1Q) **How Java script engine connected/interacts to the browser so that the document runs in the browser.**The interaction between the **JavaScript engine** and the **browser** is a fundamental part of how web applications work. Here’s a simplified explanation of how they are connected and how JavaScript runs in the browser:

**1. The Browser's Architecture**

A browser consists of several components:

* **User Interface (UI)**: Displays the browser’s buttons, address bar, etc.
* **Rendering Engine**: Handles HTML and CSS to render web pages visually.
* **JavaScript Engine**: Executes JavaScript code.
* **Networking Layer**: Manages HTTP requests and responses.
* **Storage**: LocalStorage, SessionStorage, Cookies, etc., for persisting data.
* **Compositor**: Combines visual layers (CSS animations, canvas, etc.).

**2. The Role of the JavaScript Engine**

The **JavaScript engine** (e.g., V8 in Chrome, SpiderMonkey in Firefox, Chakra in Edge) is a key part of the browser, responsible for executing JavaScript code. Here's how it works:

* **Parsing and Compilation**:
  + When the browser encounters <script> tags in the HTML or external JavaScript files, it sends the code to the JavaScript engine.
  + The engine parses the code, converts it into **Abstract Syntax Trees (AST)**, and compiles it into bytecode or machine code for faster execution.
* **Execution**:
  + The engine executes the compiled code step-by-step.
  + JavaScript operates in a **single-threaded environment** but can handle asynchronous tasks using the **event loop**.

**3. Interaction Between the JavaScript Engine and the Browser**

The JavaScript engine doesn’t work in isolation; it interacts with the browser via several APIs:

* **DOM (Document Object Model)**:
  + The browser creates a DOM tree to represent the HTML document.
  + JavaScript manipulates this DOM (e.g., document.getElementById()).
* **BOM (Browser Object Model)**:
  + Provides interfaces for browser-specific features, like:
    - window: Represents the browser window.
    - navigator: Provides information about the browser.
    - screen: Provides information about the user’s screen.
* **Web APIs**:
  + The browser provides a set of APIs that JavaScript can call, including:
    - **AJAX/Fetch**: For making HTTP requests.
    - **Timers**: setTimeout() and setInterval().
    - **Geolocation**, **Notifications**, and more.
* **Event Loop**:
  + JavaScript uses an event-driven model.
  + When a user interacts with a page (e.g., clicks a button), the browser captures the event, and the JavaScript engine executes the appropriate callback functions.

**4. Steps in a Typical JavaScript Execution**

1. The browser loads the HTML and parses it into a DOM tree.
2. When it encounters a <script> tag, the JavaScript engine:
   * Fetches the script (if external).
   * Parses and compiles it.
   * Executes the code.
3. JavaScript code interacts with the DOM or uses Web APIs provided by the browser.
4. The browser updates the UI or performs actions as needed.

**5. Visualization of the Connection**

* **HTML Parsing** → DOM Tree
* **CSS Parsing** → CSSOM Tree
* **JavaScript Execution** (via Engine):
  + Reads/Modifies the DOM.
  + Calls Web APIs.
  + Sends tasks to the Event Loop.

**Example**

Consider this JavaScript code:

javascript

Copy code

document.getElementById('root').innerHTML = "Hello, Browser!";

* The browser parses the HTML and builds the DOM.
* The JavaScript engine:
  + Retrieves the element with id="root" from the DOM.
  + Modifies the innerHTML property to display text.
* The browser updates the UI to reflect the change.

**Conclusion**

The browser provides the **environment** (DOM, BOM, Web APIs, and Event Loop) for JavaScript to operate, while the JavaScript engine executes the code and interacts with these components. This seamless integration ensures dynamic and interactive web pages.  
  
2Q) what is CDN(Content Delivery Network)

**CDN** stands for **Content Delivery Network**. It is a geographically distributed network of servers that work together to deliver content (e.g., web pages, images, videos, scripts, stylesheets) to users more quickly and efficiently.

**How a CDN Works**

### When you access a website, instead of retrieving content directly from the website's origin server, a CDN delivers it from a server that is geographically closer to you. This reduces latency and ensures faster load times. Example of a CDN in Use

A website might include a link to a JavaScript library like this:

html

Copy code

<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>

* The cdnjs.cloudflare.com is a CDN.
* This link ensures the browser fetches jQuery from Cloudflare's distributed network of servers instead of the website's origin server.
* If the user has already loaded jQuery from the same CDN on another site, the file might be cached in the browser, further improving performance.

In summary, a CDN ensures efficient, fast, and reliable content delivery, making it an essential tool for modern web applications.

3Q) How CDN is related to react

A **CDN (Content Delivery Network)** is related to React in the following ways:

1. **Serving React Libraries**: React and ReactDOM can be loaded via CDNs (e.g., unpkg, jsDelivr) for quick access, especially in small projects or without a build system.
2. **Deploying React Apps**: React applications, once built into static files, can be hosted and served through CDNs like Netlify, Vercel, or AWS CloudFront for fast, global delivery.
3. **Using External Libraries**: CDNs provide a way to include dependencies (e.g., Bootstrap, Axios) in React projects without bundling them, reducing app size and leveraging browser caching.
4. **Improving Performance**: CDNs minimize latency by delivering files from servers closer to users, speeding up React app loading times and enhancing scalability.

In short, CDNs enhance React development, deployment, and performance by providing fast and efficient content delivery.

Q1) **Never define a component inside another component!**

Components can render other components, but you must never nest their definitions:

export default function Gallery() {

// 🔴 Never define a component inside another component!

function Profile() {

// ...

}

// ...

}

The snippet above is very slow and causes bugs. Instead, define every component at the top level:

export default function Gallery() {

// ...

}

// ✅ Declare components at the top level

function Profile() {

// ...

}

When a child component needs some data from a parent, pass it by props instead of nesting definitions.

Q2)

\*In a React app, every piece of UI is a component.

\*React components are regular JavaScript functions except:

i)Their names always begin with a capital letter.

ii)They return JSX markup.

Q3)what is mean by JSX markup

Q4)what to learn after React

In addition to learning ReactJS, there are several other technologies and concepts that are commonly used in conjunction with React that you should consider learning:

**JavaScript**: React is a JavaScript library, so a solid understanding of JavaScript is essential for working with React.

**JSX**: React uses JSX, which is a syntax extension for JavaScript. JSX allows you to write HTML-like elements in your JavaScript code. It takes some getting used to, but it's fundamental in React.

**Node.js and npm**: React applications are typically built using Node.js and npm, which are used to manage dependencies and run development tools.

**Webpack**: Webpack is a popular tool for building and bundling JavaScript applications. It is commonly used with React to create a development environment and bundle the application for production.

**Git and version control:** As you build React applications, it's important to use version control to track changes to your code. Git is the most popular version control system and is widely used in the React community.

**HTML and CSS**: React is focused on building the UI of an application and it's essential to have a good understanding of HTML and CSS.

**Redux**: Redux is a library that is often used with React to manage the state of an application. It provides a centralized store for application state and makes it easy to update and retrieve state throughout your application.

**React Router**: React Router is a library that is used to handle client-side routing in React applications. It allows you to define routes and navigate between different parts of your application.

**Firebase or any other backend service**: React is a front-end library, so you will need to learn how to connect to a back-end service to handle things like user authentication, data storage, and APIs. Firebase is a popular BaaS that allows you to easily set up a back-end for your React application.

Q5)The `style` prop expects a mapping from style properties to values, not a string. For example, style={{marginRight: spacing + 'em'}} when using JSX.

It means we need to mention all the styles in the Object

const person = {

name: 'Gregorio Y. Zara',

theme: {

backgroundColor: 'black',

color: 'pink'

},

titleStyle: {

fontFamily: 'Arial, sans-serif',

fontSize: '24px',

fontWeight: 'bold',

textDecoration: 'underline'

}

};

export default function TodoList() {

return (

<div style={person.theme}>

<h1 style={person.titleStyle}>{person.name}'s Todos</h1> {/\* Updated style for h1 \*/}

<img

className="avatar"

src="https://i.imgur.com/7vQD0fPs.jpg"

alt="Gregorio Y. Zara"

/>

<ul>

<li>Improve the videophone</li>

<li>Prepare aeronautics lectures</li>

<li>Work on the alcohol-fuelled engine</li>

</ul>

</div>

);

}

Q6)The event handling can be done in different ways. Let's break down how the onClick (or onclick)

It differs when there is a event handler in the script

HTML: Uses onclick (case-insensitive, often lowercase) directly in the HTML element

<button onclick="alert('Button clicked!')">Click Me</button>  
  
JavaScript: When directly manipulating the DOM, use element.onclick to set the event handler.

<button id="myButton">Click Me</button>

<script>

const button = document.getElementById('myButton');

button.onclick = function() {

alert('Button clicked!');

};

</script>  
  
React: Uses camelCase onClick as a prop in JSX. which is a syntactic sugar for JavaScript. Therefore, onClick is used instead of onclick.

<button onClick={() => alert('Button clicked!')}>Click Me</button>  
  
**Q7) what is Rendering ?**

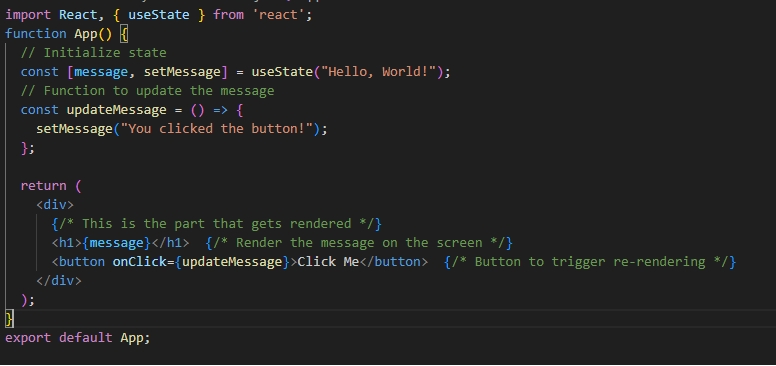
Yes, in React, **rendering** means showing or updating the content (like text, images, or UI elements) on the screen. When a React component is **rendered**, it takes the data (state or props) and converts it into HTML that is then displayed in the browser.

**Key Points about Rendering in React:**

1. **Initial Render:**
   * When a React component is used for the first time, React generates the initial HTML to display the content based on the component's state and props.
2. **Re-rendering:**
   * If the data (state or props) in the component changes, React will automatically update the HTML and show the new content without reloading the entire page. This is called **re-rendering**.
   * For example, if a user types something in a form, the component re-renders to show the updated input value.
3. **Rendering and the Virtual DOM:**
   * React uses a virtual DOM to optimize rendering. Instead of updating the entire web page, React only updates the parts that have changed, which makes the app faster.

**Example of Rendering in React:**

Let’s take a simple example where the component renders text on the screen, and the text changes when a button is clicked.



**What Happens Here:**

1. **Initial Render**: When the app first loads, the message state is "Hello, World!". React takes the JSX code:

jsx

Copy code

<h1>{message}</h1>

and renders it as:

html

Copy code

<h1>Hello, World!</h1>

So, "Hello, World!" is displayed on the screen.

1. **Re-rendering**: When you click the button, the updateMessage function is called, which changes the message state to "You clicked the button!". This triggers React to re-render the component:
   * React notices that the message has changed, so it re-renders only the <h1> element with the new text.
   * The screen is updated to display "You clicked the button!".

**Summary:**

In React, **rendering** is the process of displaying your component’s content (like text, images, buttons) on the screen. When the data in your component changes (through state or props), React automatically updates the view by **re-rendering** the component to reflect the latest data.

**Q8) what is State ?**

**State** is like a box that holds information about a component in your React app. This information can change over time, especially when users interact with your app.

**Key Points about State:**

1. **Dynamic Data**: State contains data that can change, like the score in a game, the text in an input field, or items in a shopping cart.
2. **Local to Component**: Each React component can have its own state. This means one component can keep track of its own information without affecting others.
3. **Updating State**: When something changes (like a user clicking a button), you can update the state. React then automatically updates what you see on the screen to match the new state.
4. **Using useState**: In functional components, you use the useState hook to create and manage state.

For example:  


In this example, count is the state that keeps track of how many times the button has been clicked. Every time you click the button, setCount updates the count, and React re-renders the component to show the new count.

**Summary:**

State is a way to manage and keep track of information in your React components, making your app interactive and responsive to user actions!

**Q9) Key Differences Between Before Hooks (Class Components) and After Hooks (Functional Components with Hooks)**

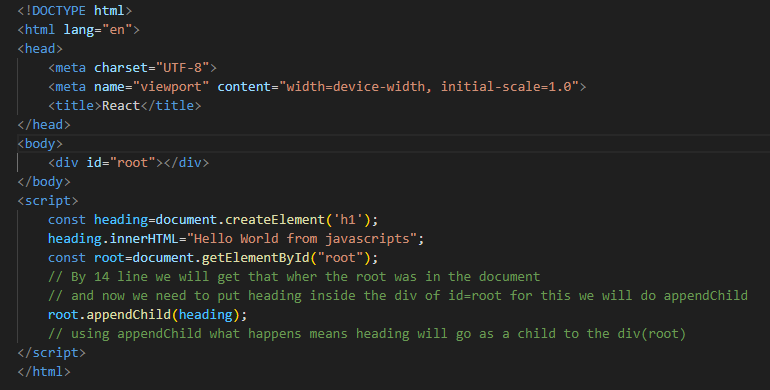
**Q10) Number of ways to execute or run a react code  
  
first way:**just copy paste CDN links in the <body>  
  
<script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>

<script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>

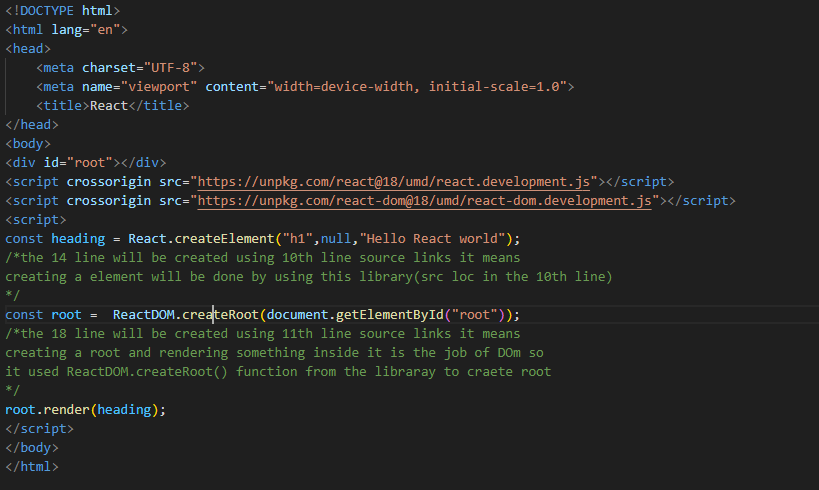
Because these contains the links in src=” “, which are used to host the react.  
As these links contains the source code of react which will be written and updated by facebook developers.  
  
So, finally react is just a java script file or a set of files/library created by facebook and we will just import it and use it . And one of the way to import is using CDN  
  
<script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>

This first src link file loc contains the core file of react like this is the core react frame work algorithm which was written inside .  
  
<script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>

The second line is react library which is useful for dom operations or this is the react DOM which we need to modify the DOM. This link acts like a bridge in between react and browsers.  
  
React works on both on phones(redact native) and also on the browsers and React 3D.so there is a separate file for react dom.  
  
=========================  
  
Its like facebook developers how are building this react and we are the developers are building web apps using this react   
============================

**Q11)First code in react in comparison with java script.**

This is a core java script code to print hello world.  
Here uses document.createElement--- > to create a new elements



Here uses React.createElement **-🡪 to create elements ,which takes 3 arguments**   
first one **is what tag you need**.  
Second one is **object🡪if we are not passing ,just full null or {}**third one is **what we have to put in that tag (like innerHTML=”” in javascript)   
  
Here we will use render() like appendChild in java script.  
  
=========================  
The most important operation in the webpage is when the DOM nodes to be manipulated (means we click some button on the webpage and something shows up into a page or something hides into a page what is happening is the DOM tree is getting changed) 🡪 (it simply means putting some nodes in dom and removing some nodes in dom) and all these frameworks and libraries are trying to optimize this.  
  
React also come with the philosophy that whenever you need to do anything on a web page do it using javascript. for this is why 🡪react gives us the helper functions to make these things work in a very performant way.  
  
=================================**  
When we want to give attribute to the h1 tag in the above then we will use **object** (the 2nd argument in the React.createElement) to pass the value.  
  
At the end heading is just a react element 🡪where **react element** is nothing but a normal **Java script object.**  
In the above example the render method is basically converting this object to h1 tag and put it on the browser  
=================================  
  
JSX will make easy when we have to create the tags  
  
  
  
 **SECOND VIDEO**