**JavaScript**

**1. Fundamentals**

**Explain the difference between var, let, and const.**

* **var**:
  + Function-scoped.
  + Can be redeclared and updated.
  + Hoisted to the top of its scope but initialized with undefined.
* **let**:
  + Block-scoped.
  + Cannot be redeclared in the same scope but can be updated.
  + Hoisted to the top of its block but not initialized.
* **const**:
  + Block-scoped.
  + Cannot be redeclared or updated.
  + Hoisted to the top of its block but not initialized.
  + Must be initialized at the time of declaration.

**What are closures and how do they work?**

Closures are functions that have access to their own scope, the scope of the outer function, and the global scope. They work by preserving the outer function's scope chain, even after the outer function has returned.

**Explain prototypal inheritance in JavaScript.**

Prototypal inheritance is a feature in JavaScript where objects inherit properties and methods from other objects. Each object has a prototype, and when a property or method is accessed, the JavaScript engine first looks at the object itself, then up the prototype chain until it finds the property or method or reaches the end of the chain.

**2. ES6+ Features**

**What are arrow functions and how do they differ from regular functions?**

Arrow functions are a more concise syntax for writing function expressions. They differ from regular functions in that they:

* Do not have their own this context.
* Do not have their own arguments object.
* Cannot be used as constructors (cannot use new).
* Do not have a prototype property.

**How do async and await work? What problem do they solve?**

async and await are used to handle asynchronous operations more gracefully than using promises alone. async functions return a promise, and await pauses the execution of the function until the promise is resolved or rejected. This makes asynchronous code look and behave more like synchronous code, making it easier to read and maintain.

**3. DOM Manipulation**

**How do you add an event listener in JavaScript?**

You can add an event listener using the addEventListener method. Example:

javascript

Copy code

element.addEventListener('click', function() {

// code to execute when the element is clicked

});

**Explain event delegation and its advantages.**

Event delegation is a technique where a single event listener is added to a parent element to manage events for its child elements. This works by taking advantage of event bubbling. Advantages include:

* Improved performance by reducing the number of event listeners.
* Easier management of dynamically added elements.

**4. Asynchronous JavaScript**

**What are promises? How do they differ from callbacks?**

Promises are objects that represent the eventual completion or failure of an asynchronous operation. They provide a cleaner and more manageable way to handle asynchronous operations compared to callbacks. Unlike callbacks, promises can be chained and provide better error handling.

**What is fetch and how is it used for making HTTP requests?**

fetch is a modern API for making network requests. It returns a promise that resolves to the response of the request. Example:

javascript

Copy code

fetch('https://api.example.com/data')

.then(response => response.json())

.then(data => console.log(data))

.catch(error => console.error('Error:', error));

**5. Error Handling**

**How can you handle errors in JavaScript?**

Errors can be handled using try...catch blocks, promises, and async/await with try...catch.

**What is try...catch and how does it work?**

try...catch is a statement that lets you handle exceptions. Code that might throw an error is placed inside the try block, and error handling is done in the catch block. Example:

javascript

Copy code

try {

// code that may throw an error

} catch (error) {

// code to handle the error

}

**HTML**

**1. Basic Tags and Attributes**

**What are semantic HTML elements? Give examples.**

Semantic HTML elements clearly describe their meaning in a human- and machine-readable way. Examples include <header>, <footer>, <article>, <section>, and <nav>.

**Explain the difference between <div> and <span>.**

* **<div>**: A block-level element used to group larger chunks of content or elements.
* **<span>**: An inline-level element used to group smaller chunks of text or elements within a block.

**2. Accessibility**

**What is ARIA (Accessible Rich Internet Applications)? Why is it important?**

ARIA is a set of attributes that define ways to make web content and applications more accessible to people with disabilities. It is important because it helps ensure that dynamic content is usable by assistive technologies.

**How can you improve the accessibility of a website?**

* Use semantic HTML.
* Provide alternative text for images.
* Ensure proper tab order and focus management.
* Use ARIA roles and attributes appropriately.
* Ensure sufficient color contrast.

**CSS**

**1. Selectors and Specificity**

**Explain CSS selectors and specificity.**

CSS selectors are patterns used to select elements to apply styles to. Specificity determines which styles are applied when there are conflicting rules. It is calculated based on the types of selectors used: inline styles, IDs, classes, attributes, and element types.

**How can specificity be overridden?**

Specificity can be overridden by using more specific selectors, !important, or inline styles.

**2. Box Model**

**What is the CSS box model? Explain briefly.**

The CSS box model describes how elements are structured and displayed, including margins, borders, padding, and the content area.

**How can you change the box model behavior using CSS properties?**

You can change the box model behavior using the box-sizing property. For example:

css

Copy code

element {

box-sizing: border-box; /\* includes padding and border in the element's total width and height \*/

}

**3. Layout and Flexbox/Grid**

**What are Flexbox and Grid layout systems? How do they differ?**

* **Flexbox**: A one-dimensional layout system for arranging items in a row or column.
* **Grid**: A two-dimensional layout system for arranging items in rows and columns.

**When would you use Flexbox over Grid, and vice versa?**

* Use Flexbox for simpler, one-dimensional layouts (e.g., navigation bars, alignments).
* Use Grid for more complex, two-dimensional layouts (e.g., web page layouts, dashboards).

**4. Responsive Design**

**How can you make a website responsive using CSS?**

* Use relative units (e.g., percentages, em, rem).
* Implement flexible grid layouts.
* Use media queries to apply styles based on screen size and other characteristics.

**HTML/CSS**

**What is the box model in CSS?**

The CSS box model describes how elements are structured and displayed, including margins, borders, padding, and the content area. The model consists of:

* **Content**: The actual content of the element.
* **Padding**: Space between the content and the border.
* **Border**: A border surrounding the padding (if any) and content.
* **Margin**: Space outside the border.

**How do you implement responsive design?**

Responsive design can be implemented using:

* **Relative units** (e.g., percentages, em, rem).
* **Flexible grid layouts** (e.g., Flexbox, CSS Grid).
* **Media queries** to apply styles based on screen size and other characteristics.
* **Viewport meta tag** to control layout on mobile browsers.

**What is the difference between inline and block elements?**

* **Inline elements**: Do not start on a new line and only take up as much width as necessary (e.g., <span>, <a>, <img>).
* **Block elements**: Start on a new line and take up the full width available (e.g., <div>, <p>, <h1>).

**How do you optimize images for web use?**

* **Choose the right format** (e.g., JPEG for photos, PNG for images with transparency, SVG for scalable vector graphics).
* **Compress images** using tools like TinyPNG or ImageOptim.
* **Use responsive images** with srcset and sizes attributes.
* **Lazy load images** to defer loading off-screen images.

**JavaScript**

**What is the difference between null and undefined?**

* **null**: Represents the intentional absence of any object value. It is an assignment value.
* **undefined**: Indicates that a variable has been declared but has not yet been assigned a value.

**What is the event loop?**

The event loop is a mechanism that allows JavaScript to perform non-blocking operations by offloading operations to the system kernel whenever possible. It continuously checks the message queue and executes any pending tasks, allowing asynchronous callbacks to be executed.

**What is the difference between var, let, and const?**

* **var**: Function-scoped, can be redeclared, hoisted to the top of its scope but initialized with undefined.
* **let**: Block-scoped, cannot be redeclared in the same scope but can be updated, hoisted but not initialized.
* **const**: Block-scoped, cannot be redeclared or updated, must be initialized at the time of declaration.

**What is the difference between == and ===?**

* **==**: Loose equality operator that performs type coercion if the types are different before comparing.
* **===**: Strict equality operator that does not perform type coercion and only returns true if both the value and type are the same.

**What is the difference between Promises and Async/Await?**

* **Promises**: Objects representing the eventual completion or failure of an asynchronous operation, providing methods like .then() and .catch().
* **Async/Await**: Syntactic sugar built on promises, allowing asynchronous code to be written in a more synchronous-like manner. async functions return a promise, and await pauses the execution until the promise is resolved or rejected.

**React**

**What is the Virtual DOM and how does it work?**

The Virtual DOM is an in-memory representation of the real DOM elements generated by React components. When the state of a component changes, a new virtual DOM is created. React then compares this with the previous virtual DOM using a process called reconciliation and updates only the changed parts in the real DOM, improving performance.

**How do you optimize the performance of a React application?**

* **Use React.memo** to prevent unnecessary re-renders.
* **Code splitting** with React.lazy and Suspense.
* **Use React.PureComponent** or React.memo.
* **Optimize component rendering** by using the useCallback and useMemo hooks.
* **Avoid inline functions and objects** in render methods.

**What is the difference between state and props?**

* **State**: Data managed within a component, typically mutable and managed by the component itself using useState or this.setState.
* **Props**: Data passed from parent components to child components, read-only and cannot be modified by the receiving component.

**How do you handle side effects in React?**

Side effects are handled using the useEffect hook in functional components or lifecycle methods (componentDidMount, componentDidUpdate, componentWillUnmount) in class components.

**What is the difference between a controlled and uncontrolled component?**

* **Controlled Component**: A component where form data is handled by the state within the component.
* **Uncontrolled Component**: A component where form data is handled by the DOM itself, using refs to access form values.

**Frontend Build Tools**

**What is Webpack and how does it work?**

Webpack is a module bundler for JavaScript applications. It processes and bundles multiple files (JavaScript, CSS, images, etc.) into one or more output files, optimizing the build process. It uses a configuration file to define entry points, loaders, and plugins.

**How do you optimize the build process for a frontend application?**

* **Code splitting**: Split the code into smaller chunks to load only the necessary parts.
* **Tree shaking**: Remove unused code during the bundling process.
* **Minification and compression**: Reduce the size of the output files.
* **Caching**: Enable caching for static assets.
* **Use production mode**: Enable optimizations for production builds.

**What is the difference between a bundler and a transpiler?**

* **Bundler**: Combines multiple files into a single output file, optimizing and managing dependencies (e.g., Webpack, Rollup).
* **Transpiler**: Converts code from one language or version to another (e.g., Babel converts ES6+ code to ES5 for browser compatibility).

**How do you implement tree shaking and code splitting?**

* **Tree shaking**: Configure Webpack to use the mode: 'production' setting and ensure that modules are written using ES6 import/export syntax.
* **Code splitting**: Use dynamic imports with import() syntax and Webpack's SplitChunksPlugin.

**What is the difference between a dev and prod build?**

* **Dev build**: Includes source maps, unminified code, and additional debugging information for development purposes.
* **Prod build**: Optimized for performance, with minified code, removed development-only features, and optimizations enabled.

**State Management**

**What is state management and why is it important?**

State management is the practice of managing the state of an application, ensuring that the state is predictable, consistent, and easily accessible. It is important because it helps maintain the integrity of the application, simplifies debugging, and enables efficient data flow and updates.

**How do you implement state management with Redux or MobX?**

* **Redux**: Use actions, reducers, and a centralized store to manage state. Connect React components to the store using react-redux.
* **MobX**: Use observables, actions, and computed values to manage state. Connect React components to observables using mobx-react.

**What is the difference between a store and a context?**

* **Store**: A centralized state management solution (e.g., Redux store) that holds the entire state of the application and allows state updates through dispatched actions.
* **Context**: A built-in feature of React (React.createContext) that provides a way to pass data through the component tree without having to pass props down manually at every level.

**How do you handle side effects with state management?**

* **Redux**: Use middleware like redux-thunk or redux-saga to handle side effects.
* **MobX**: Use autorun and reaction functions to handle side effects.

**What is the difference between a reducer and an action?**

* **Reducer**: A pure function that takes the current state and an action as arguments and returns a new state based on the action type.
* **Action**: An object that describes a state change, typically containing a type property and optionally a payload.

**Frontend Security**

**What is Cross-Site Scripting (XSS) and how do you prevent it?**

XSS is an attack where malicious scripts are injected into trusted websites. Prevention methods include:

* **Input validation**: Validate and sanitize user input.
* **Output encoding**: Encode data before rendering it to the browser.
* **Use Content Security Policy (CSP)**: Define trusted sources for content.

**How do you implement authentication and authorization?**

* **Authentication**: Verify the identity of users using methods like passwords, OAuth, or multi-factor authentication.
* **Authorization**: Control access to resources based on user roles and permissions, often implemented using access control lists (ACLs) or role-based access control (RBAC).

**What is Cross-Site Request Forgery (CSRF) and how do you prevent it?**

CSRF is an attack where unauthorized commands are transmitted from a user that the website trusts. Prevention methods include:

* **Use anti-CSRF tokens**: Include unique tokens in forms and validate them on the server.
* **SameSite cookies**: Set cookies with the SameSite attribute to prevent cross-origin requests.

**How do you handle sensitive data in a frontend application?**

* **Use HTTPS**: Encrypt data transmitted between the client and server.
* **Store sensitive data securely**: Avoid storing sensitive data in local storage or session storage.
* **Sanitize and validate input**: Prevent injection attacks.

**What is the difference between HTTPS and HTTP?**

* **HTTP**: Hypertext Transfer Protocol, used for transmitting data over the web.
* **HTTPS**: HTTP Secure, an extension of HTTP that uses SSL/TLS to encrypt data, ensuring secure communication between the client and server.

**Browser APIs**

**What is the difference between localStorage and sessionStorage?**

* **localStorage**: Stores data with no expiration time. Data persists even after the browser is closed.
* **sessionStorage**: Stores data for the duration of the page session. Data is cleared when the page or tab is closed.

**What is the difference between a cookie and a token?**

* **Cookie**: Small pieces of data stored by the browser, sent with every HTTP request to the same domain. Used for session management and tracking.
* **Token**: Self-contained pieces of data (e.g., JWT) used for authentication and authorization. Stored in localStorage, sessionStorage, or cookies.

**How do you handle browser storage and caching?**

* **Browser storage**: Use localStorage, sessionStorage, IndexedDB, or cookies to store data.
* **Caching**: Use Cache API, Service Workers, and HTTP cache headers (Cache-Control, ETag) to manage cached resources.

**What is the difference between a web worker and a service worker?**

* **Web Worker**: Runs scripts in background threads, enabling multi-threading in JavaScript. Used for performing CPU-intensive tasks without blocking the main thread.
* **Service Worker**: A type of web worker that acts as a proxy between the web app and the network, enabling features like offline support, push notifications, and background sync.

**Frontend Frameworks**

**What is the difference between Angular, React, and Vue?**

* **Angular**: A full-fledged framework by Google, using TypeScript, with two-way data binding and a component-based architecture.
* **React**: A library by Facebook for building user interfaces, focused on component-based architecture and one-way data binding. Uses JSX for templating.
* **Vue**: A progressive framework that is easy to integrate with existing projects, featuring a component-based architecture, and reactive data binding.

**What is the difference between a library and a framework?**

* **Library**: A collection of pre-written code that can be used to implement specific functionality. You call libraries in your code.
* **Framework**: A structure that provides a foundation for developing applications. The framework calls your code.

**What is the difference between a template and a component?**

* **Template**: A chunk of HTML code that defines the structure of a part of the UI.
* **Component**: A reusable, self-contained piece of UI that includes a template, styles, and behavior (JavaScript logic).

**What is the difference between Promises and Observables?**

* **Promises**: Represent a single future value or error from an asynchronous operation. They are not cancellable and handle one-time events.
* **Observables**: Represent a stream of values or events over time. They are cancellable, lazy, and handle multiple values and events. They are a core part of the Reactive Programming paradigm, often used in frameworks like Angular with RxJS.

**React Basics**

**What is JSX in React?**

JSX stands for JavaScript XML. It allows you to write HTML elements in JavaScript and place them in the DOM without using methods like createElement or appendChild. JSX makes it easier to write and add HTML in React.

**Explain the purpose of state in React components.**

State is an object that holds information that may change over the lifetime of the component. It is managed within the component and can influence the component's rendering and behavior. State allows components to create dynamic and interactive user interfaces.

**How to pass data from Parent to Child component and vice-versa?**

* **Parent to Child**: Data is passed from a parent component to a child component using props.

jsx

Copy code

<ChildComponent data={this.state.data} />

* **Child to Parent**: You can pass data from child to parent by passing a function from the parent to the child as a prop. The child can then call this function with data as arguments.

jsx

Copy code

<ChildComponent sendData={this.handleData} />

**Explain the virtual DOM concept.**

The virtual DOM is an in-memory representation of the real DOM elements generated by React components. When the state of an object changes, the virtual DOM is updated instead of the real DOM. React then compares the virtual DOM with a snapshot taken before the update and determines the most efficient way to update the real DOM to match the virtual DOM.

**React Components**

**Differentiate between functional and class components in React.**

* **Functional Components**: These are simple functions that return JSX. They do not have lifecycle methods or state by default (until the introduction of hooks).

jsx

Copy code

function Greeting(props) {

return <h1>Hello, {props.name}</h1>;

}

* **Class Components**: These are ES6 classes that extend from React.Component and can have state and lifecycle methods.

jsx

Copy code

class Greeting extends React.Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

**Describe the lifecycle methods in a React class component.**

* **Mounting**: constructor(), static getDerivedStateFromProps(), render(), componentDidMount()
* **Updating**: static getDerivedStateFromProps(), shouldComponentUpdate(), render(), getSnapshotBeforeUpdate(), componentDidUpdate()
* **Unmounting**: componentWillUnmount()

**How can we achieve lifecycle methods in functional components?**

Functional components achieve lifecycle methods using hooks:

* useEffect can be used to handle side-effects like componentDidMount, componentDidUpdate, and componentWillUnmount.

jsx

Copy code

useEffect(() => {

// componentDidMount and componentDidUpdate logic here

return () => {

// componentWillUnmount logic here

};

}, [dependencies]); // dependencies array

**Difference between controlled and uncontrolled components.**

* **Controlled Components**: Components that derive their value from the component's state. They require onChange handlers to update the state.

jsx

Copy code

<input type="text" value={this.state.value} onChange={this.handleChange} />

* **Uncontrolled Components**: Components that manage their own state internally. They use refs to access the DOM elements.

jsx

Copy code

<input type="text" ref={this.inputRef} />

**What are pure components?**

Pure components in React are components that do not re-render if the props and state remain the same. They implement shouldComponentUpdate with a shallow prop and state comparison.

**React Hooks**

**Explain the use of the useState hook in React.**

The useState hook is a function that lets you add React state to functional components. It returns an array with the current state and a function to update that state.

jsx

Copy code

const [state, setState] = useState(initialState);

**What is the useEffect hook, and why is it used?**

The useEffect hook lets you perform side effects in functional components. It is used for tasks like data fetching, setting up subscriptions, and manually changing the DOM.

jsx

Copy code

useEffect(() => {

// effect logic here

return () => {

// cleanup logic here

};

}, [dependencies]);

**What is the difference between useCallback and useMemo hook?**

* **useCallback**: Returns a memoized callback function, useful to prevent unnecessary re-creations of functions.

jsx

Copy code

const memoizedCallback = useCallback(() => {

// callback logic

}, [dependencies]);

* **useMemo**: Returns a memoized value, useful to avoid costly calculations on every render.

jsx

Copy code

const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);

**Explain useContext hook.**

The useContext hook allows you to access the context value without needing a context consumer component. It simplifies consuming context in functional components.

jsx

Copy code

const value = useContext(MyContext);

**Redux**

**What problem does Redux solve in a React application?**

Redux provides a predictable state container for JavaScript apps, helping manage the application's state in a centralized manner. This makes state changes predictable, easier to debug, and manageable, especially in large applications.

**Explain the roles of actions, reducers, and the store in Redux.**

* **Actions**: Plain JavaScript objects with a type property that describe the type of action to perform and can carry additional data.

jsx

Copy code

const action = { type: 'ADD\_TODO', payload: { text: 'Learn Redux' } };

* **Reducers**: Pure functions that take the current state and an action and return a new state.

jsx

Copy code

function todos(state = [], action) {

switch (action.type) {

case 'ADD\_TODO':

return [...state, action.payload];

default:

return state;

}

}

* **Store**: The object that holds the application's state. It is created using the createStore function from Redux and allows access to the state via getState(), dispatching actions via dispatch(), and subscribing to changes via subscribe().

jsx

Copy code

const store = createStore(todos);

**Explain useReducer.**

useReducer is a hook that is an alternative to useState for managing complex state logic. It takes a reducer function and an initial state and returns the current state and a dispatch function.

jsx

Copy code

const [state, dispatch] = useReducer(reducer, initialState);

#### 1. What is a RESTful API?

* A RESTful API (Representational State Transfer) is an architectural style for designing networked applications. using HTTP requests for CRUD operations on resources.

1. **What are the main HTTP methods used in RESTful APIs?**
   * **GET:** Retrieve data.
   * **POST:** Submit data.
   * **PUT:** Update data completely.
   * **PATCH:** Update data partially.
   * **DELETE:** Remove data.
2. **What is the difference between PUT and PATCH?**
   * **PUT:** Updates the entire resource.
   * **PATCH:** Updates part of the resource.
3. **What are status codes in REST APIs, and why are they important?**
   * Status codes indicate the result of an HTTP request. Common codes:
     + **200:** OK
     + **404:** Not Found
     + **500:** Internal Server Error
4. **How do you handle authentication and authorization in RESTful APIs?**
   * Use token-based authentication like JWT, OAuth for secure access.
5. **What are some best practices for designing RESTful APIs?**
   * Version your APIs, handle errors gracefully, use pagination for large datasets, and use consistent naming conventions.

**Integration in React**

1. **How do you make an API call in React?**
   * Use fetch or libraries like axios. Example with fetch:

javascript

Copy code

useEffect(() => {

fetch('https://api.example.com/data')

.then(response => response.json())

.then(data => setData(data))

.catch(error => console.error('Error:', error));

}, []);

1. **What is the use of useEffect hook in fetching data in React?**
   * useEffect is used to perform side effects, like fetching data, after the component renders.
2. **How do you handle loading and error states in React when making API calls?**
   * Manage state for loading and error:

javascript

Copy code

const [loading, setLoading] = useState(true);

const [error, setError] = useState(null);

useEffect(() => {

fetch('https://api.example.com/data')

.then(response => response.json())

.then(data => {

setData(data);

setLoading(false);

})

.catch(error => {

setError(error);

setLoading(false);

});

}, []);

1. **How can you optimize API calls in a React application?**
   * Use techniques like debouncing for search inputs, caching responses, and memoizing expensive calculations.
2. **What is the role of context API in managing global state for API data in React?**
   * The Context API provides a way to pass data through the component tree without prop drilling, useful for sharing fetched data across components.
3. **How would you handle pagination and infinite scrolling in a React app?**
   * For pagination, use query parameters to fetch specific pages. For infinite scrolling, listen to the scroll event and load more data as the user scrolls down.
4. **How do you secure API keys in a React application?**
   * Store API keys in environment variables and avoid exposing them in the client code. Use server-side proxy if needed.
5. **What is the difference between client-side and server-side rendering with respect to API calls?**
   * **Client-side rendering:** API calls are made from the browser, improving interactivity but may have initial load delay.
   * **Server-side rendering:** API calls are made on the server, sending fully rendered pages to the client, improving initial load time and SEO.