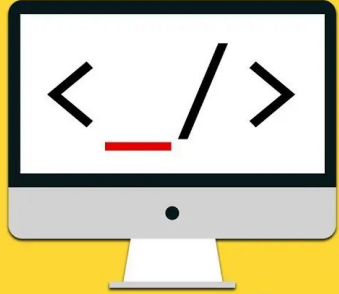


TIMES INTERNET

Introduction Rohit Azad



JavaScript

Hello everyone, my name is **Rohit Azad**, and I am a UI developer with over **14 years of experience** in **frontend development**.

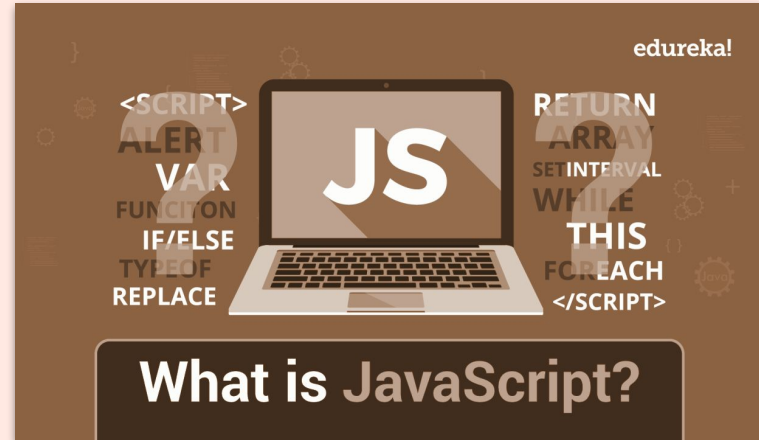
Introduction to JavaScript

What is JavaScript?

Definition: JavaScript is a high-level, interpreted programming language that conforms to the ECMAScript specification. It is a core technology of the World Wide Web, alongside HTML and CSS, and enables interactive web pages.

Dynamic: JavaScript is dynamically typed, meaning variable types are determined at runtime.

Event-Driven: JavaScript can respond to user actions and other events.



Introduction to JavaScript

History and Evolution

1995: Creation: Inventor: Brendan Eich, a programmer at Netscape Communications.

Original Name: Mocha, later renamed to LiveScript, and finally JavaScript.

Purpose: Designed to make web pages interactive, complementing HTML (for structure) and CSS (for styling).

1996: Standardization

ECMAScript: JavaScript was submitted to ECMA International, resulting in the ECMAScript standard (ES1).

Browser Wars: Netscape Navigator and Microsoft Internet Explorer competed, leading to rapid JavaScript evolution.

Introduction to JavaScript

History and Evolution

2005: AJAX Revolution :- AJAX (Asynchronous JavaScript and XML): Enabled dynamic content updates without refreshing the entire page, popularized by Google with applications like Google Maps and Gmail.

2009: Node.js :- Introduction of Node.js: Ryan Dahl created Node.js, allowing JavaScript to run on the server-side. This significantly expanded the language's capabilities and use cases.

NPM (Node Package Manager): Launched alongside Node.js, enabling developers to share and manage JavaScript libraries easily.

TIMES INTERNET

Introduction to JavaScript

History and Evolution

2015: ECMAScript 6 (ES6) :- Major Update: Also known as ECMAScript 2015 or ES6, this update introduced significant features such as classes, modules, arrow functions, template literals, and more.

Modern JavaScript: ES6 marked a new era for JavaScript, making it more powerful and easier to use for large-scale applications.

2016-Present: Continuous Evolution :- **Annual Updates:** ECMAScript continues to receive yearly updates, adding new features and improvements.

ESNext: Refers to the latest version of ECMAScript being developed.

Popular Libraries and Frameworks: Ecosystem growth with the development of libraries and frameworks like React, Angular, Vue.js, and more.

Introduction to JavaScript

Basic Syntax

Var :- Scope: Function-scoped.

Hoisting: Variables declared with var are hoisted to the top of their scope and can be accessed before they are declared (with the value undefined).

Re-declaration: Variables can be re-declared within the same scope.

```
var x = 10;  
console.log(x); // Output: 10  
  
var x = 20; // Re-declaration is allowed  
console.log(x); // Output: 20
```

Introduction to JavaScript

Basic Syntax

let :- Scope: Block-scoped.

Hoisting: Variables declared with let are also hoisted but cannot be accessed before they are declared (**Temporal Dead Zone**).

Re-declaration: Variables cannot be re-declared within the same scope

```
let y = 10;
console.log(y); // Output: 10

y = 20; // Reassignment is allowed
console.log(y); // Output: 20

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

Introduction to JavaScript

Basic Syntax

const :- Scope: Block-scoped.

Hoisting: Variables declared with const are hoisted but cannot be accessed before they are declared (Temporal Dead Zone).

Re-declaration: Variables cannot be re-declared within the same scope.

Assignment: Must be initialized during declaration and cannot be reassigned.

```
const z = 10;  
console.log(z); // Output: 10 //
```

```
z = 20; // Error: Assignment to  
constant variable //
```

```
const z = 30; // Error: Identifier 'z'  
has already been declared
```


Valid Declarations in JavaScript

In JavaScript, variable names (also known as identifiers) must follow certain rules. Here are the key points for valid variable declarations:

Start with a Letter, Underscore, or Dollar Sign: Variable names must begin with a letter (a-z or A-Z), an underscore (`_`), or a dollar sign (`$`).

Subsequent Characters: After the first character, variable names can include letters, digits (0-9), underscores, or dollar signs.

Case Sensitivity: Variable names are case-sensitive, which means `myVariable` and `myvariable` are different variables.

Reserved Keywords: Variable names cannot be reserved keywords (e.g., `let`, `class`, `return`, `function`, etc.).

TIMES INTERNET

Valid Declarations in JavaScript

```
var x = 10;  
var _name = "John";  
var $amount = 100;  
var user1 = "Alice";  
var user_name = "Bob";  
var $total_Amount2 = 200;
```

TIMES INTERNET

JavaScript used

```
<!DOCTYPE html>
<html>
<head>
  <title>Internal JavaScript Example</title>
  <script type="text/javascript">
    function showMessage() {
      alert("Hello, this is an internal JavaScript example!");
    }
  </script>
</head>
<body>
  <button onclick="showMessage()">Click me</button>
</body>
</html>
```

JavaScript used external file

Maintainability: External JavaScript files are easier to maintain and debug since they separate content (HTML) from behavior (JavaScript).

Reusability: External JavaScript files can be reused across multiple HTML pages, reducing code duplication.

Performance: Browsers can cache external JavaScript files, improving load times for users revisiting the site. However, internal JavaScript can be beneficial for small scripts that load faster inline.

Organization: Keeping JavaScript in external files helps in organizing code, making it more readable and manageable, especially in large projects.

Best Practices: For production code, it is generally recommended to use external JavaScript files to enhance the separation of concerns, promote reusability, and improve site performance.

JavaScript Data Types

JavaScript has 8 Data Types

String

Number

Bigint

Boolean

Undefined

Null

Symbol

Object

Primitive Values and Non-Primitive Values

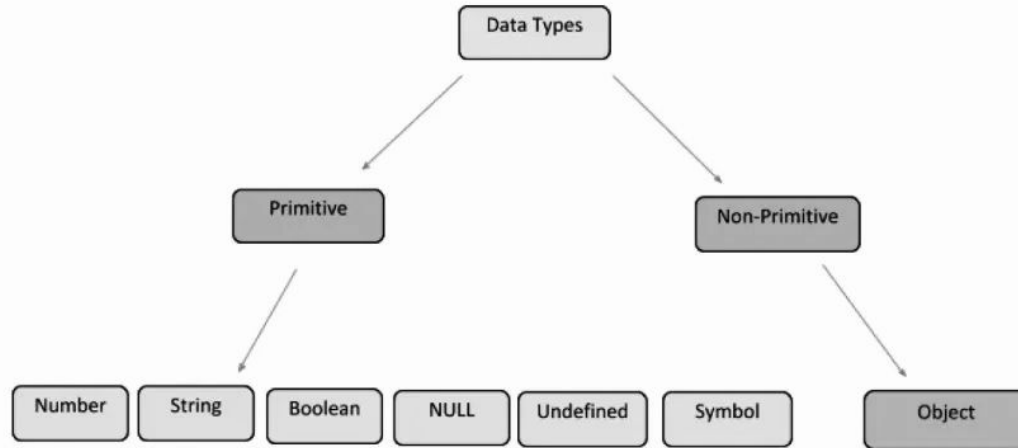
In JavaScript, values are divided into two categories: **primitive values** and **non-primitive values**.

Primitive Values

Primitive values are the most basic data types in JavaScript. They are **immutable**, meaning their values cannot be changed. When you manipulate a **primitive value**, you are working directly on the value itself.

TIMES INTERNET

JAVASCRIPT DATA TYPES



JavaScript Data Types

Primitive Values

Number :- let a = 42 // typeof(a) number

String :- let b = "Rohit Azad" // typeof(b) string

Boolean :- let c = true // typeof(c) boolean (Represents a logical entity and can have two values: **true** or **false**.)

Null :- let d = null // typeof(d) (Represents the intentional absence of any object value. It is a special keyword in JavaScript.)

Undefined :- let e; // typeof(e)

Symbol (introduced in ES6) :- let sym = Symbol("unique"); (Represents a unique and immutable identifier.)

TIMES INTERNET

JavaScript Data Types

Non Primitive Values

Non-primitive values, also known as reference types, include objects, arrays, and functions. They are mutable and can hold multiple values. When you manipulate a non-primitive value, you are working with a reference to the value, not the value itself.

Object :- Represents collections of key-value pairs.

```
let person = {  
  name: "John",  
  age: 30  
};
```

Array :- Represents an ordered list of values.

```
let numbers = [1, 2, 3, 4, 5];
```

Function :- Represents a block of code designed to perform a particular task

Date :- Represents a date and time.

```
function greet() {  
  console.log("Hello!");  
}
```

RegExp :- Represents a regular expression used for pattern matching.

TIMES INTERNET

JavaScript Data Types

Key Differences

Immutability vs. Mutability

Primitive Values: Immutable. Operations on primitive values result in new values.

```
javascript Copy code  
  
let str1 = "Hello";  
let str2 = str1.toUpperCase(); // str2 is "HELLO", str1 remains "Hello"
```

Non-Primitive Values: Mutable. Operations on non-primitive values can change the original value.

```
javascript Copy code  
  
let obj = { name: "Alice" };  
obj.name = "Bob"; // obj is now { name: "Bob" }
```

Key Differences

Storage and Assignment

Primitive Values: Stored directly in the variable.

```
javascript Copy code  
  
let a = 10;  
let b = a; // b is a copy of a  
a = 20; // a is now 20, b is still 10
```

Non-Primitive Values: Stored as references. Assignments copy the reference, not the value.

```
javascript Copy code  
  
let arr1 = [1, 2, 3];  
let arr2 = arr1; // arr2 is a reference to arr1  
arr1.push(4); // arr1 and arr2 both become [1, 2, 3, 4]
```

TIMES INTERNET

JavaScript Data Types

Key Differences

Comparison

Primitive Values: Compared by value.


javascript

 Copy code

```
let x = 5;  
let y = 5;  
console.log(x === y); // true
```

Non-Primitive Values: Compared by reference.

javascript

 Copy code

```
let obj1 = { name: "Alice" };  
let obj2 = { name: "Alice" };  
console.log(obj1 === obj2); // false, different references  
  
let obj3 = obj1;  
console.log(obj1 === obj3); // true, same reference
```

JavaScript Object

What is an Object?

In JavaScript, an object is a collection of key-value pairs where each key is a string (also called a property name), and each value can be any data type, including another object. Objects are used to store related data and functionalities together, making them a fundamental part of JavaScript programming.

**JAVASCRIPT
OBJECTS }**

JS

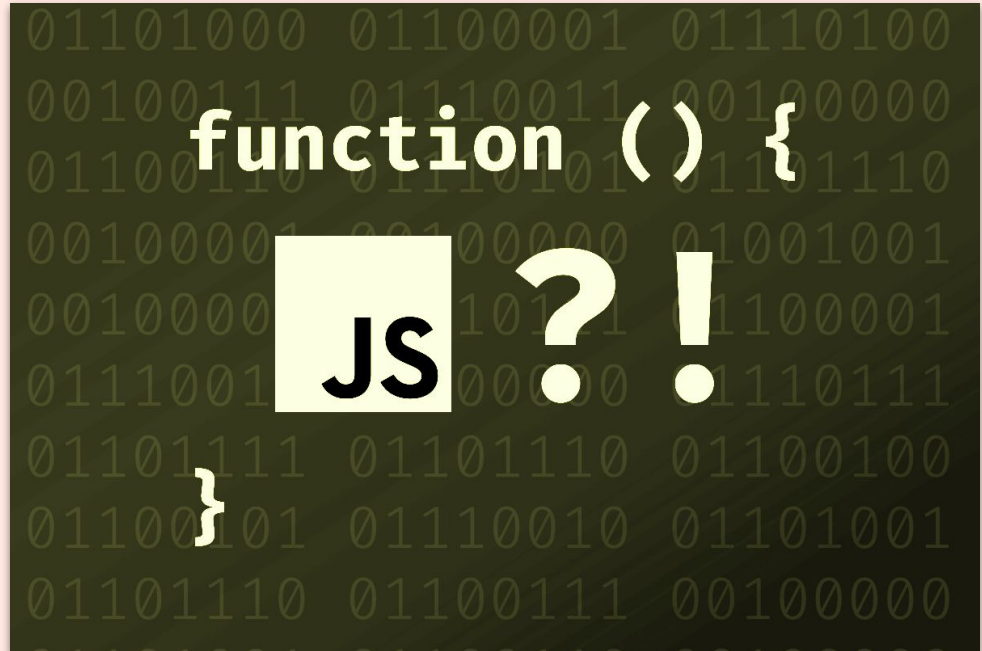
TIMES INTERNET

JavaScript Function

What is a Function in JavaScript?

A function in JavaScript is a block of code designed to perform a particular task. It is executed when "called" (or "invoked").

Functions allow you to reuse code: define the code once and use it many times. Functions can take inputs, process them, and return an output.



JavaScript Destructuring

Destructuring is a very useful way for accessing multiple properties within an array or an object in Javascript.

Learning this technique is going to be essential if you are planning to learn React in the future. Even if you do not plan to learn React, this technique is also a great way to make your code cleaner and easier to work with.

What is Destructuring?

When you destructure an array or object you are converting it to a smaller array, object, or variable.

Destructuring in
JavaScript
[...]

JSON - JavaScript Object Notation.

It is a lightweight data interchange format that is easy for humans to read and write, and easy for machines to parse and generate.

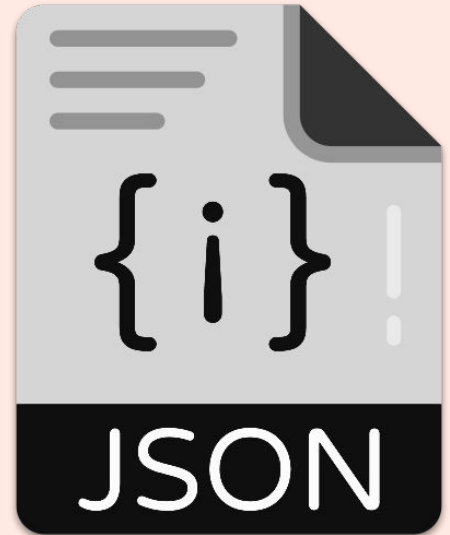
Why Use JSON?

JSON is used to exchange data between a server and a web application.

It is language-independent.

Easy to read and write.

Widely used in APIs and web services.



TIMES INTERNET

JavaScript Promises

A Promise is an object representing the eventual completion or failure of an asynchronous operation. It allows you to associate handlers with an asynchronous action's eventual success value or failure reason.

Syntax:

```
let promise = new Promise(function(resolve, reject) {  
  // executor (the producing code)  
});
```

Example:

```
let myPromise = new Promise((resolve, reject) => {  
  let condition = true;  
  if (condition) {  
    resolve("Promise resolved successfully!");  
  } else {  
    reject("Promise rejected.");  
  }  
});
```

TIMES INTERNET

Connect with Us

"If you would like to connect with me or access more coding resources, you can find me on **Facebook** and **LinkedIn** or check out my **YouTube channel 'Learn Coding with Bhai'** for free coding tutorials and projects."

Email ID :- rohit.azad@timesinternet.in

Contact No :- **+91-9953878727**

Facebook :- <https://www.facebook.com/IamRohitAzad>

Linkedin :- <https://www.linkedin.com/in/rohitazad/>

Github :- <https://github.com/rohitazad>

Youtube :- <https://www.youtube.com/@azadMalikRohit>