# DWA_12 Knowledge Check

To complete this Knowledge Check, ensure you have worked through all the lessons in **Module 12: Declarative Abstractions.**

To prepare for your session with your coach, please answer the following questions. Then download this document as a PDF and include it in the repository with your code.

---

1. What are the benefits of direct DOM mutations over replacing HTML?

Direct DOM mutations refer to making changes to the Document Object Model (DOM) of a web page without completely replacing or re-rendering the HTML content. Benefits:

- Performance: Direct DOM mutations can be more efficient in terms of performance compared to completely replacing HTML. When you modify specific elements or properties within the DOM, the browser only needs to update those specific parts, reducing the overall processing and rendering time
- Granular Updates: With direct DOM mutations, you have fine-grained control over the updates you want to make. Instead of replacing entire sections of HTML, you can target and modify individual elements or attributes.
- State Preservation: By directly modifying the DOM, you can preserve the current state of the web page
- Interactivity: Direct DOM mutations can facilitate real-time updates and interactivity without reloading the entire page. This is particularly useful in scenarios where dynamic data needs to be updated frequently, such as live chat applications, real-time dashboards, or collaborative editing tools.
- Compatibility: Direct DOM mutations work well with modern JavaScript frameworks and libraries that adopt a virtual DOM or reactive approach. These frameworks efficiently update the DOM based on changes in data or application state, minimizing the need for full HTML replacement.
- Integration: When working with third-party libraries or widgets, direct DOM mutations can be advantageous. They allow you to integrate and control specific parts of the DOM without affecting the rest of the page structure or functionality.

---

2. What low-level noise do JavaScript frameworks abstract away?

DOM Manipulation: JavaScript frameworks abstract the low-level DOM manipulation operations, such as creating, updating, and deleting DOM elements. They provide higher-level APIs and abstractions that make it easier to interact with the DOM, allowing developers to focus on application logic rather than manual DOM manipulation.

Event Handling: Frameworks handle the complexity of event binding and event delegation. They provide convenient mechanisms to attach event listeners to elements, handle event propagation, and manage event subscriptions. This abstraction simplifies the process of responding to user interactions and handling events in a cross-browser compatible manner.

Browser Compatibility: JavaScript frameworks abstract away the differences and inconsistencies across various browsers. They provide unified APIs that work consistently across different browser environments, shielding developers from the need to write browser-specific code or deal with browser quirks.

Asynchronous Operations: Frameworks abstract the complexities of managing asynchronous operations, such as AJAX requests or Promise-based operations. They provide utilities and abstractions that simplify asynchronous programming, including features like promises, async/await syntax, or reactive programming paradigms.

State Management: Many JavaScript frameworks offer built-in mechanisms for managing application state. They provide abstractions and patterns for handling data flow and state updates, which reduces the complexity of managing application state manually. This includes features like data binding, observables, or state containers.

Routing: Frameworks often provide routing capabilities, abstracting away the low-level implementation details of handling URL changes and navigation. They offer declarative routing configurations or routing APIs that simplify the creation of routes, route parameters, and navigation between different views or components.

Component Abstraction: JavaScript frameworks typically introduce component-based architectures that abstract away the complexities of building user interfaces. They provide abstractions for creating reusable and modular components, handling component lifecycle, managing component state, and enabling component composition.

---

3. What essence do JavaScript frameworks elevate?

JavaScript frameworks elevate the logic behind the transformation of data. By handling things like DOM manipulations and cross-browser compatibility, they allow the developer to focus on just the logic of what changes need to be made, rather than on how they need to be made. Furthermore, frameworks elevate the state of the app, by including state management tools that centralize and synchronize data, allowing the developer to easily get an overall sense of the state of the app and the factors affecting it. They also elevate the presentation of components as single-responsibility, modular, reusable units, using polymorphism to change only the unique features that are relevant to that component's specific function.

---

4. Very broadly speaking, how do most JS frameworks achieve abstraction?

Most JavaScript frameworks achieve abstraction using classes and interfaces to create applications. They will generally include a set of intuitive and declarative APIs that allow developers to interact with various functionalities (like DOM manipulation, HTML templating, event handling, data binding, browser compatibility, and state management) without needing to understand the underlying implementation. Many JavaScript frameworks also use a component-based architecture, where developers build reusable and self-contained components that encapsulate HTML, CSS, and JavaScript code, along with their associated behaviors and data. The framework handles the rendering, lifecycle management, and communication between components, abstracting away the complexities of integrating and interacting with components Frameworks also allow developers to extend their functionality through middleware or plugins that can provide additional features, integrate with third-party libraries, or offer custom functionality specific to the framework.

---

5. What is the most important part of learning a JS framework?

Understanding the underlying abstractions and principles upon which the framework is based. Without knowing what the framework is doing in the background, you are not able to effectively debug should any issues arise. Furthermore, you will not be able to tell whether the framework is even a good fit for the kind of project you are trying to use it for. You need to understand what problems the framework is aiming to solve, because there are always tradeoffs that will need to be made.