REACT QUESTIONS AND ANSWERS

==========

🔹 Basics of React

Q1: What is React?

A: React is a JavaScript library developed by Facebook for building user interfaces, especially single-page applications.

Q2: What are the features of React?

A:

- Virtual DOM

- Component-based

- One-way data binding

- JSX

- High performance

Q3: What is JSX?

A: JSX is a syntax extension that allows mixing HTML with JavaScript.

Q4: What is the virtual DOM?

A: A lightweight copy of the actual DOM that React uses to detect changes and update efficiently.

🔹 Components & Props

Q5: What are components in React?

A: Components are reusable pieces of UI. Types: Functional and Class Components.

Q6: What are props?

A: Props are inputs to components, passed as attributes in JSX.

Q7: Can props be changed?

A: No, props are read-only.

🔹 State & Lifecycle

Q8: What is state in React?

A: A built-in object used to contain data or information about the component.

Q9: How to update state in React?

A: Use `setState()` in class components or `useState()` in functional components.

Q10: What are lifecycle methods?

A:

- Mounting: constructor, componentDidMount
- Updating: shouldComponentUpdate, componentDidUpdate

- Unmounting: componentWillUnmount

Here's a brief example demonstrating **Mounting, Updating, and Unmounting** lifecycle methods in a **class component**:

```jsx
import React, { Component } from 'react';

class LifeCycleExample extends Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
    console.log('Constructor: Initializing state and binding methods');
  }

  componentDidMount() {
    // Called after the component is mounted (DOM is ready)
    console.log('componentDidMount: Component has been mounted');
    // Here you can do things like API calls
  }

  shouldComponentUpdate(nextProps, nextState) {
```

```
    // Determines if the component should re-
render
    console.log('shouldComponentUpdate:
Checking if the component should update');
    return nextState.count !== this.state.count;
// Only update if count changes
  }

  componentDidUpdate(prevProps, prevState) {
    // Called after the component updates (re-
rendered)
    console.log('componentDidUpdate:
Component has updated');
  }

  componentWillUnmount() {
    // Called before the component is removed
from the DOM
    console.log('componentWillUnmount:
Cleaning up resources');
  }

  increment = () => {
    this.setState({ count: this.state.count + 1 });
  };

  render() {
```

```
        console.log('render: Rendering the component');
        return (
          <div>
            <p>Count: {this.state.count}</p>
            <button onClick={this.increment}>Increment</button>
          </div>
        );
      }
    }

    export default LifeCycleExample;
    ```

    ### Key Points:

    1. **constructor**: Initializes state and sets up any methods.
    2. **componentDidMount**: Runs once the component is mounted (useful for side effects like API calls).
    3. **shouldComponentUpdate**: Allows you to prevent unnecessary re-renders (e.g., if the `count` hasn't changed).
    4. **componentDidUpdate**: Called after each render, useful for reacting to state or prop changes.
    5. **componentWillUnmount**: Called just before the component is unmounted (used for cleanup like removing event listeners).
```
```

This pattern is very useful in class-based components to manage state, side effects, and optimize performance.

🔹 React Hooks

Q11: What are hooks in React?

A: Functions that let you use state and lifecycle features in functional components.

Q12: List commonly used hooks.

A:

1. **`useState`**: Allows you to add state to a functional component.

2. **`useEffect`**: Performs side effects in functional components, such as data fetching, subscriptions, or manual DOM manipulations.

3. **`useContext`**: Allows you to access the value of a React context within a functional component.

4. **`useRef`**: Provides a way to persist values across renders without causing a re-render, commonly used for accessing DOM elements.

5. **`useReducer`**: An alternative to `useState` for managing complex state logic in functional components, using a reducer function.

6. **`useMemo`**: Memoizes a value to avoid recalculating it on every render, optimizing performance for expensive calculations.

7. **`useCallback`**: Memoizes a function to avoid creating a new function instance on every render, optimizing performance in child components.

8. **`useLayoutEffect`**: Similar to `useEffect`, but runs synchronously after all DOM mutations, allowing for layout reads and writes.

9. **`useImperativeHandle`**: Customizes the instance value that is exposed to parent components when using `ref` with functional components.

🔹 Forms & Events

Q16: How to handle forms in React?

A: Use controlled components.

Q17: How to handle events in React?

A: Use camelCase for event names, pass handler functions.

```js
<button onClick={handleClick}>Click</button>
```

🔹 Context API & Redux

Q18: What is Context API?

A: A way to share state globally without prop drilling.

Q19: What is Redux?

A: A state management library for predictable state handling.

Q20: Key Redux concepts?

A: Store, Actions, Reducers, Dispatch

🔹 Routing

Q21: How do you implement routing in React?

A: Using React Router.

```js
<BrowserRouter>
 <Routes>
  <Route path="/home" element={<Home />} />
 </Routes>
</BrowserRouter>
```

🔹 Testing in React

Q22: What are common tools?

A: Jest, React Testing Library, Enzyme

Q23: How to test a component?

A:

- Render component

- Simulate events

- Assert results

```js

render(<Button />);

fireEvent.click(screen.getByText('Click'));

expect(screen.getByText('Clicked')).toBeInTheDocument();

```

Q24: How to mock API calls?

A: Use `jest.mock()` and mock resolved values.

ANGULAR QUESTIONS AND ANSWERS

==========

🔹 Basics of Angular

Q1: What is Angular?

A: Angular is a TypeScript-based open-source web application framework developed by Google for building client-side applications.

Q2: Key features of Angular?

A:

- Components and Modules for UI and structure.

- Dependency Injection for managing services.

- TypeScript support for strong typing.

- RxJS for reactive programming.

- Ahead-of-Time (AOT) compilation for faster rendering.

🔹 Components, Modules, and Services

Q3: What are components in Angular?

A: Components are the building blocks of Angular applications, containing a TypeScript class, HTML template, and CSS styles.

Q4: What is a module?

A: A module is a container that organizes related components, services, and other code into cohesive units for easier management.

Q5: What is a service?

A: A service is a class used to encapsulate and share business logic and data between different components.

Q6: How to inject a service?

A:

```ts
constructor(private service: MyService) {}
```

🔹 Data Binding & Directives

Q7: Types of Data Binding?

A:

- **Interpolation**: `{{ data }}`
- **Property Binding**: `[property]="value"`
- **Event Binding**: `(event)="handler()"`
- **Two-way Binding**: `[(ngModel)]="value"`

Q8: What are directives?

A:

- **Structural Directives**: Change the structure of the DOM (e.g., `*ngIf`, `*ngFor`).
- **Attribute Directives**: Modify the appearance or behavior of DOM elements (e.g., `[ngClass]`, `[ngStyle]`).

🔹 Angular Forms

Q9: Types of forms in Angular?

A:

- **Template-driven Forms**: Forms created using directives in the template.

- **Reactive Forms**: Forms built programmatically with FormGroup and FormControl.

Q10: How to use Reactive Forms?

A:

```ts
form = new FormGroup({
  name: new FormControl('')
});
```

🔹 Angular Routing

Q11: How to define routes?

A:

```ts
const routes: Routes = [
```

```
  { path: 'home', component: HomeComponent }
];
```

Q12: What is lazy loading?

A: Lazy loading allows feature modules to be loaded on demand, improving the initial load time of the application.

🔹 Testing in Angular

Q13: Tools for Angular testing?

A: The primary tools for testing in Angular are **Jasmine** (for writing tests), **Karma** (for running tests in browsers), and **TestBed** (for configuring tests).

Q14: How to test services?

A:

```ts
it('should return data', () => {
  service.getData().subscribe(data => {
        expect(data).toEqual(mockData);
  });
});
```

Q15: How to test components?

A:

- Configure TestBed with component and dependencies.

- Create component instance.

- Trigger change detection.

- Assert DOM updates using Jasmine matchers.

🔹 Advanced Angular Topics

Q16: What is RxJS?

A: RxJS (Reactive Extensions for JavaScript) is a library that enables reactive programming using Observables for asynchronous data streams.

Q17: What is Change Detection?

A: Change Detection is the process Angular uses to keep the view in sync with the model by checking component properties and updating the DOM when needed.

Q18: What is Angular Universal?

A: Angular Universal is a server-side rendering tool for Angular apps, allowing for faster initial load times and better SEO.

Q19: What is Ahead-of-Time (AOT) compilation?

A: AOT compilation compiles Angular templates and TypeScript code into JavaScript during the build process, improving startup performance.

Q20: What are Standalone Components?

A: Standalone Components, introduced in Angular 14, do not require being declared in an `NgModule` and can be used directly in routes.

🔹 Lifecycle Hooks

Q21: What are lifecycle hooks in Angular?

A: Lifecycle hooks are methods called at different stages of a component's lifecycle, allowing developers to execute code at specific points.

Q22: What are common lifecycle hooks in Angular?

- **ngOnInit**: Called after the component is initialized and the input properties are set.
- **ngOnChanges**: Called when the input properties change.
- **ngDoCheck**: Called during every change detection cycle.
- **ngAfterContentInit**: Called after Angular projects content into the component's view.
- **ngAfterContentChecked**: Called after Angular checks the content projected into the component.
- **ngAfterViewInit**: Called after the component's view and child views are initialized.
- **ngAfterViewChecked**: Called after the component's view and child views are checked.
- **ngOnDestroy**: Called just before the component is destroyed.

🔹 Observables and Subscription

Q23: What are Observables in Angular?

A: Observables are streams of data that can be subscribed to in order to receive updates over time, often used in combination with RxJS for handling asynchronous data.

Q24: How do you subscribe to an Observable?

A:

```ts
observable$.subscribe(data => {
  console.log(data);
});
```

Q25: What is the purpose of `unsubscribe()` in Angular?

A: `unsubscribe()` is used to cancel the subscription to an Observable, preventing memory leaks when a component is destroyed.
