UNIT 2 –WORKING WITH JAVASCRIPT

1. Embed JavaScript into HTML

1.1 What is JavaScript?

- JavaScript is a **client-side scripting** language that adds interactivity to our websites/web apps. JavaScript allows us to create and update the contents of our web apps dynamically. It is one of the most used programming languages with HTML and CSS. Today HTML, CSS, and JavaScript are considered the three base pillars of all modern websites.
- JavaScript was initially created by Brendan Eich to add **life to web pages**. Yes, you read it right! You can do almost anything and everything crazy with JavaScript. You can make servers, web apps, mobile apps, etc using JavaScript. But what makes JavaScript different from other programming languages is that it can be used both on the client-side as well as server-side and is supported by all major browsers.
- Now that you know how powerful JavaScript is you might be wondering how to add JavaScript to HTML. Don't worry! We have got you covered. This article explains all the ways to link JavaScript with HTML in detail.

1.2 Linking JavaScript with HTML

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

1. **Embedding code:** Adding the JavaScript code between a pair of **<script>...</script>** tag
2. **Inline code:** Placing JavaScript code directly inside HTML tags using some special attributes.
3. **External file:** Making a separate JavaScript file having **.js** as an extension.

1.2.1 Embedding code

- You can directly embed your JavaScript code in your HTML pages by directly wrapping it inside a pair of <script> and </script> tag.
- The <script> tag is used to enclose client-side scripting languages
- A single HTML document can have multiple pairs of <script>....</script> tags with each pair having an infinite number of JavaScript statements.
- All these script tags are executed in the sequential order in which they are written.
- Like other HTML tags, <script> tag also has various attributes. Some of the most commonly used attributes of the <script> tag are as follows:
- **src**
    - This attribute is used to specify the location of the external script file used. The value of the attribute should always be wrapped inside double ("..") or single ('..') quotes

```
<script src = "./app.js">
     Your JavaScript code.......
</script>
```

- **defer**
  - This attribute is used to improve the performance and loading time of the websites by delaying the execution of code inside the <script>...</script> tag till the time the content is displayed on the browser.
  - In simple words, it allows downloading the script file in parallel to the parsing of the page but starts the execution only once the parsing of the page is completely finished.
  - Note: defer attribute should only be used when the src attribute is present (i.e. only with external JavaScript files).

```
<script src="./app.js" defer></script>
```

- **async**
  - **Important Note:** If **async** or **defer** attributes are not present then the parsing of the page is blocked and the script is downloaded and executed immediately as soon as the compiler encounters a **<script>** tag.
- **type**
  - This attribute is used to specify the type of script and to identify the content of the tag. Some commonly used values by this attribute are:
    1. **text/javascript (default value)**
    2. **text/ecmascript**
    3. **application/ecmascript**
    4. **application/javascript**

If this attribute is not mentioned then by default the script is treated as JavaScript.

```
<script type="text/javascript">
          Your JavaScript code....
</script>
```

✓ **The script in the head section**:

```
<!DOCTYPE html>
<html>
<head>
<title>InfoGalaxy</title>
  <script>
    alert("Hey there! Greetings from Codedamn");
  </script>
</head>
<body>
  <p>
  Demonstrating the use of SCRIPT tag within the HEAD section.
  </p>
</body>
</html>
```

✓ **The script in the body section:**

```
<!DOCTYPE html>
<html>
<head>
    <title>InfoGalaxy</title>
</head>
<body>
  <p>
  Demonstrating the use of SCRIPT tag within the BODY tag.
  </p>
  <script>
    alert("Hey there! Greetings from Codedamn");
  </script>
</body>
</html>
```

Although both the above-mentioned method works it is still preferred to use the <script> tag just before the **closing of the body tag**. The reason behind this will be explored in the **later section** of this article.

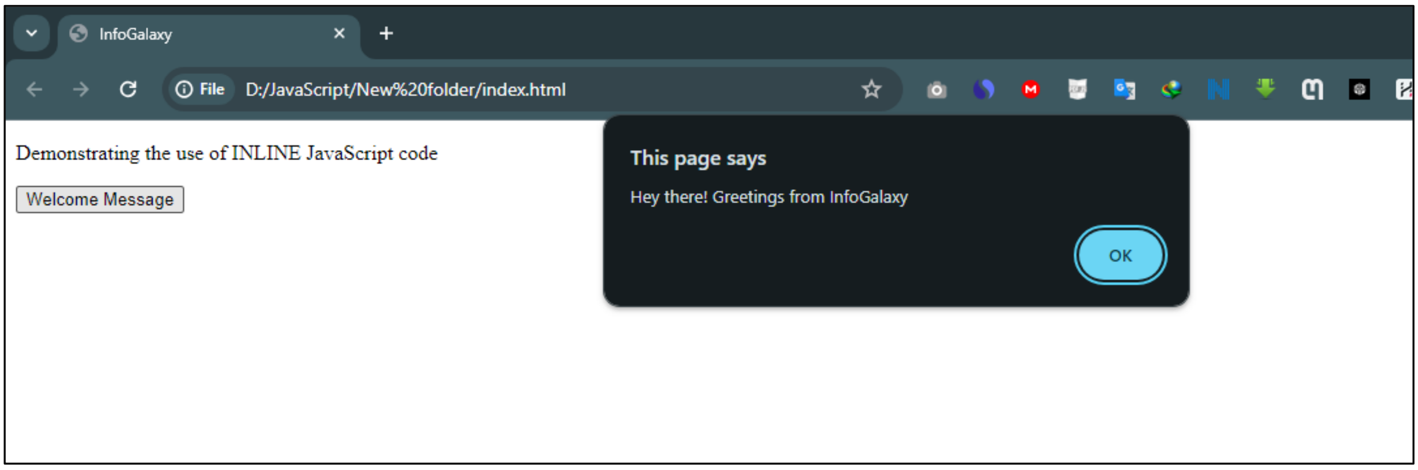
1.2.2 Inline code

- Inline code as the word suggests means writing the code in the line itself. In simple language, placing JavaScript code directly inside the **HTML tags** by using some special attributes like **onload**, **onclick**, etc is called inline JavaScript code.

✓ **Inline JavaScript code example:**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>InfoGalaxy</title>
</head>
<body>
    <p>Demonstrating the use of INLINE JavaScript code</p>
    <button onclick="alert('Hey there! Greetings from
InfoGalaxy')">Welcome Message</button>
</body>
</html>
```

✓ **Inline JavaScript code Output:**


InfoGalaxy

D:/JavaScript/New%20folder/index.html

Demonstrating the use of INLINE JavaScript code

Welcome Message

This page says

Hey there! Greetings from InfoGalaxy

OK

1.2.3 External file

- Creating a **separate JavaScript file** and calling it in your HTML file by using the **src** attribute of the **script** tag. The JavaScript file must be saved by the **.js** extension.
- This method allows us to use the same script file in **multiple** HTML documents by simply linking it using a single line thus making our code cleaner and easily maintainable

```
<html lang="en">
<head>
    <title>InfoGalaxy</title>
</head>
<body>
    <p>Demonstrating the use of EXTERNAL JavaScript file</p>
    <script src="./app.js"></script>
</body>
</html>
```

```
// app.js file
alert("Hey there! Greetings from InfoGalaxy");
```

2. Data Types

There are different types of data that we can use in a JavaScript program. For example,

```
const x = 5;
const y = "Hello";
```

Here,

- 5 is an integer data.
- "Hello" is a string data.

2.1 JavaScript Data Types

There are eight basic data types in JavaScript. They are:

Data Types	Description	Example
String	represents textual data	'hello', "hello world!" etc
Number	an integer or a floating-point number	3, 3.234, 3e-2 etc.
BigInt	an integer with arbitrary precision	900719925124740999n , 1n etc.
Boolean	Any of two values: true or false	true and false
undefined	a data type whose variable is not initialized	let a;
null	denotes a null value	let a = null;
Symbol	data type whose instances are unique and immutable	let value = Symbol('hello');
Object	key-value pairs of collection of data	let student = { };

Here, all data types except Object are primitive data types, whereas Object is non-primitive.

Note: The Object data type (non-primitive type) can store collections of data, whereas primitive data type can only store a single data.

2.2 JavaScript String

String is used to store text. In JavaScript, strings are surrounded by quotes:

- Single quotes: 'Hello'
- Double quotes: "Hello"
- Backticks: `Hello`

For example,

```
//strings example
const name = 'ram';
const name1 = "hari";
const result = `The names are ${name} and ${name1}`;
```

Single quotes and double quotes are practically the same and you can use either of them.

Backticks are generally used when you need to include variables or expressions into a string. This is done by wrapping variables or expressions with ${variable or expression} as shown above.

2.3 JavaScript Number

Number represents integer and floating numbers (decimals and exponentials).

For example,

```
const number1 = 3;
const number2 = 3.433;
const number3 = 3e5 // 3 * 10^5
```

A number type can also be +Infinity, -Infinity, and NaN (not a number).

For example,

```
const number1 = 3/0;
console.log(number1); // Infinity
const number2 = -3/0;
console.log(number2); // -Infinity
// strings can't be divided by numbers
const number3 = "abc"/3;
console.log(number3);  // NaN
```

2.4 JavaScript BigInt

In JavaScript, Number type can only represent numbers less than **(2^{53} - 1)** and more than **-(2^{53} - 1)**. However, if you need to use a larger number than that, you can use the BigInt data type.

A **BigInt** number is created by appending **n** to the end of an integer.

For example,

```
// BigInt value
const value1 = 900719925124740998n;
// Adding two big integers
const result1 = value1 + 1n;
console.log(result1); // "900719925124740999n"
const value2 = 900719925124740998n;
// Error! BitInt and number cannot be added
const result2 = value2 + 1;
console.log(result2);
```

Output :-

```
900719925124740999n
Uncaught TypeError: Cannot mix BigInt and other types
```

Note: BigInt was introduced in the newer version of JavaScript and is not supported by many browsers including Safari. Visit JavaScript BigInt support to learn more.

2.4 JavaScript Boolean

This data type represents logical entities. Boolean represents one of two values: true or false. It is easier to think of it as a yes/no switch.

For example,

```
const dataChecked = true;
const valueCounted = false;
```

2.5 JavaScript undefined

The undefined data type represents **value that is not assigned**. If a variable is declared but the value is not assigned, then the value of that variable will be undefined.

For example,

```
let name;
console.log(name); // undefined
```

It is also possible to explicitly assign a variable value undefined.

For example,

```
let name = undefined;
console.log(name); // undefined
```

Note: It is recommended not to explicitly assign undefined to a variable. Usually, null is used to assign 'unknown' or 'empty' value to a variable.

2.6 JavaScript null

In JavaScript, null is a special value that represents **empty** or **unknown value**.

For example,

```
const number = null;
```

The code above suggests that the number variable is empty.

Note: null is not the same as NULL or Null.

2.7 JavaScript Symbol

This data type was introduced in a newer version of JavaScript (from ES2015).

A value having the data type Symbol can be referred to as a **symbol value**. Symbol is an immutable primitive value that is unique.

For example,

```
/ two symbols with the same description
const value1 = Symbol('hello');
const value2 = Symbol('hello');
```

Though value1 and value2 both contain 'hello', they are different as they are of the Symbol type.

2.8 JavaScript Object

An object is a complex data type that allows us to store collections of data.

For example,

```
const student = {
    firstName: 'ram',
    lastName: null,
    class: 10
};
```

2.9 JavaScript Type

JavaScript is a dynamically typed (loosely typed) language. JavaScript automatically determines the variables' data type for you.

It also means that a variable can be of one data type and later it can be changed to another data type.

For example,

```
// data is of undefined type
let data;
// data is of integer type
data = 5;
// data is of string type
data = "JavaScript Programming";
```

2.10 JavaScript typeof

To find the type of a variable, you can use the typeof operator.

For example,

```
const name = 'ram';
typeof(name); // returns "string"
const number = 4;
typeof(number); //returns "number"
const valueChecked = true;
typeof(valueChecked); //returns "boolean"
const a = null;
typeof(a); // returns "object"
```

Notice that typeof returned "object" for the null type. This is a known issue in JavaScript since its first release.

3. Operators & Expressions

✓ What is an Operator?

In JavaScript, an operator is a special symbol used to perform operations on operands (values and variables).

For example,

```
2 + 3; // 5
```

Here + is an operator that performs addition, and 2 and 3 are operands.

✓ JavaScript Operator Types

Here is a list of different operators you will learn in this tutorial.

- Assignment Operators
- Arithmetic Operators
- Comparison Operators
- Logical Operators
- Bitwise Operators
- String Operators
- Other Operators

3.1 JavaScript Assignment Operators

Assignment operators are used to **assign** values to variables.

For example,

```
const x = 5;
```

Here, the = operator is used to assign value 5 to variable x.

Here's a list of commonly used assignment operators:

Operator	Name	Example
=	Assignment operator	a = 7; // 7
+=	Addition assignment	a += 5; // a = a + 5
-=	Subtraction Assignment	a -= 2; // a = a - 2
*=	Multiplication Assignment	a *= 3; // a = a * 3
/=	Division Assignment	a /= 2; // a = a / 2
%=	Remainder Assignment	a %= 2; // a = a % 2
**=	Exponentiation Assignment	a **= 2; // a = a**2

Note: The commonly used assignment operator is =. You will understand other assignment operators such as +=, -=, *= etc. once we learn arithmetic operators.

3.2 JavaScript Arithmetic Operators

Arithmetic operators are used to perform **arithmetic calculations**.

For example,

```
const number = 3 + 5; // 8
```

Here, the + operator is used to add two operands.

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Remainder	x % y
++	Increment (increments by 1)	++x or x++
--	Decrement (decrements by 1)	--x or x--
**	Exponentiation (Power)	x ** y

Example 1: Arithmetic operators in JavaScript

```
let x = 5;
let y = 3;
// addition
console.log('x + y = ', x + y);  // 8
// subtraction
console.log('x - y = ', x - y);  // 2
// multiplication
console.log('x * y = ', x * y);  // 15
// division
console.log('x / y = ', x / y);  // 1.6666666666666667
// remainder
console.log('x % y = ', x % y);   // 2
// increment
console.log('++x = ', ++x); // x is now 6
console.log('x++ = ', x++); // prints 6 and then increased to 7
console.log('x = ', x);     // 7
// decrement
console.log('--x = ', --x); // x is now 6
console.log('x-- = ', x--); // prints 6 and then decreased to 5
console.log('x = ', x);     // 5
//exponentiation
console.log('x ** y =', x ** y);
```

Note: The ** operator was introduced in ECMAScript 2016 and some browsers may not support them.

3.3 JavaScript Comparison Operators

Comparison operators **compare** two values and return a boolean value, either true or false.

For example,

```
const a = 3, b = 2;
console.log(a > b); // true
```

Here, the comparison operator > is used to compare whether a is greater than b.

Operator	Description	Example
==	**Equal to**: returns true if the operands are equal	x == y
!=	**Not equal to**: returns true if the operands are not equal	x != y
===	**Strict equal to**: true if the operands are equal and of the same type	x === y
!==	**Strict not equal to**: true if the operands are equal but of different type or not equal at all	x !== y
>	**Greater than**: true if left operand is greater than the right operand	x > y
>=	**Greater than or equal to**: true if left operand is greater than or equal to the right operand	x >= y
<	**Less than**: true if the left operand is less than the right operand	x < y
<=	**Less than or equal to**: true if the left operand is less than or equal to the right operand	x <= y

Example 2: Comparison operators in JavaScript

```
// equal operator
console.log(2 == 2); // true
console.log(2 == '2'); // true
// not equal operator
console.log(3 != 2); // true
console.log('hello' != 'Hello'); // true
// strict equal operator
console.log(2 === 2); // true
console.log(2 === '2'); // false
// strict not equal operator
console.log(2 !== '2'); // true
console.log(2 !== 2); // false
```

Comparison operators are used in decision-making and loops. You will learn about the use of comparison operators in detail in later tutorials.

3.4 JavaScript Logical Operators

Logical operators perform logical operations and return a boolean value, either true or false. **For example,**

```
const x = 5, y = 3;
(x < 6) && (y < 5); // true
```

Here, && is the logical operator **AND**. Since both x < 6 and y < 5 are true, the result is true.

Operator	Description	Example
&&	**Logical AND**: true if both the operands are true, else returns false	x && y
||	**Logical OR**: true if either of the operands is true; returns false if both are false	x || y
!	**Logical NOT**: true if the operand is false and vice-versa.	!x

Example 3: Logical Operators in JavaScript

```
// logical AND
console.log(true && true); // true
console.log(true && false); // false
// logical OR
console.log(true || false); // true
// logical NOT
console.log(!true); // false
```

Logical operators are used in decision making and loops.

3.5 JavaScript Bitwise Operators

Bitwise operators perform operations on binary representations of numbers.

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<	Left shift
>>	Sign-propagating right shift
>>>	Zero-fill right shift

Bitwise operators are rarely used in everyday programming.

3.6 JavaScript String Operators

In JavaScript, you can also use the + operator to concatenate (join) two or more strings.

Example 4: String operators in JavaScript

```
// concatenation operator
console.log('hello' + 'world');
let a = 'JavaScript';
a += ' tutorial';  // a = a + ' tutorial';
console.log(a);
```

Note: When + is used with strings, it performs concatenation. However, when + is used with numbers, it performs addition.

3.7 Other JavaScript Operators

Here's a list of other operators available in JavaScript. You will learn about these operators in later tutorials.

Operator	Description	Example
,	evaluates multiple operands and returns the value of the last operand.	let a = (1, 3 , 4); // 4
?:	returns value based on the condition	(5 > 3) ? 'success' : 'error'; // "success"
delete	deletes an object's property, or an element of an array	delete x
typeof	returns a string indicating the data type	typeof 3; // "number"
void	discards the expression's return value	void(x)
in	returns true if the specified property is in the object	prop in object
instanceof	returns true if the specified object is of of the specified object type	object instanceof object_type

4. JavaScript Comments

JavaScript comments are hints that a programmer can add to make their code easier to read and understand. They are completely ignored by JavaScript engines.

There are two ways to add comments to code:

- // - Single Line Comments
- /* */ -Multi-line Comments

4.1 Single Line Comments

In JavaScript, any line that starts with // is a single line comment.

For example,

```
name = "Jack";
// printing name on the console
console.log("Hello " + name);
```

Here, // printing name on the console is a comment.

You can also use single line comment like this:

```
name = "Jack";
console.log("Hello " + name);  // printing name on the console
```

4.2 Multi-line Comments

In Javascript, any text between /* and */ is a multi-line comment.

For example,

```
/* The following program contains the source code for a game called
    Baghchal.
Baghchal is a popular board game in Nepal where two players choose
    either sheep or tiger. It is played on a 5x5 grid.
For the player controlling the tiger to win, they must capture all the
    sheep. There are altogether 4 tigers on the board.
For the sheep to win, all tigers must be surrounded and cornered so
    that they cannot move. The player controlling the sheep has 20
    sheep at his disposal.
*/
```

Since the rest of the source code will be used to implement the rules of the game, the comment above is a good example where you might use a multi-line comment.

4.3 Using Comments for Debugging

Comments can also be used to disable code to prevent it from being executed.

For example,

```
console.log("some code");
console.log("Error code);
console.log("other code");
```

If you get an error while running the program, instead of removing the error-prone code, you can use comments to disable it from being executed; this can be a valuable debugging tool.

```
console.log("some code");
// console.log("Error code);
console.log("other code");
```

Pro Tip: Remember the shortcut for using comments; it can be really helpful. For most code editors, it's Ctrl + / for Windows and Cmd + / for Mac.