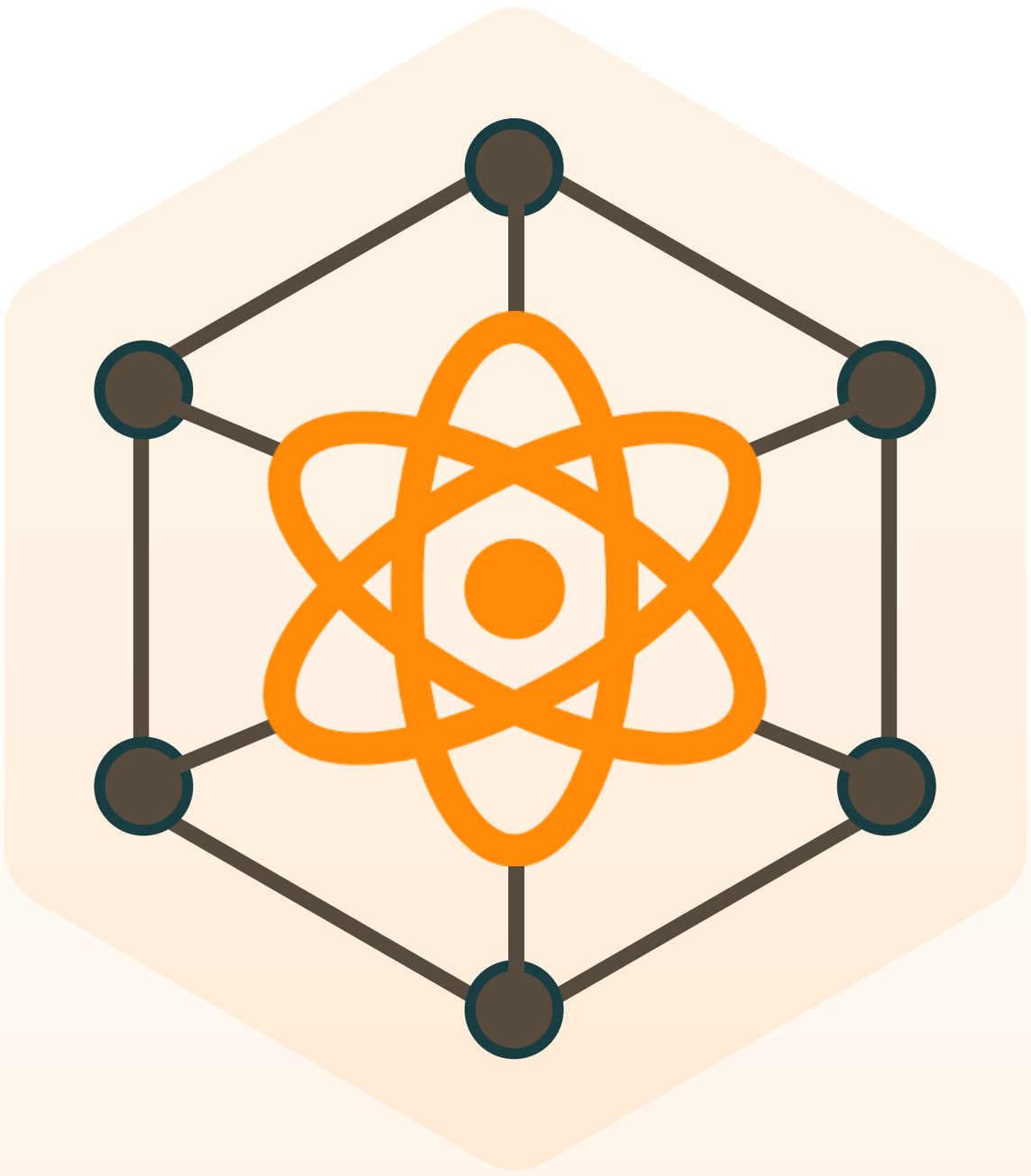
BOSSCODER ACADEMY

REACT.JS QUESTIONS

with Answers to Crack the

TECHNICAL INTERVIEW



Disclaimer

Everyone learns uniquely.

What matters is preparing for the technical interviews thoroughly.

This doc will help you with the same. Prepare React better with these Interview Questions.

EASY

Q.1

What are the features of React?

Ans →

The features of React are as follows:

1. JSX

JSX serves as a syntax extension to JavaScript, facilitating the combination of HTML structures with JavaScript code within React files.

2. Components

JSX serves as a syntax extension to JavaScript, facilitating the combination of HTML structures with JavaScript code within React files.

3. Virtual DOM

React employs a Virtual DOM, which is a lightweight representation of the actual DOM stored in memory. This approach allows React to selectively update only the relevant parts of the real DOM when the state of an object changes.

EASY

4. Data Binding

React adopts a one-way data-binding approach, ensuring a modular and efficient structure. Unidirectional data flow signifies that in a React app, child components are often nested within parent components.

5. High Performance

React's high performance is driven by its ability to update only the components that undergo changes, rather than refreshing the entire set. This results in significantly faster web applications.

EASY

Q.2

What is difference between element and component in React?

Ans →

The features of React are as follows:

Element

An Element is a simple object that describes what you want to show on the screen. It defines the structure of DOM nodes or other components. Elements can include other Elements in their properties. Once created, Elements cannot be changed. Creating a React Element is a straightforward and inexpensive operation.

Example of creating an Element using JSX:

JSX serves as a syntax extension to JavaScript, facilitating the combination of HTML structures with JavaScript code within React files.

JSX

```
const element = <div id="login-btn">Login</div>;
```

EASY

When expressed without JSX, it would look like:

JSX

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

This element is essentially an object:

JSX

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

This object is then rendered to the DOM using `ReactDOM.render()`.

JSX

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

EASY

Component

A Component can be declared in various ways. It can be a class with a render() method, or it can be defined as a function. Components take props as input and return a JSX tree as output. Components are more powerful and flexible compared to Elements.

Example of creating a functional Component using JSX

JSX

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

When transpiled, it becomes:

JSX

```
const Button = ({ handleLogin }) =>
  React.createElement(
    "div",
```

```
    { id: "login-btn", onClick: handleLogin },
       "Login"
     );
```

In this example, Button is a functional component that takes handleLogin as a prop and returns a JSX tree. Components are a more dynamic way to create reusable pieces of UI.

EASY

Q.3

How to create components in React?

Ans →

1. Function Components

Function components are the simplest way to create a component in React. They are pure JavaScript functions that take a props object as the first parameter and return React elements to display the output.

Example of a function component

JSX

```
function Welcome({ personName }) {
  return <h1>{`Welcome, ${personName}!`}</h1>;
}
```

In this function component, `Welcome` takes a `personName` prop and displays a personalized welcome message.

2. Class Components

Alternatively, you can use ES6 classes to define a component. The equivalent class component for the above function component would look like this:

EASY

JSX

```
class Greeting extends React.Component {
  render() {
    return <h1>{`Greetings, ${this.props.userName}!`}
</h1>;
  }
}
```

In this class component, `Greeting` extends `React.Component` and uses a `render` method to display a greeting. The prop is accessed using `this.props.userName`.

Both function and class components accomplish the same goal of creating reusable and modular pieces of UI. The choice between them depends on the complexity of the component and whether state or lifecycle methods are needed.

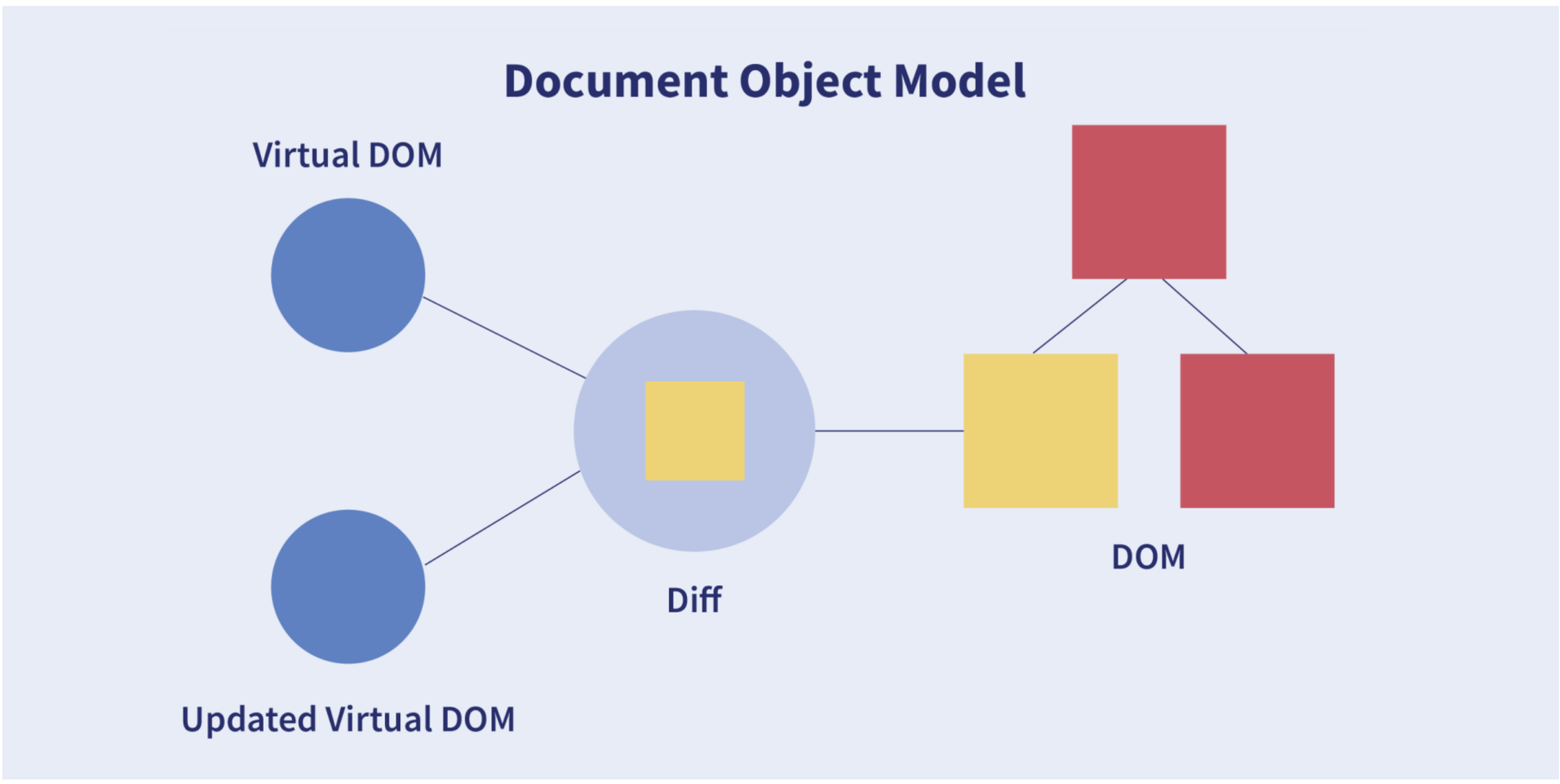
EASY

Q.4

What is the Virtual DOM?

Ans →

The Virtual DOM is like a blueprint or a copy of the real DOM that is stored in the computer's memory. It's a concept used by React to make updating and changing things on a webpage more efficient.

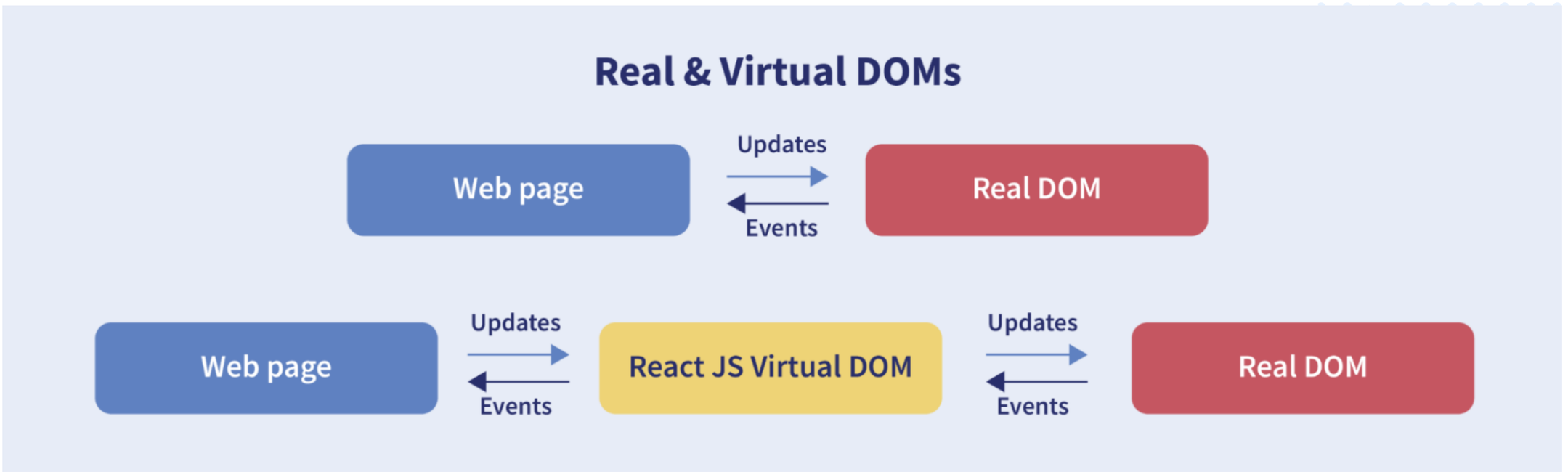


Why is it Needed?

When we make changes to a webpage, like updating a list, traditional methods often involve updating the entire webpage, even if only a small part has changed. This can be slow and inefficient.

EASY

How Does it Work?



1. Virtual DOM Objects

For every object on the webpage, there is a corresponding virtual object in the memory. These virtual objects have the same properties as the real objects.

2. Blueprint of the DOM

Think of the virtual DOM as a blueprint of the real DOM. Changes made to the virtual DOM don't immediately show up on the screen; they are like plans for what should change.

3. Faster Updates

Updating the virtual DOM is much faster than updating the real DOM. It's like working on a draft before finalising a document.

EASY

4. Two Virtual DOMs

React uses two sets of virtual DOMs – one to store the current state and another to store the previous state of objects.

5. Efficient Updating

When something changes, React compares the two virtual DOMs to see what's different. It then updates only the parts that have changed in the real DOM, rather than updating the entire webpage.

In simpler terms, the Virtual DOM is like a behind-the-scenes helper that makes updating web pages faster and more efficient by smartly figuring out what needs to change and updating only those parts.

EASY

Q.5

What are keys in React and why do we need them?

Ans →

The "key" is a special attribute used when working with arrays of elements in React. It helps React keep track of changes, additions, and removals in the array.

When you're rendering a list of items, React needs a way to identify each item uniquely. The "key" prop serves this purpose, allowing React to efficiently update the user interface.

Example

Suppose you have a list of books:

JSX

```
const books = [
  { id: 1, title: "React Magic" },
  { id: 2, title: "JavaScript Wonders" },
  { id: 3, title: "Web Development Odyssey" },
];
```

EASY

You can use the "key" prop when mapping over this array to render each book:

JSX

```
const bookItems = books.map((book) => <li
key={book.id}>{book.title}</li>);
```

Here, we're using the unique id of each book as the key.

Note

- It's crucial to use unique keys among siblings to avoid issues.
- If your data doesn't have stable IDs, using the item index as a key is a last resort. However, this is not recommended if the order of items may change, as it can impact performance.
- If you extract list items into separate components, apply keys to the component instead of the li tag.
- The "key" attribute accepts either a string or a number, and it's converted internally to a string type.
- A warning message will appear in the console if the "key" prop is not present on list items.

EASY

Q.6

Explain the steps to create a react application and print hello world?

Ans →

Steps to Create a React Application

1. Install Node

Before installing React, ensure that Node is installed on your computer. You can download it from `Node.js`.

2. Create React App

Open the terminal and run the following command to create a new React application (replace `my-react-app` with your preferred application name):

JSX

```
npx create-react-app my-react-app
```

Navigate to the Application Folder

Move to the newly created application folder:

EASY

JSX

```
cd my-react-app
```

Print "Hello World!" Example

Now, open the `src/App.js` file and replace its content with the following:

JSX

```
import React from 'react';

function App() {
  return (
    <div>
      <h1>Hello World!</h1>
    </div>
  );
}

export default App;
```

Save the file.

EASY

Run the Application

In the terminal, run the following command to start the development server:

```jsx
npm start
```

This will open your new React application in a web browser, and you should see "Hello World!" displayed on the webpage.

In this example, the `App` component is a simple React function component that returns JSX to render the "Hello World!" message. The `npm start` command is used to run the application and launch a development server.

EASY

Q.7

How are comments written in React?

Ans →

Comments in React/JSX are similar to JavaScript multiline comments but are enclosed in curly braces.

Single-line comments

JSX

```
<div>
  {/* Single-line comments(In vanilla JavaScript, the
single-line comments are represented by double
slash(//)) */}
  {`Welcome, ${userName}! Let's dive into React`}
</div>
```

Multi-line comments

JSX

```
<div>
  {/*
    This is a multiline comment in React.
    It provides additional information about the
code.
```

```
    */}
    {`Welcome, ${userName}! Let's dive into React`}
  </div>
```

In these modified examples, the comments now convey a welcoming message to the user, demonstrating how comments can be used to explain and document code within the JSX structure.

EASY

Q.8

Explain how lists are created in React?

Ans →

Lists are essential for displaying dynamic content on a website. In React, you can create a list using the map method of an array. Here's an example:

JSX

```
import React from 'react';
import ReactDOM from 'react-dom';
 // Example list of items
const fruits = ['Apple', 'Banana', 'Orange', 'Grapes', 'Watermelon'];
 // Using map to create a list of JSX elements
const fruitList = fruits.map((fruit, index) => {
  return <li key={index}>{fruit}</li>;
});
 // Rendering the list inside an unordered list
ReactDOM.render(
  <ul>
    {fruitList}
```

EASY

In this modified example, we have a list of fruits, and the `map` method is used to create a list of JSX elements (`<li>` elements) dynamically. Each fruit is represented as a list item, and the resulting list is rendered inside an unordered list (`<ul>`) in the specified HTML element with the ID 'root'. The `key` attribute is added to each `<li>` element for better performance and React's internal tracking of list items.

EASY

Q.9

Explain the difference between functional components and class components.

Ans →

Functional Components

1. Definition

A functional component is a plain JavaScript pure function that accepts `props` as an argument.

2. Rendering

Does not use the `render` method. Instead, the component's return value represents the UI.

3. State

Cannot use state. It is also known as a stateless component.

4. Lifecycle Methods

Cannot use React lifecycle methods (e.g., `componentDidMount`).

EASY

5. Constructor

Does not use a constructor.

Class Components

1. Definition

A class component requires you to extend from `React.Component` and create a `render` function.

2. Rendering

Must have the `render()` method, which returns JSX representing the UI.

3. State

Can use state. It is also known as a stateful component.

4. Lifecycle Methods

Can use React lifecycle methods (e.g., `componentDidMount`).

5. Constructor

Uses a constructor, especially when state needs to be stored.

EASY

Summary

- Functional components are simple functions that accept props and return JSX. They are stateless and don't use a constructor or React lifecycle methods.
- Class components are ES6 classes that extend React.Component. They have a render method, can use state, a constructor, and React lifecycle methods. They are suitable for managing state and implementing more complex logic.

Choosing between them depends on the specific requirements of the component. Functional components are preferred for simpler scenarios, while class components offer more features for complex state management and lifecycle methods.

EASY

Q.10

What are React Hooks?

Ans →

React Hooks are built-in functions introduced in React version 16.8 that allow developers to utilize state and lifecycle methods within functional components. They enhance code reusability and provide flexibility in navigating the component tree.

Before Hooks, class components were primarily used for managing state and lifecycle methods. With Hooks, developers can now access these features directly in functional components, eliminating the need for class components.

Example of a Hook: useState

JSX

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


function Counter() {
  // Declare a state variable called 'count'
  // 'setCount' is a function to update/change the
value of 'count'
```

EASY

```
let [count, setCount] = useState(0);


  // 'count' can be directly used inside the
component's JSX
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}
>Increment</button>
    </div>
  );
}
```

In this updated example, the `useState` hook is used to manage the state of a counter. The state variable 'count' is initially set to 0, and the updater function 'setCount' is used to increment the count when a button is clicked. The 'count' value is then directly used within the JSX to display the current count.

EASY

Q.11

What is useState() in React?

Ans →

The useState() is a fundamental React Hook used to introduce state variables into functional components, especially when dynamic control over elements in the DOM is required.

Consider this alternative example where `useState("Hello")` is employed:

JSX

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

 function Greeting() {

  // Declare a state variable 'message' initialised with "Hello"

  // 'setMessage' is the function to update/change the value of 'message'

  const [message, setMessage] = useState("Hello");

   // 'message' can be directly used inside the component's JSX

  return (

    <div>

      <p>{message}, React!</p>
```

```
        <button onClick={() => setMessage("Hola")}
>Change Greeting</button>
      </div>
    );
}
```

In this example, `useState("Hello")` initialises the state variable `message` with the value "Hello," and the updater function `setMessage` allows changing the greeting dynamically. Clicking the button triggers the update, demonstrating the versatility of `useState()` for managing various state scenarios in React functional components.

MEDIUM

Q.12

What are the different types of Hooks in React?

Ans →

Basic Hooks

useState()

Used to manage and retrieve state in functional components.

useEffect()

Enables performing side effects in functional components, like data fetching or DOM manipulation.

useContext()

Creates shared data accessible by components in a hierarchy without passing props through each level.

Additional Hooks

useReducer()

Helpful for complex state logic or when the next state depends on the previous state, optimizing performance by passing dispatch down.

MEDIUM

useCallback()

Useful when passing callbacks to optimized child components to prevent unnecessary renders by checking reference equality.

useImperativeHandle()

Allows modifying the instance passed with a ref object.

useDebugValue()

Displays a label for custom hooks in React DevTools.

useRef()

Creates a reference to a DOM element directly within a functional component.

useLayoutEffect()

Reads layout from the DOM and triggers synchronous re-rendering.

Custom Hooks

These are functions in JavaScript that follow React's Hook rules and begin with "use."

They help extract component logic into reusable functions, making your code more modular and easier to understand.

MEDIUM

Q.13

What is Strict Mode in React?

Ans →

`React.StrictMode` is a component designed to highlight potential issues and enforce best practices in a React application. It does not introduce additional DOM elements and operates exclusively in development mode.

Usage

Wrap parts of the application in `<React.StrictMode>` to activate additional checks and warnings.

Development Mode Only

Strict mode checks apply exclusively in development mode, helping developers catch potential problems early.

Example

In the example below, strict mode checks apply to `<ComponentOne>` and `<ComponentTwo>`.

MEDIUM

JSX

```
import React from "react";

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

Strict Mode Checks

- Identifies components with unsafe lifecycle methods.
- Warns about the usage of legacy string refs.
- Warns against using findDOMNode method.
- Highlights potential issues with legacy context API usage.

MEDIUM

Q.14

How is React different from Angular?

Ans →

	Angular	React
Author	Google	Facebook Community
Developer	Misko Hevery	Jordan Walke
Initial Release	October 2010	March 2013
Language	JavaScript, HTML	JSX
Type	Open Source MVC Framework	Open Source JS Framework
Rendering	Client-Side	Server-Side
Data-Binding	Bi-directional	Uni-directional
DOM	Regular DOM	Virtual DOM
Testing	Unit and Integration Testing	Unit Testing
App Architecture	MVC	Flux
Performance	Slow	Fast, due to virtual DOM.

MEDIUM

Q.15

What are the different phases of the component lifecycle?

Ans →

Phases of the Component Lifecycle in React

The lifecycle of a React component is divided into four phases:

1. Initialization

In this phase, the React component gets ready by setting up default props and initializing the state.

2. Mounting

Mounting involves putting the elements into the browser DOM. React utilizes VirtualDOM, and during mounting, only the changed elements are updated in the browser DOM. This phase includes the following lifecycle methods:

- `componentWillMount`
- `componentDidMount`

3. Updating

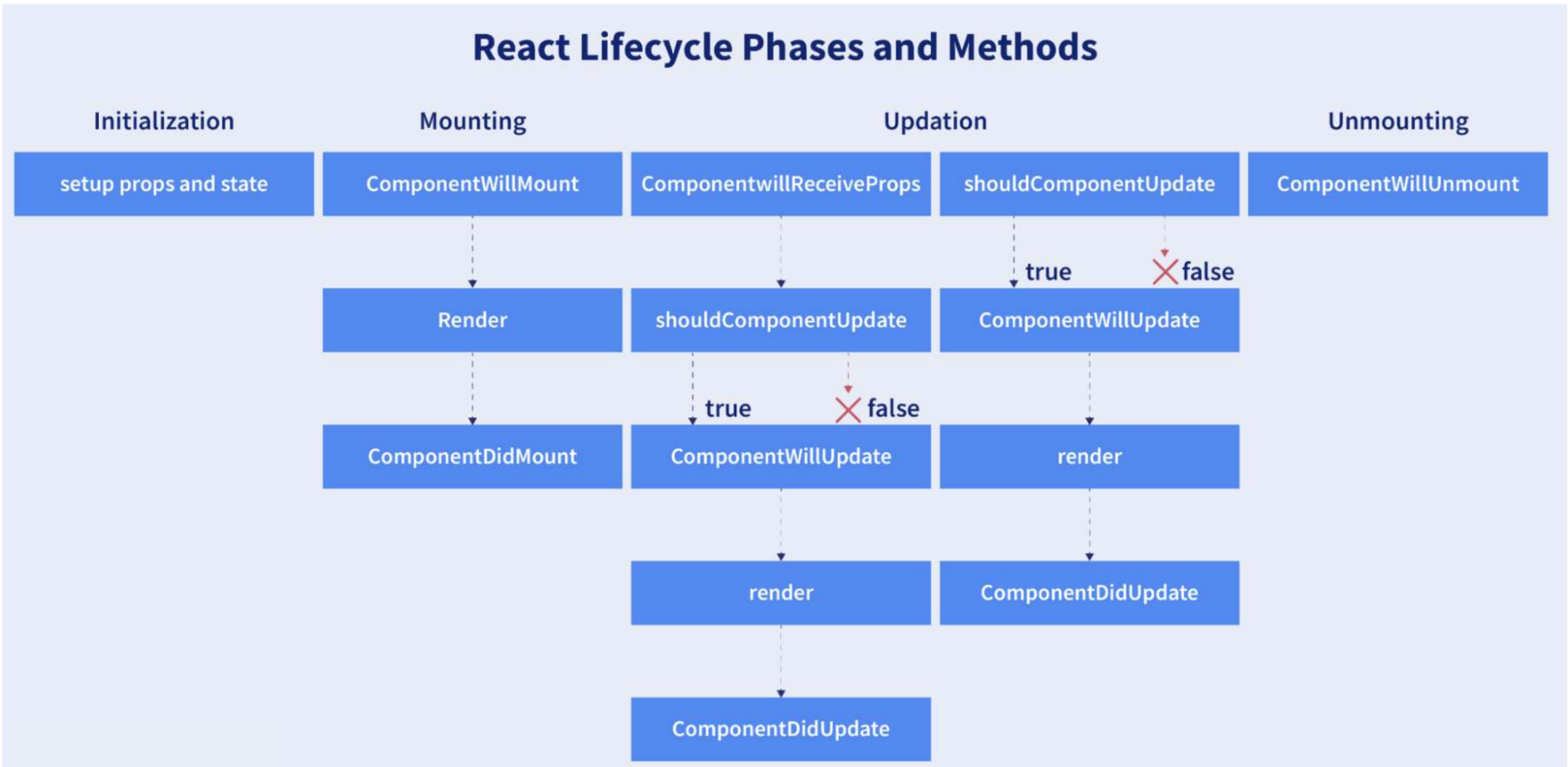
When there is a change in the state or props of a component, the updating phase is triggered. This phase includes the following lifecycle methods:

- `componentWillUpdate`
- `shouldComponentUpdate`
- `render`
- `componentDidUpdate`

4. Unmounting

In the unmounting phase, the component is removed from the DOM or unmounted. This phase includes the following lifecycle method:

- `componentWillUnmount`



MEDIUM

Q.16

What are the lifecycle methods of React?

Ans →

React lifecycle methods are functions automatically called at different phases in a component's lifecycle, offering control over its behavior. Understanding and utilizing these methods empower developers to efficiently manage various aspects throughout the component's existence.

Example Scenario

Consider a YouTube application. When a user switches to another app after playing a video, efficient resource management is crucial. Lifecycle methods help developers ensure optimal utilization of resources like network and battery.

Key Lifecycle Methods

1. **constructor()**
   - Called during component initiation.
   - Sets up initial state and values.

2. getDerivedStateFromProps()

- Called just before rendering elements in the DOM.
- Sets up the state based on initial props.
- First method called on component update.

3. render()

- Outputs or re-renders HTML to the DOM with new changes.
- Essential method called on every render.

4. componentDidMount()

- Called after component rendering.
- Executes statements requiring the component to be in the DOM.

5. shouldComponentUpdate()

- Returns a Boolean specifying whether React should proceed with rendering.
- Default value is `true`.

6. getSnapshotBeforeUpdate()

- Provides access to props and state before the update.
- Allows checking previous values after the update.

7. componentDidUpdate()

- Called after the component is updated in the DOM.

8. componentWillUnmount()

- Called when the component is about to be removed from the DOM.

MEDIUM

Q.17

What is prop drilling?

Ans →

The lifecycle of a React component is divided into four phases:

Example Scenario

- Consider a scenario where `<EditUsersPage />` maintains `selectedUserAddress` in its state.
- `<EditUsersPage />` renders `<User />`, which, in turn, renders `<UserDetails />`.
- `<UserDetails />` contains a `<UserAddress />` component that requires access to `selectedUserAddress`.

Approach

The straightforward solution is to pass `selectedUserAddress` as a prop from `<EditUsersPage />` to `<User />`, then to `<UserDetails />`, and finally to `<UserAddress />`.

MEDIUM

Avoiding Prop Drilling

1. Alternative Approach

- Utilize React context to sidestep prop drilling.

2. React Context

- Define a Provider component to supply data.
- Nested components can then consume this context data through a Consumer component or the useContext hook.

3. Benefits

- Utilize React context to sidestep prop drilling.

4. Global State Sharing

- Utilize React context to sidestep prop drilling.

5. State Management Module

- Alternatively, state management modules like Redux can be employed to handle data indirectly through context.

MEDIUM

Q.18

What is React Router?

Ans →

React Router is like a navigation manager for React applications. It helps build single-page web apps where you can navigate to different sections without refreshing the entire page. This keeps the user experience smooth and also updates the browser URL as you move around.

In React, components are a big deal, and React Router uses this concept. You don't have to use React Router, but it's a popular choice for managing navigation.

Key components of React Router

1. **BrowserRouter**
   - This is like the boss. It uses the HTML5 history API to keep your app in sync with the URL. It's like the container that holds all the other components.
2. **Routes**
   - This is a newer addition to React (as of version 6). Think of it as an upgraded version of the component that helps with routing.

3. Route

- This is where the action happens. Whenever the URL matches the path you set, this component decides what UI to show. It's like a conditionally displayed part of your app.

4. Link

- Similar to an anchor tag in HTML, this helps create links to different routes, making navigation smooth across your application.

In simpler terms, React Router is like a guide for your React app, helping you move between different pages or sections without reloading the entire page. It's a way to organize and manage how your app responds to different URLs.

HARD

Q.19

What are Custom Hooks in React?

Ans →

Custom Hooks in React

Custom Hooks in React are reusable functions that encapsulate logic and stateful behavior, allowing you to share that logic across different components. They follow a naming convention starting with "use" (e.g., useCustomHook).

Purpose

Custom Hooks provide a way to extract and manage complex logic outside of components, promoting code reuse and maintaining a clean and modular codebase.

Example

- Consider a custom hook for handling form input:

HARD

JSX

```
// useInput.js
import { useState } from 'react';

const useInput = (initialValue) => {
  const [value, setValue] = useState(initialValue);

  const handleChange = (e) => {
    setValue(e.target.value);
  };

  return {
    value,
    onChange: handleChange,
  };
};

export default useInput;
```

Usage in a Component

- Now, you can use the useInput custom hook in any component to manage input state:

HARD

JSX

```jsx
import React from 'react';
import useInput from './useInput';


const MyComponent = () => {
  const usernameInput = useInput('');
  const passwordInput = useInput('');


  return (
    <form>
      <label>Username:
        <input type="text" {...usernameInput} />
      </label>

      <label>Password:
        <input type="password" {...passwordInput} />
      </label>
    </form>
  );
};
```

HARD

Explanation

- The useInput hook abstracts away the state management and event handling for input fields.
- The component using this custom hook can easily manage multiple input fields without duplicating similar logic.

Benefits of Custom Hooks

- **Reusability:** Logic can be reused across different components, promoting a DRY (Don't Repeat Yourself) codebase.
- **Readability:** Components become more concise and focused on rendering, with logic abstracted into custom hooks.
- **Maintainability:** Changes to shared logic can be made in one place, affecting all components using the custom hook.

HARD

Q.20

What are higher order components in React?

Ans →

Definition

- HOCs in React are functions that take a component and return an enhanced version, leveraging React's compositional nature.

Purity of HOCs

Often termed "pure components," HOCs accept any child component without altering its behavior.

Usage Pattern

Create an enhanced component using a higher-order function:

```jsx
const EnhancedComponent =
higherOrderComponent(WrappedComponent);
```

HARD

Use Cases

- **Code Reuse and Logic Abstraction**
  → Encapsulate and reuse code, abstracting logic for enhanced components.
- **Render Hijacking**
  → Customize component rendering by intercepting and modifying the process.
- **State and Props Manipulation**
  → Manage state within HOCs, manipulate or enhance props before passing them down.

Advantages

- **Modularity and Separation of Concerns**
  → Enhances code organization by separating concerns like state, logic, and rendering.
- **Composability**
  → Compose multiple HOCs for granular and reusable component composition.
- **Encapsulation**
  → Encapsulates specific functionalities, improving code clarity and testability.

BOSSCODER ACADEMY

WHY BOSSCODER?

1000+ Alumni placed at Top Product-based companies.

More than **136% hike** for every **2 out of 3** working professional.

Average package of **24LPA.**

The syllabus is most **up-to-date** and the list of problems provided covers **all important topics.**

Lavanya
Meta

Course is very well **structured** and **streamlined** to crack any **MAANG** company

Rahul .
Google

EXPLORE MORE