|  |  |
| --- | --- |
| **JavaScript**  ***Brendan Eich*** | JavaScript (JS) is a high-level programming language primarily used for developing interactive web pages. JavaScript is mainly used for client-side web development. It runs in web browsers and allows dynamic manipulation of web page content, handling user interactions, and performing various tasks on the client side.  Suriya Stephen  *Prepared By*  **PARANOIA TECHNOLOGIES** |

**JavaScript**

**What is JavaScript?**

* JavaScript was designed to add interactivity to HTML pages
* JavaScript is a scripting language
* A scripting language is a lightweight programming language
* JavaScript is usually embedded directly into HTML pages
* JavaScript is an interpreted language (means that scripts execute without preliminary compilation)
* Everyone can use JavaScript without purchasing a license

It was originally developed by Brendan Eich at Netscape Communications and first released in 1995.

# What can a JavaScript Do?

* JavaScript gives HTML designers a programming tool
* JavaScript can put dynamic text into an HTML page
* JavaScript can react to events
* JavaScript can read and write HTML elements
* JavaScript can be used to validate data
* JavaScript can be used to detect the visitor's browser
* JavaScript can be used to create cookies

**What is Script tag?**

The <script> tag is an HTML element used to embed or reference JavaScript code within an HTML document. It allows you to include external JavaScript files or write inline JavaScript directly in the HTML file.

Here are a few ways to use the <script> tag:

1. **Inline JavaScript:**

**<script>**

**// Inline JavaScript code**

**alert("Hello, world!");**

**</script>**

In this example, the JavaScript code is placed directly within the **<script>** tags. It will be executed when the browser encounters it while parsing the HTML file.

1. **External JavaScript file:**

**<script src="script.js"></script>**

By specifying the **src** attribute, you can include an external JavaScript file (**script.js** in this example). The browser will fetch and execute the JavaScript code from the specified file.

1. **Deferred execution:**

**<script src="script.js" defer></script>**

Adding the **defer** attribute to the **<script>** tag tells the browser to defer executing the script until the HTML document has finished parsing. This can improve page loading performance, especially when the script is not required to modify the initial HTML structure.

1. **Asynchronous loading:**

**<script src="script.js" async></script>**

The **async** attribute allows the browser to load the script asynchronously while continuing to parse the HTML document. This can be beneficial for non-blocking script loading, but note that the order of execution may not be preserved if multiple scripts are loaded asynchronously.

**Linking JS file to html file**

To link a JavaScript file to an HTML file, you can use the <script> tag with the src attribute. Here's how you can do it:

1. **Create a JavaScript file:** Start by creating a separate JavaScript file with a .js extension. For example, script.js.
2. **Place the JavaScript file in your project directory:** Put the JavaScript file in the same directory as your HTML file, or you can specify a relative path to the file if it's located in a different directory.
3. **Link the JavaScript file in the HTML file:** Open your HTML file in a text editor or an HTML editor of your choice. Inside the <head> or just before the closing </body> tag, add the <script> tag with the src attribute pointing to your JavaScript file.

<html>

<head>

**<!-- Other HTML code -->**

<script src="script.js"></script>

</head>

<body>

**<!-- HTML body content -->**

**<!-- JavaScript code can be placed here as well -->**

</body>

</html>

In the above example, the JavaScript file **script.js** is linked to the HTML file. The browser will load and execute the JavaScript code when it encounters the **<script>** tag.

1. **Save and run your HTML file:** Save the changes in your HTML file and open it in a web browser. The browser will fetch the JavaScript file specified in the **<script>** tag and execute its contents.

**Variables**

* In JavaScript, variables are used to store and manipulate data.
* They act as containers that hold values of various types, such as numbers, strings, booleans, objects, or functions.
* Variables in JavaScript are declared using the **var**, **let**, or **const** keywords.

Here's an overview of each type of variable declaration:

1. **var**:

* The **var** keyword is the oldest way to declare variables in JavaScript.
* It has function scope or global scope, meaning the variable is accessible within the function or globally throughout the code.
* Variables declared with **var** can be redeclared and reassigned.

**Example:**

**var age = 25**; // Variable declaration and initialization

**var name;** // Variable declaration (without initialization)

**name = "John";** // Variable assignment

1. **let:**

* The let keyword was introduced in ECMAScript 6 (ES6).
* It has block scope, meaning the variable is accessible within the nearest enclosing block (e.g., if statement, loop, or function).
* Variables declared with let can be reassigned but not redeclared within the same block.

**Example:**

**let age = 25;** // Variable declaration and initialization

**let name;** // Variable declaration (without initialization)

**name = "John";** // Variable assignment

1. **const:**

* The const keyword also came with ES6.
* It is used to declare variables with block scope that cannot be reassigned or redeclared.
* const does not make the value itself immutable; it only prevents the variable from being reassigned.

**Example:**

**const age = 25;** // Variable declaration and initialization

**const name= "John";** // Variable declaration and assignment

# Datatypes

* **Numbers -** are values that can be processed and calculated. You don't enclose them in quotation marks. The numbers can be either positive or negative.
* **Strings -** are a series of letters and numbers enclosed in quotation marks. JavaScript uses the string literally; it doesn't process it. You'll use strings for text you want displayed or values you want passed along.
* **Boolean** (**true**/**false**) - lets you evaluate whether a condition meets or does not meet specified criteria.
* **Null** - is an empty value. **null** is not the same as 0 **--** 0 is a real, calculable number, whereas **null** is the **absence of any value**.
* **Undefined**: Represents an uninitialized variable or a variable with no assigned value.

**Arrays**

An array is an ordered collection of values. It is a versatile data structure that allows you to store multiple values of different types within a single variable. Arrays in JavaScript can dynamically grow or shrink in size as needed.

1. **Creating an Array**: You can create an array using the array literal syntax, which uses square brackets (**[]**), or by using the **new** keyword with the **Array()** constructor.

**Example:**

**const fruits = ["apple", "banana", "orange"];**

**const numbers = new Array(1, 2, 3, 4, 5);**

1. **Accessing Array Elements**: Array elements are accessed using their index, starting from 0. You can use square brackets along with the index to access or modify individual elements.

**Example:**

**const fruits = ["apple", "banana", "orange"];**

**console.log(fruits[0]);** // Outputs "apple"

**fruits[1] = "grape";** // Modifies the second element

1. **Array Length:** The **length** property of an array returns the number of elements in the array. You can use it to get the current length or modify the length to add or remove elements.

**Example:**

**const fruits = ["apple", "banana", "orange"];**

**console.log(fruits.length);** // Outputs 3

**fruits.length = 2;** // Truncates the array to 2 elements

1. **Array Methods**: JavaScript provides a variety of built-in methods to manipulate arrays. Some commonly used methods include:
   * + push(): Adds one or more elements to the end of an array.
     + pop(): Removes the last element from an array and returns it.
     + shift(): Removes the first element from an array and returns it.
     + unshift(): Adds one or more elements to the beginning of an array.
     + slice(): Returns a new array with a portion of the original array.
     + splice(): Modifies an array by adding, removing, or replacing elements.

**Example:**

**const fruits = ["apple", "banana"];**

**fruits.push("orange");** // Adds "orange" to the end

**const removed = fruits.pop();** // Removes and returns "orange"

**fruits.unshift("grape");** // Adds "grape" to the beginning

**const sliced = fruits.slice(1);** // Returns a new array ["banana"]

**fruits.splice(1, 0, "kiwi");**  // Inserts "kiwi" at index 1

1. **Iterating Over Arrays**: You can iterate over the elements of an array using loops such as **for**, **for...of**, or array methods like **forEach ()**.

**Example:**

**const fruits = ["apple", "banana", "orange"];**

**for (let i = 0; i < fruits.length; i++) {**

**console.log(fruits[i]);**

**}**

**// Using forEach method**

**fruits.forEach(function (fruit) {**

**console.log(fruit);**

**});**

**Objects**

* Objects are composite data types that allow you to store collections of key-value pairs.
* Objects are used extensively in JavaScript for representing complex data structures and modelling real-world entities or concepts.

**Overview of working with objects in JavaScript:**

1. **Object Literal Syntax:** The simplest way to create an object is by using the object literal syntax, which involves enclosing key-value pairs within curly braces **{}**.

**Example:**

**let person = {**

**name: "John",**

**age: 30,**

**email: "john@example.com"**

**};**

1. **Accessing Object Properties:** You can access properties of an object using dot notation (**objectName.propertyName**) or square bracket notation (**objectName['propertyName']**).

**Example:**

**console.log(person.name);** // Output: John

**console.log(person['age']);** // Output: 30

1. **Modifying Object Properties:** You can modify the values of object properties by assigning new values to them.

**Example:**

**person.age = 35;**

**person['email'] = "john.doe@example.com";**

1. **Adding and Removing Properties:** You can add new properties to an existing object by assigning values to them. Similarly, you can remove properties using the **delete** keyword.

**Example:**

**person.address = "Paradise Street, Pebbles Avenue";**

**delete person.email;**

1. **Object Methods:** Objects can also contain functions, which are known as methods. These methods can be invoked using dot notation and can operate on the object's properties or perform other tasks.

**Example:**

**let person = {**

**name: "John",**

**age: 30,**

**greet: function() {**

**console.log("Hello, my name is " + this.name);**

**}**

**};**

**person.greet(); // Output: Hello, my name is John**

**Array of Objects**

* In JavaScript, an array of objects is a common data structure that allows you to store multiple objects within a single array.
* Each element of the array is an object with its own set of properties and values.

**Example:**

**Creating an array of objects**

**let students = [**

**{**

**name: "John",**

**age: 20,**

**grade: "A"**

**},**

**{**

**name: "Jane",**

**age: 22,**

**grade: "B"**

**},**

**{**

**name: "Mike",**

**age: 21,**

**grade: "A+"**

**}**

**];**

**Accessing object properties in the array**

**console.log(students[0].name); // Output: John**

**console.log(students[1].age);** // Output: 22

**Note:** The above Array of Object Format is similar to JSON (JavaScript Object Notation). This Format is used for sending data from client to server and vice versa.

The method to implement this is: **console.log(JSON.stringify(students));**

**Loops**

In JavaScript, there are several types of loops that allow you to repeatedly execute a block of code. Here are the main types of loops in JavaScript:

1. **for loop:**

* A for loop is commonly used when you know the number of iterations in advance.
* It consists of an initialization, a condition, an iteration statement, and a code block.

**Example:**

**for (let i = 0; i < 5; i++) {**

**console.log(i); }**

1. **while loop:** A while loop continues to execute a block of code as long as a specified condition is true.

**Example:**

**let i = 0;**

**while (i < 5) {**

**console.log(i);**

**i++;**

**}**

1. **do-while loop**:

* A **do-while** loop is similar to a **while** loop, but it ensures that the code block is executed at least once before checking the condition.

**Example:**

**let i = 0;**

**do {**

**console.log(i);**

**i++;**

**} while (i < 5);**

**Conditions**

In JavaScript, conditions are used to make decisions and control the flow of code execution based on certain criteria.

**Conditional Statements are:**

**if statement**:

* The **if** statement allows you to execute a block of code if a specified condition is true.
* It can be followed by an optional **else if** statement and an optional **else** statement.

**Example:**

**let num = 10;**

**if (num > 0) {**

**console.log("The number is positive.");**

**} else if (num < 0) {**

**console.log("The number is negative.");**

**} else {**

**console.log("The number is zero.");**

**}**

**switch statement**: The **switch** statement provides a way to perform different actions based on different values of a variable or expression.

**Example:**

**let day = 3;**

**let dayName;**

**switch (day) {**

**case 1:**

**dayName = "Monday";**

**break;**

**case 2:**

**dayName = "Tuesday";**

**break;**

**case 3:**

**dayName = "Wednesday";**

**break;**

**default:**

**dayName = "Unknown";**

**break;**

**}**

**console.log("Today is " + dayName);**

**ternary operator:**

* The ternary operator (condition? expression1 : expression2) is a shorthand way to write an if-else statement.
* It evaluates a condition and returns the value of expression1 if the condition is true, or the value of expression2 if the condition is false.

**Example:**

let num = 10;

let result = (num > 0) ? "Positive" : "Negative or zero";

console.log(result);

**logical operators**: JavaScript provides logical operators (**&&**, **||**, **!**) to combine or negate conditions.

* **&&** (AND): Returns **true** if both operands are **true**.
* **||** (OR): Returns **true** if either operand is **true**.
* **!** (NOT): Negates the result of a condition.

**Example:**

**let hour = 10;**

**let isWeekend = true;**

**if (hour >= 9 && hour <= 17 && !isWeekend)**

**{**

**console.log("Office hours");**

**} else {**

**console.log("Non-office hours");**

**}**

**Functions**

* Functions are reusable blocks of code that can be invoked or called to perform a specific task.
* Functions are a fundamental building block of JavaScript applications and allow you to organize and modularize your code.

**Creating and using functions in JavaScript:**

**Function Declaration**: The basic syntax for declaring a function is as follows:

function functionName(parameter1, parameter2) {

// Function body - code to be executed

// You can use the parameters inside the function

// Optionally, a return statement can be used to specify the function's return value

}

**Example:**

**function addNumbers(a, b) {**

**let sum = a + b;**

**return sum;**

**}**

**Function Expression**: Functions can also be assigned to variables. This is known as a function expression.

**Example:**

**let multiplyNumbers = function(a, b) {**

**let product = a \* b;**

**return product;**

**};**

**Function Invocation:** Once a function is declared, it can be invoked or called to execute its code.

**Example:**

**let result = addNumbers(5, 3);** // Function invocation

**console.log(result);** // Output: 8