# **JAVASCRIPT**

**Why do we need JavaScript?**

1. **Client-Side Interactivity:** JavaScript allows developers to create interactive elements on web pages. This includes things like form validation, dynamic content updates without refreshing the page (AJAX), interactive maps, sliders, and much more.
2. **Enhanced User Experience:** By using JavaScript, developers can create smoother and more responsive user interfaces. Actions such as animations, transitions, and smooth scrolling can all be implemented to enhance the overall user experience.
3. **Cross-Browser Compatibility:** JavaScript helps in ensuring that web applications work across different browsers and devices. Modern JavaScript frameworks and libraries often provide solutions for handling browser inconsistencies and optimizing performance.
4. **Server-Side Communication:** Through AJAX (Asynchronous JavaScript and XML) and modern frameworks like React, Angular, or Vue.js, JavaScript enables efficient communication with the server, allowing for dynamic updates and real-time data retrieval.
5. **Rich Web Applications:** JavaScript is crucial for building Single Page Applications (SPAs) where the entire application runs in the browser, providing a more desktop-like experience. Frameworks like React, Angular, and Vue.js facilitate the development of complex, interactive web applications.
6. **Community and Ecosystem:** JavaScript has a vast ecosystem of libraries, frameworks, and tools that simplify and accelerate development. This ecosystem fosters innovation and allows developers to leverage existing solutions to solve common problems.

**What are the data types in JavaScript?**

Primitive Data Type :(Number, Boolean, String, Null, Symbol, Bigint, Undefined).

* **Stored by Value:** Primitive data types (like numbers, strings, booleans, etc.) are stored directly in memory as simple, single values. When you assign a new value to a variable that holds a primitive type, you're directly replacing the old value with the new one.
* **Immutable:** Primitive values are immutable, meaning their values cannot be changed. If you modify a variable holding a primitive value, you're actually creating a new value in memory rather than modifying the original.

Non-Primitive/Complex: (Objects, Arrays, Functions).

* **Stored by Reference:** Objects in JavaScript (arrays, functions, and objects themselves) are stored by reference in memory. This means that when you assign an object to a variable, you're actually storing a reference (or a pointer) to the memory location where the object is stored.
* **Mutable:** Objects are mutable, meaning you can change their properties and values directly. Modifying an object's property does not create a new object; it modifies the existing object in memory.

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

**var:**

* **Scope:** Function-scoped. Variables declared with var are accessible throughout the function they are declared in, or globally if declared outside any function.
* **Reassignable:** Yes, values can be reassigned.
* **Redeclarable:** Yes, the same variable can be redeclared within the same scope.
* **Hoisting:** Variables declared with var are hoisted to the top of their scope but remain undefined until the line of initialization.

**let:**

* **Scope:** Block-scoped. Variables declared with let are only accessible within the block {} they are declared in.
* **Reassignable:** Yes, values can be reassigned.
* **Redeclarable:** No, the same variable cannot be redeclared within the same scope.
* **Hoisting:** Variables declared with let are hoisted, but they remain in a "temporal dead zone" until they are initialized.

**Const:**

* **Scope:** Block-scoped, like let.
* **Reassignable:** No, values cannot be reassigned. However, if the value is an object or array, its properties can still be modified.
* **Redeclarable:** No, the same variable cannot be redeclared within the same scope.
* **Hoisting:** Like let, it is hoisted but remains in the "temporal dead zone" until initialization.

**What is the difference between parameter and argument?**

* Parameters are used when defining a function.
* Parameters act as placeholders for the values that the function will receive.
* Arguments are used when calling (or invoking) a function.
* Arguments are the actual values supplied to the function.

**What is the difference between slice and splice ?**

* Creates a shallow copy of a portion of an array.
* Does not modify the original array.
* Syntax: array.slice(start, end)
* This is example for slice:
* let fruits = ["apple", "banana", "cherry", "date"];
* let slicedFruits = fruits.slice(1, 3); // Extracts from index 1 to 2 (not including 3)
* console.log(slicedFruits); // ["banana", "cherry"]
* console.log(fruits); // ["apple", "banana", "cherry", "date"]
* Adds or removes elements from an array.
* Modifies the original array.
* Syntax: array.splice(start, deleteCount, item1, item2, ...)
* This is example for splice:
* let fruits = ["apple", "banana", "cherry", "date"];
* let removedFruits = fruits.splice(1, 2, "blueberry", "kiwi"); // Removes 2 elements starting from index 1 and adds "blueberry" and "kiwi" console.log(removedFruits); // ["banana", "cherry"] console.log(fruits); // ["apple", "blueberry", "kiwi", "date"]

**Why do we use map method in array?**

The map method in JavaScript is used to create a new array populated with the results of calling a provided function on every element in the calling array. It is particularly useful for transforming or processing elements of an array in a concise and readable way. Here are some key points about the map method and Why it is used:

* **Transformation:** The primary use of map is to transform elements in an array. For example, you can use it to apply a function to each element, such as converting an array of strings to uppercase or calculating the square of each number in an array.
* **Immutability:** map does not modify the original array. Instead, it returns a new array with the transformed elements, which helps in maintaining immutability and avoiding side effects in your code.
* **Readability:** Using map often makes code more readable and concise compared to using a for loop. It clearly expresses the intention of applying a transformation to each element of the array.
* **Functional Programming:** map is a fundamental concept in functional programming, where functions are first-class citizens and operations are performed on data using pure functions. It encourages writing clean, declarative code.

**When we will use Key ?**

In JavaScript, particularly when using frameworks like React, the key property is used to give elements a stable identity. This is essential for efficient updating and rendering of lists of elements. Keys help React identify which items have changed, are added, or are removed, allowing for optimized re-rendering of the components.

* **Rendering Lists**: When you create a list of elements using an array, each element should have a unique key to ensure efficient updates.
* **Dynamic Data**: When the data that is used to generate components changes frequently, key ensures that the correct elements are updated.
* **Reordering Items**: When items in a list can be reordered, key helps maintain the correct order and avoid unnecessary re-renders.

**What are the uses of spread operator?**

* **Array Manipulation**: Copy, merge, and add elements to arrays.
* **Object Manipulation**: Copy, merge, and update objects.
* **Function Arguments**: Spread elements as function arguments.
* **String to Array Conversion**: Convert strings to arrays of characters.
* **Rest Parameters**: Collect multiple function arguments into an array.

The spread operator is a versatile tool that simplifies and enhances JavaScript code, making it more readable and efficient.

**What are promises in javaScript?**

Promises in JavaScript are objects that represent the eventual completion (or failure) of an asynchronous operation and its resulting value. They provide a way to handle asynchronous code in a more manageable and readable way compared to traditional callback-based approaches.

**Methods;**

(then,catch,finally)

**What is asynchronous?**

Asynchronous programming is a form of parallel programming that allows a unit of work to run separately from the main application thread and notifies the main thread of its completion, failure, or progress. This is crucial in modern web development due to the non-blocking nature it provides, allowing for a smoother and more responsive user experience.

**Why Use Asynchronous Programming?**

1. **Non-blocking Nature:**
   * Prevents the application from freezing while waiting for time-consuming operations to complete. For example, fetching data from a server won't block the user interface from updating or accepting user input.
2. **Better Performance:**
   * Allows multiple tasks to run concurrently, leading to better utilization of system resources and improved performance.
3. **Improved User Experience:**
   * Keeps the application responsive, providing a smoother and more seamless experience for users.

**What is DOM?**

DOM stands for Document Object Model. It is a programming interface for web documents that represents the structure of HTML or XML documents as a tree-like model. In simpler terms, the DOM is a structured representation of a web page's document where each element in the document is represented as an object. This allows scripts (like JavaScript) to dynamically access and manipulate the content, structure, and style of the document.

**What is event bubling and event capturing?**

Event bubbling is the default behavior in which an event is triggered on the target element and then propagates upward to its parent elements.

Event capturing is the is first captured by the outermost element and then propagates to the target element.

**Why do we need react and Why can’t we able to create application in JavaScript?**

React is a JavaScript library for building user interfaces, particularly single-page applications where you need a fast and interactive user experience. While you can certainly create applications using vanilla JavaScript (without libraries or frameworks like React), React offers several advantages that can make development faster, more efficient, and more maintainable. Here are some reasons Why developers often choose React over plain JavaScript:

**What is difference between imperative and declarative?**

Basically, the react is a declarative approach of programming if we make JavaScript project that is imperative approach of making application.

Imperative programming focuses on **how** to perform tasks. It involves giving the computer explicit, step-by-step instructions on What actions to take to achieve a desired outcome. This approach is more concerned with the flow of control and the sequence of operations.

* **Step-by-Step Instructions:** You write code that specifies the exact steps the computer should take.
* **Control Flow:** You have detailed control over the flow of the program using loops, conditionals, and other control structures.
* **Mutable State:** Variables and data structures are often mutable, meaning their state can change over time

Declarative programming focuses on **What** the outcome should be, rather than how to achieve it. It involves describing the desired result without explicitly listing the steps to achieve that result. This approach is more concerned with the logic of the computation rather than the control flow.

* **Describing Outcomes:** You specify What you want to achieve and let the underlying system determine how to achieve it.
* **Abstracted Control Flow:** The control flow is abstracted away, often managed by the language or framework itself.
* **Immutable State:** Emphasis on immutability and avoiding side effects, leading to more predictable and understandable code.

**What is setTimeout?**

* function: The function to be executed after the delay.
* delay: The time in milliseconds to wait before executing the function.

**What is setTimeInterval?**

* function: The function to be executed at each interval.
* interval: The time in milliseconds between each execution of the function.

**What is async?**

* An async function always returns a promise. If the function returns a value, the promise will be resolved with that value. If the function throws an exception, the promise will be rejected.
* Inside an async function, you can use the await keyword to pause execution and wait for a promise to settle (either fulfill or reject) before continuing.

**What is await?**

**await** makes a function wait for a Promise. The await keyword in JavaScript is used within async functions to pause the execution of the function until a Promise is settled (either fulfilled or rejected).

**Function Borrowing**

With the bind() method, an object can borrow a method from another object.

 **Single Thread**: JavaScript processes one task at a time in sequence using a single call stack, blocking further execution until the task finishes.

 **Asynchronous Execution**: JavaScript can handle multiple tasks non-blockingly via callbacks, promises, or async/await by using Web APIs and the event loop.

 **Multi-threading (Web Workers)**: Web Workers enable JavaScript to run tasks in parallel threads, allowing background processing without blocking the main thread.

**What is API ?**

In JavaScript, the fetch function is a powerful and modern way to make asynchronous HTTP requests to retrieve data from a server or to send data to it. It is part of the **Fetch API**, which is a replacement for the older XMLHttpRequest.

**Key Features of fetch:**

1. **Promises-based**: fetch returns a **promise** that resolves once the request completes. If the request is successful, the promise is resolved with a Response object; if it fails, the promise is rejected.
2. **Simplicity**: The API is simpler and cleaner compared to older methods, like XMLHttpRequest, and allows chaining with .then() and .catch().

**What is Hoisting?**

We can use variables and functions before they are declared in the code.

Example: console.log(x); // undefined var x = 5; console.log(x); /

**What is Closure?**

When a function is defined inside another function, the inner function has access to the variables of the outer function due to **lexical scoping**.

Example: **(**function outerFunction() { let outerVariable = 'I am outside!'; function innerFunction() { console.log(outerVariable); // Can access outerVariable } return innerFunction; } const closure = outerFunction(); closure(); // "I am outside!"**)**

**Types of Events in JavaScript:**

1. **Mouse Events**: These events are triggered by mouse actions.
   * click: Occurs when the user clicks on an element.
   * dblclick: Triggered when the user double-clicks on an element.
   * mousedown: Fired when the mouse button is pressed down.
   * mouseup: Triggered when the mouse button is released.
   * mousemove: Occurs when the mouse pointer moves over an element.
   * mouseenter / mouseleave: Triggered when the mouse enters or leaves an element, without bubbling.
   * mouseover / mouseout: Triggered when the mouse hovers over or leaves an element, with bubbling.
2. **Keyboard Events**: These events are related to keyboard input.
   * keydown: Triggered when a key is pressed down.
   * keyup: Occurs when a key is released.
   * keypress: Fired when a key is pressed (deprecated, use keydown instead).
3. **Form Events**: Events related to form elements, such as input fields, checkboxes, and buttons.
   * submit: Triggered when a form is submitted.
   * change: Fired when the value of an input or select element changes.
   * input: Triggered when the value of an input field is changed by the user.
   * focus: Occurs when an element gains focus.
   * blur: Fired when an element loses focus.
   * reset: Triggered when a form is reset.
4. **Touch Events** (primarily for mobile devices): These events are specific to touch interfaces.
   * touchstart: Fired when a finger touches the screen.
   * touchend: Triggered when the finger is lifted from the screen.
   * touchmove: Occurs when a finger is dragged across the screen.
   * touchcancel: Fired when the touch event is interrupted (e.g., when a pop-up appears).
5. **Drag and Drop Events**: These events deal with dragging and dropping items.
   * dragstart: Triggered when the user starts dragging an item.
   * drag: Fired repeatedly as the item is dragged.
   * dragover: Triggered when the dragged item is moved over a valid drop target.
   * drop: Occurs when the dragged item is dropped on a target.
   * dragend: Triggered when the drag operation ends.
6. **Clipboard Events**: These events deal with copying, cutting, and pasting.
   * copy: Fired when content is copied.
   * cut: Triggered when content is cut from a document.
   * paste: Occurs when content is pasted into a document.
7. **Media Events**: These events occur during media loading, playing, and other actions related to audio and video elements.
   * play: Triggered when media starts playing.
   * pause: Fired when media is paused.
   * ended: Occurs when media playback reaches the end.
   * timeupdate: Triggered when the playing position of the media changes.
8. **Window Events**: These events occur when actions affect the browser window or the DOM.
   * load: Fired when the whole page (including images, scripts, etc.) is fully loaded.
   * resize: Triggered when the browser window is resized.
   * scroll: Fired when the user scrolls through a page.
   * unload: Occurs when a document is unloaded (e.g., the user navigates away from the page).
9. **Focus and Blur Events**: These are events specifically tied to focusing on or losing focus from elements.
   * focusin: Similar to focus, but it bubbles up through the DOM.
   * focusout: Similar to blur, but it bubbles up through the DOM.
10. **Animation and Transition Events**: Events related to CSS animations and transitions.
    * animationstart: Triggered when a CSS animation starts.
    * animationend: Fired when a CSS animation ends.
    * transitionend: Occurs when a CSS transition has finished.
11. **Network Events**: These events relate to network status.
    * online: Triggered when the browser gains network connectivity.
    * offline: Fired when the browser loses network connectivity.
12. **What is the difference between fetch Api and Rest Api ?**

* Definition: The **Fetch API** is a built-in JavaScript interface used for making HTTP requests to servers. It allows you to asynchronously request data (e.g., using GET, POST, etc.) from a URL and handle responses in JavaScript.
* A **REST** (Representational State Transfer) **API** is a set of rules and architectural constraints for building APIs that communicate over HTTP. It defines how to interact with a server's resources (like GET, POST, PUT, DELETE operations) via a set of stateless URLs.

**Fetch API**: A JavaScript method for making HTTP requests.

**REST API**: A set of rules and conventions for building web APIs that interact over HTTP.

**What is the difference between spread operator and rest operator?**

**Spread Operator**: Expands arrays or objects into individual elements or properties.

**Rest Operator**: Collects multiple elements or properties into an array or object.

**What is the usage of RegExp in JavaScript ?**

RegExp (short for "Regular Expression") in JavaScript is a powerful tool used for pattern matching in strings. It allows you to search, match, and manipulate text using defined patterns, which can be used for a wide range of text-processing tasks such as validation, extraction, or replacement.

**Common Use Cases of RegExp:**

1. **Pattern Matching**: Find specific patterns within a string.
2. **Validation**: Check if a string matches a certain format (e.g., email address, phone number).
3. **Search and Replace**: Find and replace text within a string.
4. **Splitting Strings**: Break a string into an array based on a pattern.

**JAVASCRIPT RULES:**

1. No Javascript keywords name
2. Should not start with numbers.
3. No space & No -.
4. It’s case sensitive.
5. Use meaningful names.