### **Introduction to JavaScript**

* **Founder**: JavaScript was created by **Brendan Eich** in **1995** while working at Netscape Communications.
* **Reason for Creation**: It was developed to make web pages dynamic and interactive, complementing HTML and CSS.
* **Synchronous or Asynchronous**: JavaScript is **single-threaded** and synchronous by default. However, it has asynchronous capabilities (via features like callbacks, promises, and async/await) for handling time-consuming tasks without blocking the main thread.
* **Importance**:
  1. **Client-Side Scripting**: Runs directly in the browser.
  2. **Versatile**: Can be used for both front-end (e.g., React) and back-end (e.g., Node.js).
  3. **Popular Libraries and Frameworks**: React, Angular, Vue.js.
  4. **Interactivity**: Enables features like animations, forms, and event handling.

### **Variables in JavaScript**

JavaScript variables can be declared using let, var, or const.

#### **1. var**

* Function-scoped and can be re-declared.
* Allows hoisting but may lead to unexpected bugs.

var name = "John";

var name = "Doe"; // Re-declaration allowed

console.log(name); // Output: Doe

#### **2. let**

* Block-scoped and cannot be re-declared within the same scope.
* Prevents issues caused by var.

let age = 25;

// let age = 30; // Error: Identifier 'age' has already been declared

console.log(age); // Output: 25

#### **3. const**

* Block-scoped and used for constants.
* The value cannot be re-assigned, but objects can have their properties modified.

const PI = 3.14;

// PI = 3.15; // Error: Assignment to constant variable

const user = { name: "Alice" };

user.name = "Bob"; // Allowed

console.log(user); // Output: { name: 'Bob' }

### **Data Types**

JavaScript has **7 primitive types**:

1. **Number**
2. **String**
3. **Boolean**
4. **Undefined**
5. **Null**
6. **Symbol**
7. **BigInt**

#### **Examples:**

javascript

Copy

let num = 10; // Number

let str = "Hello"; // String

let isTrue = true; // Boolean

let notAssigned; // Undefined

let emptyValue = null; // Null

### **Type Conversion**

JavaScript allows converting data types using:

1. **Implicit Conversion**: Automatic conversion by JavaScript.
2. **Explicit Conversion**: Manually converting types.

#### **Examples:**

// Implicit Conversion

console.log("5" + 2); // Output: "52" (Number to String)

// Explicit Conversion

let num = Number("123"); // String to Number

console.log(num + 1); // Output: 124

### **Operators**

1. **Arithmetic Operators**: +, -, \*, /, %, \*\*
2. **Comparison Operators**: ==, ===, !=, !==, >, <, >=, <=
3. **Logical Operators**: &&, ||, !
4. **Ternary Operator**: condition ? expr1 : expr2

#### **Example:**

let a = 5, b = 10;

console.log(a + b); // Arithmetic: 15

console.log(a === b); // Comparison: false

console.log(a > 0 && b > 0); // Logical: true

console.log(a > b ? "A is larger" : "B is larger"); // Ternary: B is larger

### **Functions**

JavaScript functions allow code reuse and modular programming.

#### **Arrow Functions**

Introduced in ES6, they have a simpler syntax and do not bind their own this.

const greet = (name) => `Hello, ${name}`;

console.log(greet("Alice")); // Output: Hello, Alice

#### **Hoisting**

JavaScript hoists variable and function declarations to the top of their scope.

console.log(add(5, 3)); // Output: 8

function add(a, b) {

return a + b;

}

#### **Closures**

A closure is a function that remembers its outer variables even after the outer function has finished executing.

javascript

Copy

function counter() {

let count = 0;

return function () {

count++;

return count;

};

}

const increment = counter();

console.log(increment()); // Output: 1

console.log(increment()); // Output: 2

### **Advanced Array Methods**

**map**: Creates a new array by applying a function to each element.  
  
const nums = [1, 2, 3];

const squared = nums.map(n => n \* n);

console.log(squared); // Output: [1, 4, 9]

**filter**: Creates a new array with elements that pass a condition.  
javascript  
Copy  
const nums = [1, 2, 3, 4];

const even = nums.filter(n => n % 2 === 0);

console.log(even); // Output: [2, 4]

**reduce**: Reduces an array to a single value.  
  
const nums = [1, 2, 3, 4];

const sum = nums.reduce((acc, curr) => acc + curr, 0);

console.log(sum); // Output: 10

### **Table 1: Variables in JavaScript**

| **Feature** | **var** | **let** | **const** |
| --- | --- | --- | --- |
| **Scope** | Function-scoped | Block-scoped | Block-scoped |
| **Re-declaration** | Allowed | Not allowed | Not allowed |
| **Re-assignment** | Allowed | Allowed | Not allowed |
| **Hoisting** | Yes (initialized as undefined) | Yes (not initialized) | Yes (not initialized) |
| **Usage** | Avoid due to potential issues | Preferred for variables that change | Preferred for constants or fixed values |

### **Table 2: Array Methods**

| **Method** | **Purpose** | **Example** | **Output** |
| --- | --- | --- | --- |
| **map** | Creates a new array by applying a function to each element | nums.map(n => n \* 2) | [2, 4, 6] (if nums = [1, 2, 3]) |
| **filter** | Creates a new array with elements that meet a condition | nums.filter(n => n > 2) | [3, 4] (if nums = [1, 2, 3, 4]) |
| **reduce** | Reduces an array to a single value | nums.reduce((acc, curr) => acc + curr, 0) | 10 (if nums = [1, 2, 3, 4]) |
| **forEach** | Executes a function for each element in the array | nums.forEach(n => console.log(n)) | Logs each element |
| **find** | Returns the first element that meets a condition | nums.find(n => n > 2) | 3 (if nums = [1, 2, 3]) |
| **some** | Checks if at least one element meets a condition | nums.some(n => n > 2) | true (if nums = [1, 2, 3]) |
| **every** | Checks if all elements meet a condition | nums.every(n => n > 0) | true (if nums = [1, 2, 3]) |
| **sort** | Sorts the elements of an array | nums.sort((a, b) => a - b) | [1, 2, 3] (if nums = [3, 2, 1]) |

### **Table 3: Functions in JavaScript**

| **Function Type** | **Definition** | **Example** | **Output** |
| --- | --- | --- | --- |
| **Regular Function** | Traditional function declaration | function add(a, b) { return a + b; } | add(2, 3) → 5 |
| **Arrow Function** | Concise syntax, no this binding | const add = (a, b) => a + b; | add(2, 3) → 5 |
| **Anonymous Function** | Function without a name | (function() { console.log("Hello"); })(); | Logs Hello |
| **Hoisting** | Functions are hoisted and can be used before declaration | console.log(add(2, 3)); function add(a, b) { return a + b; } | 5 |
| **Closures** | Functions that retain access to their outer scope even after the outer function is executed | function outer() { let count = 0; return function() { count++; return count; }; } | increment() → 1, increment() → 2 |