

Courses ▾Placement ▾Data Science ▾GATE ▾

IBM

æ

Q Search... X

🌙🔔20📄

A

React-Introductions100%◀

Articles (2)

Why React Over JS✔

Last Updated: 2025-03-05

React App Walkthrough✔

Last Updated: 2025-03-05

ArticlesVideos

Articles Read  
2 of 2 Complete. (100%)

Progress may take upto 2 hours to reflect.

## Why React Over JS

React is a popular **JavaScript library** for building dynamic user interfaces, specifically focusing on the "view" layer of applications. React makes it easier to create reusable components and manage complex state interactions. While JavaScript offers flexibility, React introduces enhancements that simplify and optimize the development process, especially in large applications.

### Why React Over JS

#### 1. DOM Manipulation:

In traditional JavaScript, updating the DOM (Document Object Model) can be cumbersome. Every time an update is needed, developers manually interact with the DOM, which can result in complicated, error-prone code.

##### Challenges with Vanilla JavaScript DOM Manipulation:

- Complex code when updating elements.
- Difficulty in maintaining performance, especially with dynamic, large-scale apps.
- Debugging issues due to direct DOM manipulation.

##### React Improvement: Virtual DOM:

React introduces the **Virtual DOM**, an in-memory representation of the actual DOM. Instead of directly manipulating the DOM, React updates only the necessary parts of the UI. This improves performance and simplifies the rendering process.

- React compares the Virtual DOM with the actual DOM.
- It then updates only the changed components, reducing the number of DOM manipulations.
- This process leads to better performance and faster rendering, especially in large-scale applications.

#### 2. Unidirectional Data Flow:

Managing state and data flow in traditional JavaScript can be tricky, especially when dealing with bidirectional data binding. This often leads to unexpected side effects and makes it harder to trace the flow of data through the application.

##### Challenges with Vanilla JavaScript:

- Bidirectional data binding can result in unpredictable state changes.
- Harder to debug data flow, especially as the application grows in complexity.

##### React Improvement: One-Way Data Binding:

React adopts a **unidirectional data flow**, meaning data only flows in one direction, from parent to child components.

- **Benefits:** Makes state management predictable and simpler to debug.
- React's state management is enhanced with libraries like **Flux** or **Redux** for **centralized state management**, further improving maintainability.

#### 3. Componentization:

Vanilla JavaScript can make it difficult to create reusable, modular components, leading to larger, harder-to-maintain codebases. This also makes collaboration and code reuse more complicated.

##### Challenges with Vanilla JavaScript:

- Lack of a standardized way to create reusable components.
- Difficulty maintaining consistency across the app when creating multiple instances of similar components.

##### React Improvement: Component-Based Architecture:

React promotes a **component-based architecture**, which makes it easy to create **self-contained, reusable components**.

- Components are **modular**, meaning you can write them once and reuse them throughout the application.
- Each component encapsulates specific functionality, improving the **maintainability** and **scalability** of your codebase.
- This modular approach improves **collaboration**, as teams can work on isolated components without interfering with other parts of the code.

#### 4. React is Declarative while Vanilla JS is Imperative

In imperative programming, developers explicitly write step-by-step instructions for the computer to follow. This involves detailing exactly how a task should be accomplished, often involving direct manipulation of the DOM.

For example, consider the imperative approach to updating the text content of an HTML element:

```
const element = document.getElementById('example');  
  
element.textContent = 'Hello, World!';
```

In this example, you're explicitly instructing the browser to find an element with the ID 'example' and update its text content.

**React (Declarative):** React, on the other hand, takes a declarative approach. Instead of specifying how tasks should be performed, developers declare what the UI should look like based on the current state of the application. React abstracts away the direct manipulation of the DOM by introducing a virtual representation of it.

For instance, consider the declarative approach to the same task using React:

```
const ExampleComponent = () => {  
  
  return <div>Hello, World!</div>;  
  
}
```

In this example, you declare the desired UI structure, and React takes care of updating the actual DOM accordingly. React maintains a virtual representation of the DOM (Virtual DOM) and efficiently updates only the necessary parts based on changes in the application state. This leads to cleaner, more readable code and simplifies the process of managing dynamic user interfaces.

The declarative nature of React makes it easier to understand, maintain, and scale applications, as developers focus on defining the desired outcome rather than manually specifying each step of the process.

### Prerequisites to Learn React

To get started with React, you should have a basic understanding of JavaScript and some key concepts. Here are the main prerequisites:

## 1. Variables and Data Types

- Understand how to declare variables (`let`, `const`, and `var`) and the different data types in JavaScript.

## 2. Functions

- Learn about functions, how to pass parameters, and how to return values from them.

## 3. Objects and Arrays

- Grasp how to work with objects and arrays, including their methods and properties.

## 4. ES6+ Features

Familiarize yourself with modern JavaScript features like:

- **Arrow Functions:** Understand the syntax and advantages of arrow functions.
- **Destructuring:** Learn how to destructure objects and arrays for concise code.
- **let and const:** Understand block-scoping using `let` and constant variables with `const`.
- **Classes:** Familiarize yourself with ES6 class syntax for object-oriented programming.

## 5. Asynchronous JavaScript

Understand how to handle asynchronous operations:

- **Promises:** To handle async operations.
- **Async/Await:** For cleaner, easier-to-read asynchronous code.

## 6. Scope and Closures

- Learn about lexical scope and closures, which are foundational for understanding how functions retain access to variables from their outer scope.

## 7. DOM Manipulation

- Get comfortable with the Document Object Model (DOM) and how to manipulate it using JavaScript.

## 8. Event Handling

- Learn how to handle events like clicks and form submissions using event listeners.

## 9. HTTP Requests and APIs

- Learn how to make **HTTP** requests (e.g., using `fetch` or `Axios`) to interact with backend services.
- Understand the basics of **RESTful APIs** and how to consume them in your applications.

Marked as Read

 Report An Issue

If you are facing any issue on this page. Please let us know.