UNIT 4

JAVASCRIPT

4.1 Introduction to JavaScript

What is JavaScript?

JavaScript is a lightweight, cross-platform, single-threaded, and interpreted compiled programming language. It is also known as the scripting language for webpages. It is well-known for the development of web pages, and many non-browser environments also use it.

JavaScript is a weakly typed language (dynamically typed). JavaScript can be used for Client-side developments as well as Server-side developments. JavaScript is both an imperative and declarative type of language. JavaScript contains a standard library of objects, like Array, Date, and Math, and a core set of language elements like operators, control structures, and statements.

- Client-side: It supplies objects to control a browser and its Document Object Model (DOM). Like if client-side extensions allow an application to place elements on an HTML form and respond to user events such as **mouse clicks**, **form input**, and **page navigation**. Useful libraries for the client side are AngularJS, ReactJS, VueJS, and so many others.
- Server-side: It supplies objects relevant to running JavaScript on a server. For if the server-side extensions allow an application to communicate with a database, and provide continuity of information from one invocation to another of the application, or perform file manipulations on a server. The useful framework which is the most famous these days is node.js.

How to Link JavaScript File in HTML?

JavaScript can be added to HTML file in two ways:

- Internal JS**:** We can add JavaScript directly to our HTML file by writing the code inside the <script> tag. The <script> tag can either be placed inside the <head> or the <body> tag according to the requirement.
- External JS**:** We can write JavaScript code in another files having an extension.js and then link this file inside the <head> tag of the HTML file in which we want to add this code.

Syntax:

```
<script>
  // JavaScript Code
</script>
```

Example:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <title>
        Basic Example to Describe JavaScript
    </title>
</head>

<body>

    <!-- JavaScript code can be embedded inside
        head section or body section -->
    <script>
        console.log("Welcome to GeeksforGeeks");
    </script>
</body>

</html>
```

Output: Welcome to GeeksforGeeks

History of JavaScript

It was created by **Brendan Eich** in 1995. He was an engineer at Netscape. It was originally going to be named LiveScript but was renamed. Unlike most programming languages, JavaScript language has no concept of input or output. It is designed to run as a scripting language in a host environment, and it is up to the host environment to provide mechanisms for communicating with the outside world. The most common host environment is the browser.

Features of JavaScript

According to a recent survey conducted by **Stack Overflow**, JavaScript is the most popular language on earth. With advances in browser technology and JavaScript having moved into the server with Node.js and other frameworks, JavaScript is capable of so much more. Here are a few things that we can do with JavaScript:

- JavaScript was created in the first place for DOM manipulation. Earlier websites were mostly static, after JS was created dynamic Web sites were made.
- Functions in JS are objects. They may have properties and methods just like other objects. They can be passed as arguments in other functions.
- Can handle date and time.
- Performs Form Validation although the forms are created using HTML.
- No compiler is needed.

Applications of JavaScript

- Web Development: Adding interactivity and behavior to static sites JavaScript was invented to do this in 1995. By using AngularJS that can be achieved so easily.
- Web Applications**:** With technology, browsers have improved to the extent that a language was required to create robust web applications. When we explore a map in Google Maps then we only need to click and drag the mouse. All detailed view is just a click away, and this is possible only because of JavaScript. It uses Application Programming Interfaces(APIs) that provide extra power to the code. The Electron and React are helpful in this department.
- Server Applications: With the help of Node.js, JavaScript made its way from client to server and Node.js is the most powerful on the server side.
- Games: Not only in websites, but JavaScript also helps in creating games for leisure. The combination of JavaScript and HTML 5 makes JavaScript popular in game development as well. It provides the EaseJS library which provides solutions for working with rich graphics.
- Smartwatches: JavaScript is being used in all possible devices and applications. It provides a library PebbleJS which is used in smartwatch applications. This framework works for applications that require the Internet for their functioning.
- Art: Artists and designers can create whatever they want using JavaScript to draw on HTML 5 canvas, and make the sound more effective also can be used **p5.js** library.
- Machine Learning: This JavaScript ml5.js library can be used in web development by using machine learning.
- Mobile Applications: JavaScript can also be used to build an application for non-web contexts. The features and uses of JavaScript make it a powerful tool for

creating mobile applications. This is a Framework for building web and mobile apps using JavaScript. Using React Native, we can build mobile applications for different operating systems. We do not require to write code for different systems. Write once use it anywhere!

Why JavaScript is known as a lightweight programming language?

JavaScript is considered lightweight due to the fact that it has low CPU usage, is easy to implement, and has a minimalist syntax. Minimalist syntax as in, has no data types. Everything is treated here as an object. It is very easy to learn because of its syntax similar to C++ and Java.

A lightweight language does not consume much of your CPU's resources. It doesn't put excess strain on your CPU or RAM. JavaScript runs in the browser even though it has complex paradigms and logic which means it uses fewer resources than other languages. *For example*, NodeJs, a variation of JavaScript not only performs faster computations but also uses fewer resources than its counterparts such as Dart or Java.

4.2 Adding JavaScript to a Webpage

There are following three main ways to add JavaScript to a web pages:

1. Inline JavaScript

You can add JavaScript directly within an HTML element using the onclick, onchange, or other event attributes.

Example:

```
<!DOCTYPE html>
<html>
<head><title>Inline JavaScript Example</title></head>
<body>
    <h2>Click the button:</h2>
    <button onclick="alert('Hello, World!')">Click me</button>
</body>
</html>
```

***Pros**: Simple for small scripts or testing code. **Cons**: Can make HTML harder to read and manage, especially for larger scripts.*

2. Internal JavaScript

You can place JavaScript within the <script> tags inside the <head> or <body> section of your HTML file.

Example :

```
<!DOCTYPE html>
<html>
<head>
    <title>Inline JavaScript Example</title>
    <script>
        function showMessage() { alert("Hello from Internal JavaScript!"); }
    </script>
</head>
<body>
    <h2>Click the button:</h2>
    <button onclick="showMessage()">Click me</button>
</body>
</html>
```

***Pros**: Keeps the JavaScript in one place within the HTML file. **Cons**: Still not ideal for large scripts, as it can make the HTML file longer and harder to read.*

3. External JavaScript

The best practice is often to keep JavaScript in an external file. This keeps the HTML and JavaScript separate, making the code easier to manage, especially as it grows.

1. **Create an external JavaScript file**, typically with the .js extension, such as script.js.

   Example:

```
// script.js
function showMessage() {
  alert("Hello from External JavaScript!");
}
```

2. **Link the external JavaScript file** in your HTML file using a <script> tag with the src attribute.

   Example:

```
<!DOCTYPE html>
<html>
<head>
        <title>Inline JavaScript Example</title>
        <script src="script.js"></script>
</head>
<body>
        <h2>Click the button:</h2>
        <button onclick="showMessage()">Click me</button>
</body>
</html>
```

Pros: Keeps HTML and JavaScript separate, ideal for reusability, and makes code easier to read and maintain. **Cons**: Requires managing separate files, but this is usually beneficial in the long run.

Placement of <script> Tag: <head> or <body>?

- Placing the <script> tag at the **bottom of the <body>** section ensures that the HTML content loads first, which is generally better for performance.
- If the <script> tag is placed in the <head>, it might delay the loading of the page content. To avoid this, you can use the defer attribute, which tells the browser to load the JavaScript only after the HTML document is completely parsed:

Example: `<script src="script.js" defer></script>`

4.3 JavaScript Syntax

JavaScript syntax is the set of rules, how JavaScript programs are constructed:

```
// How to create variables:
    var x;
    let y;

// How to use variables:
   x = 5;
   y = 6;
   let z = x + y;
```

JavaScript Variables

In a programming language, variables are used to store data values.

JavaScript uses the keywords var, let and const to declare variables.

An equal sign is used to assign values to variables.

In this example, x is defined as a variable. Then, x is assigned (given) the value 6:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Variables</h2>
<p>In this example, x is defined as a variable. Then, x is assigned the value of 6:</p>
<p id="demo"></p>
<script>
        let x;
        x = 6;
        document.getElementById("demo").innerHTML = x;
</script>
</body>
</html>
```

Output: 6

4.4 JavaScript Comments

JavaScript comments can be used to explain JavaScript code, and to make it more readable.

JavaScript comments can also be used to prevent execution, when testing alternative code.

Single Line Comments

Single line comments start with //.

Any text between // and the end of the line will be ignored by JavaScript (will not be executed).

This example uses a single-line comment before each code line:

```
<!DOCTYPE html>
<html>
<body>
    <h1 id="myH"></h1>
    <p id="myP"></p>
    <script>
    // Change heading:
    document.getElementById("myH").innerHTML = "JavaScript Comments";
    // Change paragraph:
    document.getElementById("myP").innerHTML = "My first paragraph.";
    </script>
</body>
</html>
```

Output:

JavaScript Comments

My first paragraph.

Multi-line Comments

Multi-line comments start with /* and end with */.

Any text between /* and */ will be ignored by JavaScript.

This example uses a multi-line comment (a comment block) to explain the code:

```
<!DOCTYPE html>
<html>
<body>

<h1 id="myH"></h1>
<p id="myP"></p>
<script>
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
*/
document.getElementById("myH").innerHTML = "JavaScript Comments";
document.getElementById("myP").innerHTML = "My first paragraph.";
</script>

</body>
</html>
```

Output:

JavaScript Comments

My first paragraph.

Note:

It is most common to use single line comments.
Block comments are often used for formal documentation.

Using Comments to Prevent Execution

Using comments to prevent execution of code is suitable for code testing.

Adding // in front of a code line changes the code lines from an executable line to a comment.

This example uses // to prevent execution of one of the code lines:

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript Comments</h2>
    <h1 id="myH"></h1>
    <p id="myP"></p>
    <script>
    //document.getElementById("myH").innerHTML = "JavaScript Comments";
    document.getElementById("myP").innerHTML = "My first paragraph.";
    </script>
    <p>The line starting with // is not executed.</p>
</body>
</html>
```

Output:

JavaScript Comments

My first paragraph.

The line starting with // is not executed.

4.5 JavaScript Variables

Variables are Containers for Storing Data

JavaScript Variables can be declared in 4 ways:

- Automatically
- Using var

- Using let
- Using const

In this first example, x, y, and z are undeclared variables.

They are automatically declared when first used:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Variables</h1>
    <p>In this example, x, y, and z are undeclared.</p>
    <p>They are automatically declared when first used.</p>
    <p id="demo"></p>
    <script>
    x = 5;
    y = 6;
    z = x + y;
    document.getElementById("demo").innerHTML = "The value of z is: " + z;
    </script>
</body>
</html>
```

Output:

JavaScript Variables

In this example, x, y, and z are undeclared.

They are automatically declared when first used.

The value of z is: 11

Note:

It is considered good programming practice to always declare variables before use.

From the examples you can guess:

- x stores the value 5
- y stores the value 6
- z stores the value 11

```
<!DOCTYPE html>
<html>
<body>
        <h1>JavaScript Variables</h1>
        <p> In this example, x, y, and z are variables.</p>
        <p id="demo"></p>
        <script>
        var x = 5;
        var y = 6;
        var z = x + y;
        document.getElementById("demo").innerHTML = "The value of z is: " + z;
        </script>
</body>
</html>
```

Output:

JavaScript Variables

In this example, x, y, and z are variables.

The value of z is: 11

When to Use var, let, or const?

1. Always declare variables
2. Always use **const** if the value should not be changed
3. Always use **const** if the type should not be changed (Arrays and Objects)
4. Only use **let** if you can't use const
5. Only use **var** if you MUST support old browsers.

JavaScript Identifiers

All JavaScript **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain **letters**, **digits**, **underscores**, and **dollar signs**.
- Names must begin with a **letter**.
- Names can also begin with **$** and _.
- Names are case sensitive (**y** and **Y** are different variables).
- Reserved words (like JavaScript keywords) cannot be used as names.

Note:

JavaScript identifiers are case-sensitive.

4.6 JavaScript Data Types

JavaScript has 8 Datatypes

- String
- Number
- Bigint
- Boolean

- Undefined
- Null
- Symbol
- Object

The Object Datatype

The object data type can contain both **built-in objects**, and **user defined objects**:

Built-in object types can be:

objects, arrays, dates, maps, sets, intarrays, floatarrays, promises, and more.

Example:

```
// Numbers:
let length = 16;
let weight = 7.5;

// Strings:
let color = "Yellow";
let lastName = "Johnson";

// Booleans
let x = true;
let y = false;

// Object:
const person = {firstName:"John", lastName:"Doe"};

// Array object:
const cars = ["Saab", "Volvo", "BMW"];

// Date object:
const date = new Date("2022-03-25");
```

Note:

A JavaScript variable can hold any type of data.

The Concept of Data Types

In programming, data types is an important concept.

To be able to operate on variables, it is important to know something about the type.

Without data types, a computer cannot safely solve this:

```
let x = 16 + "Volvo";
```

Does it make any sense to add "Volvo" to sixteen? Will it produce an error or will it produce a result?

JavaScript will treat the example above as:

```
let x = "16" + "Volvo";
```

Note:

When adding a number and a string, JavaScript will treat the number as a string.

Example 1:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript</h2>

<p>When adding a number and a string, JavaScript will treat the number as a string.</p>

<p id="demo"></p>

<script>

let x = 16 + "Volvo";

document.getElementById("demo").innerHTML = x;

</script>

</body>
</html>
```

Output:

JavaScript

When adding a number and a string, JavaScript will treat the number as a string.

16Volvo

Example 2:

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript</h2>
    <p> When adding a string and a number, JavaScript will treat the number as a string.</p>
    <p id="demo"></p>
    <script>
    let x = "Volvo" + 16;
    document.getElementById("demo").innerHTML = x;
    </script>
</body>
</html>
```

Output:

JavaScript

When adding a string and a number, JavaScript will treat the number as a string.

Volvo16

JavaScript evaluates expressions from left to right. Different sequences can produce different results:

Example 1:

JavaScript: `let x = 16 + 4 + "Volvo";` **Result:** 20Volvo

Example 2:

JavaScript: let x = "Volvo" + 16 + 4; **Result:** Volvo164

In the first example, JavaScript treats 16 and 4 as numbers, until it reaches "Volvo".

In the second example, since the first operand is a string, all operands are treated as strings.

JavaScript Strings

A string (or a text string) is a series of characters like "John Doe".

Strings are written with quotes. You can use single or double quotes:

Example:

```
<!DOCTYPE html >
<html>
<body>
    <h2>JavaScript Strings</h2>
    <p>Strings are written with quotes. You can use single or double quotes:</p>
    <p id="demo"></p>
    <script>
    let carName1 = "Volvo XC60";
    let carName2 = 'Volvo XC60';
    document.getElementById("demo").innerHTML =carName1 + "<br>" +
    carName2;
    </script>
</body>
</html>
```

Output:

JavaScript Strings

Strings are written with quotes. You can use single or double quotes:

Volvo XC60
Volvo XC60

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

Example:

```
// Single quote inside double quotes:
let answer1 = "It's alright";

// Single quotes inside double quotes:
let answer2 = "He is called 'Johnny'";

// Double quotes inside single quotes:
let answer3 = 'He is called "Johnny"';
```

JavaScript Numbers

All JavaScript numbers are stored as decimal numbers (floating point).

Numbers can be written with, or without decimals:

Example:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>Numbers can be written with, or without decimals:</p>

<p id="demo"></p>

<script>
let x1 = 34.00   let x2 = 34;    let x3 = 3.14;

document.getElementById("demo").innerHTML = x1 + "<br>" + x2 + "<br>" + x3;

</script>

</body>
</html>
```

Output:

JavaScript Numbers

Numbers can be written with, or without decimals:

34
34
3.14

Exponential Notation

Extra large or extra small numbers can be written with scientific (exponential) notation:

Example:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>Extra large or extra small numbers can be written with scientific (exponential) notation:</p>

<p id="demo"></p>

<script>
let y = 123e5;
let z = 123e-5;
document.getElementById("demo").innerHTML = y + "<br>" + z;
</script>

</body>
</html>
```

Output:

JavaScript Numbers

Extra large or extra small numbers can be written with scientific (exponential) notation:

12300000
0.00123

Note:

Most programming languages have many number types:

Whole numbers (integers):
- byte (8-bit), short (16-bit), int (32-bit), long (64-bit)

Real numbers (floating-point):
-float (32-bit), double (64-bit).

Javascript numbers are always one type:
-double (64-bit floating point).

JavaScript BigInt

All JavaScript numbers are stored in a 64-bit floating-point format.

JavaScript BigInt is a new datatype (ES2020) that can be used to store integer values that are too big to be represented by a normal JavaScript Number.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript Numbers</h2>
    <p>A BigInt can not have decimals.</p>
    <p id="demo"></p>
    <p>You cannot perform math between a BigInt type and a Number type.</p>
    <script>
    let x = BigInt("12345678901234567890123456789O");
    document.getElementById("demo").innerHTML = x;
    </script>
</body>
</html>
```

Output:

JavScript Bigint

A BigInt can not have decimals.

12345678901234567890123456789O

You cannot perform math between a BigInt type and a Number type.

JavaScript Booleans

Booleans can only have two values: true or false.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript Numbers</h2>
    <p> Booleans can have two values: true or false:</p>
    <p id="demo"></p>
    <script>
    let x = 5;      let y = 5;      let z = 6;
    document.getElementById("demo").innerHTML = (x == y) + "<br>" + (x == z);
    </script>
</body>
</html>
```

Output:

JavaScript Booleans

Booleans can have two values: true or false:

true
false

- Booleans are often used in conditional testing.

JavaScript Arrays

JavaScript arrays are written with square brackets.

Array items are separated by commas.

The following code declares (creates) an array called cars, containing three items (car names):

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h2> JavaScript Arrays </h2>
    <p> Array indexes are zero-based, which means the first item is [0].</p>
    <p id="demo"></p>
    <script>
        const cars = ["Saab","Volvo","BMW"];
        document.getElementById("demo").innerHTML = cars[0];
    </script>
</body>
</html>
```

Output:

JavaScript Arrays

Array indexes are zero-based, which means the first item is [0].

Saab

- Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

JavaScript Objects

JavaScript objects are written with curly braces {}.

Object properties are written as name:value pairs, separated by commas.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h2> JavaScript Objects </h2>
    <p id="demo"></p>
    <script>
        const person = {
          firstName : "John",
          lastName  : "Doe",
          age     : 50,
          eyeColor  : "blue"
        };
        document.getElementById("demo").innerHTML =
        person.firstName + " is " + person.age + " years old.";
    </script>
</body>
</html>
```

Output:

JavaScript Objects

John is 50 years old.

- The object (person) in the example above has 4 properties: firstName, lastName, age, and eyeColor.

Undefined

In JavaScript, a variable without a value, has the value undefined.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Operators</h1>
    <h2>The typeof Operator</h2>
    <p>The value (and the data type) of a variable with no value
    is<b>undefined</b>.</p>
    <p id="demo"></p>
    <script>
    let car;
    document.getElementById("demo").innerHTML = car + "<br>" + typeof car;
    </script>
</body>
</html>
```

Output:

JavaScript Operators

The typeof Operator

The value of a variable with no value is **undefined**.

Undefined

- Any variable can be emptied, by setting the value to undefined.

Example:

```
<!DOCTYPE html>
<html>
```

```
<body>
    <h1>JavaScript Operators</h1>
    <h2>The typeof Operator</h2>
    <p>Variables can be emptied if you set the value to <b>undefined</b>.</p>
    <p id="demo"></p>
    <script>
    let car = "Volvo";
    car = undefined;
    document.getElementById("demo").innerHTML =
    "<p>The value of car is " + car +
    "<p>The typeof car is " + typeof car;
    </script>
</body>
</html>
```

Output:

JavaScript Operators

The typeof Operator

Variables can be emptied if you set the value to **undefined**.

The value of car is undefined

The typeof car is undefined

The typeof Operator

You can use the JavaScript typeof operator to find the type of a JavaScript variable.

The typeof operator returns the type of a variable or an expression:

Example 1:

```
<!DOCTYPE html>
<html>
<body>
```

```
<h1>JavaScript Operators</h1>
<h2>The typeof Operator</h2>
<p>The typeof operator returns the type of a variable or an expression.</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
typeof " " + "<br>" +
typeof "John" + "<br>" +
typeof "John Doe";
</script>
</body>
</html>
```

Output:

JavaScript Operators

The typeof Operator

The typeof operator returns the type of a variable or an expression.

string
string
string

Example 2:

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Operators</h1>
<h2>The typeof Operator</h2>
<p>The typeof operator returns the type of a variable or an expression.</p>
<p id="demo"></p>
```

```
<script>
document.getElementById("demo").innerHTML =
typeof 0 + "<br>" +
typeof 314 + "<br>" +
typeof 3.14 + "<br>" +
typeof (3) + "<br>" +
typeof (3 + 4);
</script>
</body>
</html>
```

Output:

JavaScript Operators

The typeof Operator

The typeof operator returns the type of a variable or an expression.

number
number
number
number
number

Empty Values

An empty value has nothing to do with undefined.

An empty string has both a legal value and a type.

Example:

```
<!DOCTYPE html>
<html>
<body>
<h2> JavaScript Datatypes</h2>
<p>An empty string has both a legal value and a type:</p>
```

```
        <p id="demo"></p>
        <script>
            let car = "";
            document.getElementById("demo").innerHTML =
            "The value is: " + car + "<br>" + "The type is: " + typeof car;
        </script>
    </body>
    </html>
```

Output:

JavaScript Datatypes

An empty string has both a legal value and a type:

The value is:
The type is: string

4.7 JavaScript Operators

JavaScript operators are used to perform different types of mathematical and logical computations.

Examples:

The **Assignment Operator** = assigns values

The **Addition Operator** + adds values

The **Multiplication Operator** * multiplies values

The **Comparison Operator** > compares values

JavaScript Assignment

The Assignment Operator (=) assigns a value to a variable:

Example 1:

```
let x = 10;
```

Example 2:

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Operators</h1>
<h2>The Assignment (=) Operator</h2>
<p id="demo"></p>
<script>
    // Assign the value 5 to x
    let x = 5;
    // Assign the value 2 to y
    let y = 2;
    // Assign the value x + y to z
    let z = x + y;
    // Display z
    document.getElementById("demo").innerHTML = "The sum of x + y is: " + z;
</script>
</body>
</html>
```

Output:

JavaScript Operators

The Assignment (=) Operator

The sum of x + y is: 7

JavaScript Addition

The **Addition Operator** (**+**) adds numbers:

Example:

```
<!DOCTYPE html>
```

```
<html>
<body>
    <h1>JavaScript Arithmetic</h1>
    <h2>The + Operator</h2>
    <p id="demo"></p>
    <script>
        let x = 5;
        let y = 2;
        let z = x + y;
        document.getElementById("demo").innerHTML = z;
    </script>
</body>
</html>
```

Output:

JavaScript Arithmetic

The + Operator

7

JavaScript Multiplication

The **Multiplication Operator** (*) multiplies numbers:

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Arithmetic</h1>
    <h2>The * Operator</h2>
    <p id="demo"></p>
```

```
        <script>
            let x = 5;
            let y = 2;
            let z = x * y;
            document.getElementById("demo").innerHTML = z;
        </script>
    </body>
    </html>
```

Output:

JavaScript Arithmetic

The * Operator

10

Types of JavaScript Operators

There are different types of JavaScript operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- String Operators
- Logical Operators
- Bitwise Operators
- Ternary Operators
- Type Operators

4.8 JavaScript Arithmetic Operators

Arithmetic operators perform arithmetic on numbers (literals or variables).

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Remainder)
++	Increment
--	Decrement

Arithmetic Operations

A typical arithmetic operation operates on two numbers.

The two numbers can be literals:

Example 1:

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Arithmetic</h1>

<p>A typical arithmetic operation takes two numbers and produces a new number.</p>

<p id="demo"></p>

<script>
```

```
        let x = 100 + 50;
        document.getElementById("demo").innerHTML = x;
    </script>
</body>
</html>
```

Output:

JavaScript Arithmetic

A typical arithmetic operation takes two numbers and produces a new number.

150

or variables:

Example 2:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Arithmetic</h1>
    <p>A typical arithmetic operation takes two numbers (or variables) and produces a new number.</p>
    <p id="demo"></p>
    <script>
        let a = 100;
        let b = 50;
        let x = a + b;
        document.getElementById("demo").innerHTML = x;
    </script>
</body>
</html>
```

Output:

JavaScript Arithmetic

A typical arithmetic operation takes two numbers (or variables) and produces a new number.

150

or expressions:

Example 3:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Arithmetic</h1>
    <p>A typical arithmetic operation takes two numbers (or expressions) and produces a new number.</p>
    <p id="demo"></p>
    <script>
        let a = 3;
        let x = (100 + 50) * a;
        document.getElementById("demo").innerHTML = x;
    </script>
</body>
</html>
```

Output:

JavaScript Arithmetic

A typical arithmetic operation takes two numbers (or expressions) and produces a new number.

450

Operators and Operands

The numbers (in an arithmetic operation) are called **operands**.

The operation (to be performed between the two operands) is defined by an **operator**.

Operand	Operator	Operand
100	+	50

Adding

The **addition** operator (+) adds numbers:

Example:

```
let x = 5;
let y = 2;
let z = x + y;
```

Subtracting

The **subtraction** operator (-) subtracts numbers.

Example:

```
let x = 5;
let y = 2;
let z = x - y;
```

Multiplying

The **multiplication** operator (*) multiplies numbers.

Example:

```
let x = 5;
let y = 2;
let z = x * y;
```

Dividing

The **division** operator (/) divides numbers.

Example:

```
let x = 5;
let y = 2;
let z = x / y;
```

Remainder

The **modulus** operator (%) returns the division remainder.

Example:

```
let x = 5;
let y = 2;
let z = x % y;
```

*In arithmetic, the division of two integers produces a **quotient** and a **remainder**.*

*In mathematics, the result of a **modulo operation** is the **remainder** of an arithmetic division.*

Incrementing

The **increment** operator (++) increments numbers.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Arithmetic</h1>
    <p>The ++ Operator</p>
    <p id="demo"></p>
    <script>
        let x = 5;
        x++;
```

```
        let z = x;
        document.getElementById("demo").innerHTML = z;
    </script>
</body>
</html>
```

Output:

JavaScript Arithmetic

The ++ Operator

6

Decrementing

The **decrement** operator (--) decrements numbers.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Arithmetic</h1>
    <p>The -- Operator</p>
    <p id="demo"></p>
    <script>
        let x = 5;
        x--;
        let z = x;
        document.getElementById("demo").innerHTML = z;
    </script>
</body>
</html>
```

Output:

JavaScript Arithmetic

The -- Operator

4

Exponentiation

The **exponentiation** operator (**) raises the first operand to the power of the second operand.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Arithmetic</h1>
    <p>The ** Operator</p>
    <p id="demo"></p>
    <script>
        let x = 5;
        document.getElementById("demo").innerHTML = x ** 2;
    </script>
</body>
</html>
```

Output:

JavaScript Arithmetic

The ** Operator

25

*x ** y produces the same result as* ***Math.pow(x,y)****:*

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Arithmetic</h1>
    <p> Math.pow()</p>
    <p id="demo"></p>
    <script>
        let x = 5;
        document.getElementById("demo").innerHTML = Math.pow(x,2);
    </script>
</body>
</html>
```

Output:

JavaScript Arithmetic

Math.pow()

25

Operator Precedence

Operator precedence describes the order in which operations are performed in an arithmetic expression.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Arithmetic</h1>
    <h2> Operator Precedence</h2>
    <p> Multiplication has precedence over addition.</p>
    <p id="demo"></p>
```

```
        <script>
            document.getElementById("demo").innerHTML = 100 + 50 * 3;
        </script>
    </body>
    </html>
```

Output:

JavaScript Arithmetic

Operator Precedence

Multiplication has precedence over addition.

250

*Is the result of example above the same as **150 * 3**, or is it the same as **100 + 150**?*

Is the addition or the multiplication done first?

*As in traditional school mathematics, the **multiplication** is **done first**.*

***Multiplication (*)** and **division (/)** have higher **precedence** than **addition (+)** and **subtraction (-)**.*

*And (as in school mathematics) the **precedence** can be changed by **using parentheses**.*

*When using parentheses, the **operations inside the parentheses** are **computed first**:*

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Arithmetic</h1>
    <h2>Operator Precedence</h2>
    <p>Multiplication has precedence over addition.</p>
    <p>But parenthesis has precedence over multiplication.</p>

    <p id="demo"></p>
```

```
        <script>
            document.getElementById("demo").innerHTML = (100 + 50) * 3;
        </script>
    </body>
    </html>
```

Output:

JavaScript Arithmetic

Operator Precedence

Multiplication has precedence over addition.

But parenthesis has precedence over multiplication.

450

*When many operations have **the same precedence** (like **addition** and **subtraction** or **multiplication** and **division**), they are **computed** from **left to right**:*

Example 1:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Arithmetic</h1>
    <h2>Operator Precedence</h2>
    <p> When many operations has the same precedence, they are computed from left to right.</p>
    <p id="demo"></p>
    <script>
        document.getElementById("demo").innerHTML = 100 + 50 - 3;
    </script>
</body>
</html>
```

Output:

JavaScript Arithmetic

Operator Precedence

When many operations has the same precedence, they are computed from left to right.

147

OR

Example 2:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Arithmetic</h1>
    <h2>Operator Precedence</h2>
    <p> When many operations has the same precedence, they are computed from left to right.</p>
    <p id="demo"></p>
    <script>
        document.getElementById("demo").innerHTML = 100 / 50  * 3;
    </script>
</body>
</html>
```

Output:

JavaScript Arithmetic

Operator Precedence

When many operations has the same precedence, they are computed from left to right.

6

4.9 JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

Shift Assignment Operators

Operator	Example	Same As
<<=	x <<= y	x = x << y
>>=	x >>= y	x = x >> y
>>>=	x >>>= y	x = x >>> y

Bitwise Assignment Operators

Operator	Example	Same As
&=	x &= y	x = x & y
^=	x ^= y	x = x ^ y
|=	x |= y	x = x | y

Logical Assignment Operators

Operator	Example	Same As
&&=	x &&= y	x = x && (x = y)
||=	x ||= y	x = x || (x = y)
??=	x ??= y	x = x ?? (x = y)

Note:

The Logical assignment operators are ES2020.

The = Operator

The **Simple Assignment Operator** assigns a value to a variable.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Assignments</h1>
```

```
        <h2>Simple Assignment</h2>
        <h3>The = Operator</h3>
        <p id="demo"></p>
        <script>
            let x = 10;
            document.getElementById("demo").innerHTML = "Value of x is: " + x;
        </script>
    </body>
    </html>
```

Output:

JavaScript Assignments

Simple Assignment

The = Operator

Value of x is: 10

The += Operator

The **Addition Assignment Operator** adds a value to a variable.

Example:

```
    <!DOCTYPE html>
    <html>
    <body>
        <h1>JavaScript Assignments</h1>
        <h2> Addition Assignment</h2>
        <h3>The += Operator</h3>
        <p id="demo"></p>
        <script>
            let x = 10;
            x += 5;
```

```
        document.getElementById("demo").innerHTML = "Value of x is: " + x;
    </script>
</body>
</html>
```

Output:

JavaScript Assignments

Addition Assignment

The += Operator

Value of x is: 15

The -= Operator

The **Subtraction Assignment Operator** subtracts a value from a variable.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Assignments</h1>
    <h2> Subtraction Assignment</h2>
    <h3>The -= Operator</h3>
    <p id="demo"></p>
    <script>
        let x = 10;
        x -= 5;
        document.getElementById("demo").innerHTML = "Value of x is: " + x;
    </script>
</body>
</html>
```

Output:

JavaScript Assignments

Subtraction Assignment

The -= Operator

Value of x is: 5

The *= Operator

The **Multiplication Assignment Operator** multiplies a variable.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Assignments</h1>
    <h2>Multiplication Assignment</h2>
    <h3>The *= Operator</h3>
    <p id="demo"></p>
    <script>
        let x = 10;
        x *= 5;
        document.getElementById("demo").innerHTML = "Value of x is: " + x;
    </script>
</body>
</html>
```

Output:

JavaScript Assignments

Multiplication Assignment

The *= Operator

Value of x is: 50

The **= Operator

The **Exponentiation Assignment Operator** raises a variable to the power of the operand.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Assignments</h1>
    <h2> Exponentiation Assignment</h2>
    <h3>The **= Operator</h3>
    <p id="demo"></p>
    <script>
        let x = 10;
        x **= 5;
        document.getElementById("demo").innerHTML = "Value of x is: " + x;
    </script>
</body>
</html>
```

Output:

JavaScript Assignments

Exponentiation Assignment

The **= Operator

Value of x is: 100000

The /= Operator

The **Division Assignment Operator** divides a variable.

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Assignments</h1>
    <h2> Division Assignment</h2>
    <h3>The /= Operator</h3>
    <p id="demo"></p>
    <script>
        let x = 10;
        x /= 5;
        document.getElementById("demo").innerHTML = x;
    </script>
</body>
</html>
```

Output:

JavaScript Assignments

Division Assignment

The /= Operator

2

The %= Operator

The **Remainder Assignment Operator** assigns a remainder to a variable.

Example:

```
<!DOCTYPE html>
<html>
<body>
```

```
        <h1>JavaScript Assignments</h1>
        <h2> Remainder Assignment</h2>
        <h3> The %= Operator</h3>
        <p id="demo"></p>
        <script>
            let x = 10;
            x %= 5;
            document.getElementById("demo").innerHTML = "Value of x is: " + x;
        </script>
    </body>
    </html>
```

Output:

JavaScript Assignments

Remainder Assignment

The %= Operator

Value of x is: 0

The <<= Operator

The **Left Shift Assignment Operator** left shifts a variable.

Example:

```
    <!DOCTYPE html>
    <html>
    <body>
        <h1>JavaScript Assignments</h1>
        <h2> Left Shift Assignment</h2>
        <h3> The <<= Operator</h3>
        <p id="demo"></p>
        <script>
```

```
            let x = -100;
            x <<= 5;
            document.getElementById("demo").innerHTML = "Value of x is: " + x;
        </script>
    </body>
    </html>
```

Output:

JavaScript Assignments

Left Shift Assignment

The <<= Operator

Value of x is: -3200

The >>= Operator

The **Right Shift Assignment Operator** right shifts a variable (signed).

Example:

```
    <!DOCTYPE html>
    <html>
    <body>
        <h1>JavaScript Assignments</h1>
        <h2> Right Shift Assignment</h2>
        <h3> The >>= Operator </h3>
        <p id="demo"></p>
        <script>
            let x = -100;
            x >>= 5;
            document.getElementById("demo").innerHTML = "Value of x is: " + x;
        </script>
    </body>
```

```
</html>
```

Output:

JavaScript Assignments

Right Shift Assignment

The >>= Operator

Value of x is: -4

The >>>= Operator

The **Unsigned Right Shift Assignment Operator** right shifts a variable (unsigned).

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Assignments</h1>
    <h2> Right Shift Assignment</h2>
    <h3> The >>>= Operator </h3>
    <p id="demo"></p>
    <script>
        let x = -100;
        x >>>= 5;
        document.getElementById("demo").innerHTML = "Value of x is: " + x;
    </script>
</body>
</html>
```

Output:

JavaScript Assignments

Right Shift Assignment

The >>>= Operator

Value of x is: 134217724

The &= Operator

The **Bitwise AND Assignment Operator** does a bitwise AND operation on two operands and assigns the result to the the variable.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Assignments</h1>
    <h2> Bitwise AND Assignment</h2>
    <h3> The &= Operator</h3>
    <p id="demo"></p>
    <script>
        let x = 100;
        x &= 5;
        document.getElementById("demo").innerHTML = "Value of x is: " + x;
    </script>
</body>
</html>
```

Output:

JavaScript Assignments

Bitwise AND Assignment

The &= Operator

Value of x is: 4

The |= Operator

The **Bitwise OR Assignment Operator** does a bitwise OR operation on two operands and assigns the result to the variable.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Assignments</h1>
    <h2> Bitwise OR Assignment</h2>
    <h3> The |= Operator</h3>
    <p id="demo"></p>
    <script>
        let x = 100;
        x |= 5;
        document.getElementById("demo").innerHTML = "Value of x is: " + x;
    </script>
</body>
</html>
```

Output:

JavaScript Assignments

Bitwise OR Assignment

The |= Operator

Value of x is: 101

The ^= Operator

The **Bitwise XOR Assignment Operator** does a bitwise XOR operation on two operands and assigns the result to the variable.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Assignments</h1>
    <h2> Bitwise XOR Assignment</h2>
    <h3> The ^= Operator</h3>
    <p id="demo"></p>
    <script>
        let x = 100;
        x ^= 5;
        document.getElementById("demo").innerHTML = "Value of x is: " + x;
    </script>
</body>
</html>
```

Output:

JavaScript Assignments

Bitwise XOR Assignment

The ^= Operator

Value of x is: 97

The &&= Operator

The **Logical AND assignment operator** is used between two values.

If the first value is true, the second value is assigned.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Assignments</h1>
```

```
<h2>Logical AND Assignment</h2>
<h3> The &&= Operator</h3>
<p> If the first value is true, the second value is assigned.</p>
<p id="demo"></p>
<script>
    let x = 100;
    x &&= 5;
    document.getElementById("demo").innerHTML = "Value of x is: " + x;
</script>
</body>
</html>
```

Output:

JavaScript Assignments

Logical AND Assignment

The &&= Operator

If the first value is true, the second value is assigned.

Value of x is: 5

Note:

The &&= operator is an ES2020 feature.

The ||= Operator

The **Logical OR assignment operator** is used between two values.

If the first value is false, the second value is assigned.

Example:

```
<!DOCTYPE html>
<html>
<body>
```

```
        <h1>JavaScript Assignments</h1>
        <h2> Logical OR Assignment</h2>
        <h3> The ||= Operator</h3>
        <p> If the first value is false, the second value is assigned:</p>
        <p id="demo"></p>
        <script>
            let x = undefined;
            x ||= 5;
            document.getElementById("demo").innerHTML = "Value of x is: " + x;
        </script>
    </body>
    </html>
```

Output:

JavaScript Assignments

Logical OR Assignment

The ||= Operator

If the first value is false, the second value is assigned:

Value of x is: 5

Note:

The ||= operator is an ES2020 feature.

The ??= Operator

The **Nullish coalescing assignment operator** is used between two values.

If the first value is undefined or null, the second value is assigned.

Example:

```
<!DOCTYPE html>
<html>
```

```
<body>
    <h1>JavaScript Assignments</h1>
    <h2> The ??= Operator</h2>
    <p> The ??= operator is used between two values. If the first value is undefined or null, the second value is assigned.</p>
    <p id="demo"></p>
    <script>
        let x;
        document.getElementById("demo").innerHTML = x ??= 5;
    </script>
</body>
</html>
```

Output:

JavaScript Assignments

The ??= Operator

The ??= operator is used between two values. If the first value is undefined or null, the second value is assigned.

5

Note:

The ??= operator is an ES2020 feature.

4.10 JavaScript Comparison and Logical Operators

Comparison and Logical operators are used to test for true or false.

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that x = 5, the table below explains the comparison operators:

Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
		x == "5"	true
===	equal value and equal type	x === 5	true
		x === "5"	false
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== 5	false
		x !== "5"	true
		x !== 8	true
>	greater than	x > 8	false
<	less than	x < 8	true
>=	greater than or equal to	x >= 8	false
<=	less than or equal to	x <= 8	true

How Can it be Used

Comparison operators can be used in conditional statements to compare values and take action depending on the result:

Example:

```
if (age < 18) text = "Too young to buy alcohol";
```

Logical Operators

Logical operators are used to determine the logic between variables or values.

Given that x = 6 and y = 3, the table below explains the logical operators:

Operator	Description	Example
&&	and	(x < 10 && y > 1) is true
||	or	(x == 5 || y == 5) is false
!	not	!(x == y) is true

Conditional (Ternary) Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax

variablename = (*condition*) ? *value1*:*value2*

Example:

```
let voteable = (age < 18) ? "Too young":"Old enough";
```

If the variable age is a value below 18, the value of the variable voteable will be "Too young", otherwise the value of voteable will be "Old enough".

Comparing Different Types

Comparing data of different types may give unexpected results.

When comparing a string with a number, JavaScript will convert the string to a number when doing the comparison. An empty string converts to 0. A non-numeric string converts to NaN which is always false.

Case	Value
2 < 12	true
2 < "12"	true
2 < "John"	false

2 > "John"	false
2 == "John"	false
"2" < "12"	false
"2" > "12"	true
"2" == "12"	false

When comparing two strings, "2" will be greater than "12", because (alphabetically) 1 is less than 2.

To secure a proper result, variables should be converted to the proper type before comparison:

Example:

```
age = Number(age);
if (isNaN(age)) {
  voteable = "Input is not a number";
} else {
  voteable = (age < 18) ? "Too young" : "Old enough";
}
```

The Nullish Coalescing Operator (??)

The ?? operator returns the first argument if it is not **nullish** (null or undefined).

Otherwise it returns the second argument.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Operators</h1>
    <h2>The ?? Operator</h2>
```

```
<p>The ?? operator returns the first argument if it is not nullish (null or undefined). Otherwise it returns the second.</p>

<p id="demo"></p>

<script>

let name = null;

let text = "missing";

let result = name ?? text;

document.getElementById("demo").innerHTML = "The name is " + result;

</script>

</body>

</html>
```

Output:

JavaScript Operators

The ?? Operator

The ?? operator returns the first argument if it is not nullish (null or undefined). Otherwise it returns the second.

The name is missing

The Optional Chaining Operator (?.)

The ?. operator returns undefined if an object is undefined or null (instead of throwing an error).

Example:

```
<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Operators</h1>

<h2>The ?. Operator</h2>

<p>The ?. operator returns undefined if an object is undefined or null (instead of throwing an error).</p>
```

```
        <p>Car name is:</p>
        <p id="demo"></p>
        <script>
            const car = {type:"Fiat", model:"500", color:"white"};
            let name = car?.name;
            document.getElementById("demo").innerHTML = name;
        </script>
    </body>
    </html>
```

Output:

JavaScript Operators

The ?. Operator

The ?. operator returns undefined if an object is undefined or null (instead of throwing an error).

Car name is:

undefine

4.11 JavaScript String Operators

JavaScript String Operators are used to manipulate and perform operations on strings. There are two operators which are used to modify strings in JavaScript. These operators help us to join one string to another string.

Type of JavaScript String Operators

There are two type of String Operators in JavaScript, these are:

- String Concatenate Operator
- String Concatenate Assignment Operator

String Concatenate Operator

Concatenate Operator in JavaScript combines strings using the '+' operator and creates a new string that includes the contents of the original strings in which Concatenate string1 and string2, ensuring the first character of string2 immediately follows the last character of string1.

Syntax:

```
str1 + str2
```

In this example, Concatenating str1 and str2 using the '+' operator, the result variable holds the string "GeeksforGeeks".

Example:

```
let str1 = "Geeks";

let str2 = "forGeeks";

let result = (str1 + str2);

console.log(result);
```

Output:

```
GeeksforGeeks
```