Module 6) WD - JAVASCRIPT BASIC & DOM

1. **What is JavaScript?**

**Ans.** JavaScript is the Programming Language for the Web.JavaScript can update and change both HTML and CSS.JavaScript can calculate, manipulate and validate data. JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

**2.What is the use of isNaN function?**

**Ans.** The **isNaN() function** is used to check whether a given value is an illegal number or not. It returns true if value is a NaN else returns false. It is different from the Number.isNaN() Method.

Syntax:

isNaN( value )

Parameter Values: This method accepts single parameter as mentioned above and described below: 

value: It is a required value passed in the isNaN() function.

Return Value: It returns a Boolean value i.e. returns true if the value is NaN else returns false.

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Global Methods</h1>

<h2>The isNaN() Method</h2>

<p>isNaN() returns true if a value is NaN:</p>

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

<script>

let result =

"Is 123 NaN? " + isNaN(123) + "<br>" +

"Is -1.23 NaN? " + isNaN(-1.23) + "<br>" +

"Is 5-2 NaN? " + isNaN(5-2) + "<br>" +

"Is 0 NaN? " + isNaN(0);

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

</script>

</body>

</html>

**3.What is negative Infinity?**

**Ans.** The negative infinity in JavaScript is a constant value which is used to represent a value which is the lowest available. This means that no other number is lesser than this value. It can be generated using a self-made function or by an arithmetic operation.

Negative infinity is different from mathematical infinity in the following ways:

Negative infinity results in 0 when divided by any other number.

When divided by itself or positive infinity, negative infinity return NaN

Negative infinity, when divided by any positive number (apart from positive infinity) is negative infinity.

Negative infinity, divided by any negative number (apart from negative infinity) is positive infinity.

If we multiply negative infinity with NaN, we will get NaN as a result.

The product of NaN and negative infinity is 0.

The product of two negative infinities is always a positive infinity.

The product of both positive and negative infinity is always negative infinity.

Syntax.

Example.

<!DOCTYPE html>

<html>

<body>

<style>

h1 {

color: green;

}

</style>

<h1>

What is negative infinity in JavaScript?

</h1>

<button onclick="geekNegativeInfinity()">

Generate negative infinite

</button>

<p id="geek"></p>

<script>

function geekNegativeInfinity() {

//negative value greater than the

//largest representable number in JavaScript

var n = (-Number.MAX\_VALUE) \* 2;

document.getElementById("geek").innerHTML = n;

}

</script>

</body>

</html>

**4.Which company developed JavaScript?**

**Ans**. JavaScript was invented by Brendan Eich in 1995.It was developed for Netscape 2, and became the ECMA-262 standard in 1997.After Netscape handed JavaScript over to ECMA, the Mozilla foundation continued to develop JavaScript for the Firefox browser.

**5.What are undeclared and undefined variables?**

**Ans.**Undeclared − It occurs when a variable which hasn’t been declared using var, let or const is being tried to access.

Undefined − It occurs when a variable has been declared using var, let or const but isn’t given a value.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Document</title>

<style>

body {

font-family: "Segoe UI", Tahoma, Geneva, Verdana, sans-serif;

}

.result {

font-size: 18px;

font-weight: 500;

color: blueviolet;

}

</style>

</head>

<body>

<h1>Undeclared vs Undefined</h1>

<div class="result"></div>

<div class="result"></div>

<button class="Btn">Click here</button>

<h3>Click on the above button to access undeclared and undefined variable

</h3>

<script>

let BtnEle = document.querySelector(".Btn");

let resEle = document.querySelectorAll(".result");

let a;

BtnEle.addEventListener("click", () => {

resEle[0].innerHTML += "Accessing undefined variable = " + a;

try {

resEle[1].innerHTML = b;

} catch (err) {

resEle[1].innerHTML = "Accessing undeclared variable = " + err;

}

});

</script>

</body>

</html>

**6.Write the code for adding new elements dynamically?**

**Ans.** HTML documents can be easily accessed and manipulated using HTML DOM , which represents an HTML document as a tree-structure. When an HTML document is loaded in a browser window, it becomes a Document object. With document.createElement() method you can create a specified HTML element dynamically in JavaScript. After creation you can add attributes. If you want the element to show up in your document, you have to insert in into the DOM-tree of the document.

<html>

<body>

<button onclick="create()">Create Heading</button>

<script>

function create() {

var h1 = document.createElement('h1');

h1.textContent = "New Heading!!!";

h1.setAttribute('class', 'note');

document.body.appendChild(h1);

}

</script>

</body>

</html>

So document.createElement is used with an HTML tag to create the element. The textContent is then modified and then the class attribute is modified using setAttribute . This could also be used to add a data attribute or any other kind of attribute, like you can in HTML. Finally the element is appended to the body using the body element's appendChild method.

Actually, It's essentially equivalent to < H1 class="note" > New Heading!!! < /h1 > .

The power of what we seen today is that we are creating and styling elements "on the fly". It's your decision to either add all the elements on the page while designing or you can create and insert HTML elements at runtime, that is dynamically using the createElement() method.

**7.What is the difference between ViewState and SessionState?**

**Ans.**  The values of controls of a particular page of the client browser is persisted by ViewState at the time of post back operation is done. If the user requests another page, the data of previous page is no longer available.  
  
- The data of a particular server persists in the server by SessionState. The availability of the user data is up to the completion of a session or closure of the browser.

 A ViewState is a state of a page within a browser wherein the values of controls persist when post back operation is done.  
When another page is loaded, the previous page data is no longer available.  
  
- SessionState is the data of a user session and is maintained on the server side. This data available until user closes the browser or session time-outs.

**8. What is === operator?**

**Ans.** The strict equality operator (===) checks whether its two operands are equal, returning a Boolean result. Unlike the [equality](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Equality) operator, the strict equality operator always considers operands of different types to be different.

Example:-

console.log(1 === 1);

// expected output: true

console.log('hello' === 'hello');

// expected output: true

console.log('1' === 1);

// expected output: false

console.log(0 === false);

// expected output: false

> true

> true

> false

> false

The strict equality operators (=== and !==) provide the [IsStrictlyEqual](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness" \l "strict_equality_using) semantic.

If the operands are of different types, return false.

If both operands are objects, return true only if they refer to the same object.

If both operands are null or both operands are undefined, return true.

If either operand is NaN, return false.

Otherwise, compare the two operand's values:

Numbers must have the same numeric values. +0 and -0 are considered to be the same value.

Strings must have the same characters in the same order.

Booleans must be both true or both false.

The most notable difference between this operator and the [equality](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Equality) (==) operator is that if the operands are of different types, the == operator attempts to convert them to the same type before comparing.

**9.How can the style/class of an element be changed?**

**Ans.** Changing CSS with the help of the style property:

Syntax:

document.getElementById("id").style.property = new\_style

Example: In this example, we have built a PAN number validator. First, we will take the input value and match it with a regex pattern. If it matches then using JavaScript add an inline style on the <p> tag. Otherwise, add a different style on the <p> tag.

<!DOCTYPE html>

<html lang="en">

<body>

<h2>

How can the style/class of

an element be changed?

</h2>

<b>Validate Pan Number</b>

<input type="text" id="pan" />

<p></p>

<button id="submit">Validate</button>

<script>

const btn = document.getElementById("submit");

btn.addEventListener("click", function () {

const pan = document.getElementById("pan").value;

const para = document.querySelector("p");

let regex = /([A-Z]){5}([0-9]){4}([A-Z]){1}$/;

if (regex.test(pan.toUpperCase())) {

para.innerHTML = "Hurrey It's correct";

// Inline style

para.style.color = "green";

} else {

para.innerHTML = "OOps It's wrong!";

// Inline style

para.style.color = "red";

}

});

</script>

</body>

</html>

**10.How to read and write a file using JavaScript?**

**Ans.**The read and write operations in a file can be done by using some commands. But the module which is required to perform these operations is to be imported. The required module is ‘fs’ which is called as File System module in JavaScript.

After the File System file is imported then, the writeFile() operation is called. The writeFile() method is used to write into the file in JavaScript. The syntax of this method is as follows −

writeFile(path,inputData,callBackFunction)

The writeFile() function accepts three parameters −

Path − The first parameter is the path of the file or the name of the file into which the input data is to be written.

If there is a file already, then the contents in the file are deleted and the input which is given by the user will get updated or if the file is not present, then the file with that will be created in the given path and the input information is written into it.

inputData − The second parameter is the input data which contains the data to be written in the file that is opened.

callBackFuntion − The third parameter is the function which is the call back function which takes the error as the parameter and shows the fault if the write operation fails.

Following is an example of the write operation in files in JavaScript.

const fs = require('fs')

let fInput = "You are reading"

fs.writeFile('tp.txt', fInput, (err) => {

if (err) throw err;

else{

console.log("The file is updated with the given data")

}

})

Reading from the file

After the File System module is imported, the reading of the file in JavaScript can be done by using the readFile() function.

Syntax

The syntax to read from a file is as follows −

readFile(path, format, callBackFunc)

The readFile() function accepts three parameters including one optional parameter.

Path − The first parameter is the path of the test file from which the contents are to read. If the current location or directory is the same directory where the file which is to be opened and read is located then, only the file name has to be given.

Format − The second parameter is the optional parameter which is the format of the text file. The format can be ASCII, utf-8 etc.

CallBackFunc − The third parameter is the call back function which takes the error as the parameter and displays the fault is any raised due to the error.

Following example tries to read the contents of the file populate in the previous example and print it –

const fs = require('fs')

fs.readFile('tp.txt', (err, inputD) => {

if (err) throw err;

console.log(inputD.toString());

})

**11.What are all the looping structures in JavaScript?**

**Ans. JavaScript Loops**

Loops are handy, if you want to run the same code over and over again, each time with a different value.

Often this is the case when working with arrays:

Instead of writing:

text += cars[0] + "<br>";  
text += cars[1] + "<br>";  
text += cars[2] + "<br>";  
text += cars[3] + "<br>";  
text += cars[4] + "<br>";  
text += cars[5] + "<br>";

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript For Loop</h2>

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

<script>

const cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];

let text = "";

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

text += cars[i] + "<br>";

}

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

</script>

</body>

</html>

Output.

JavaScript For Loop

BMW  
Volvo  
Saab  
Ford  
Fiat  
Audio

**12.How can you convert the string of any base to an integer in JavaScript?**

**Ans.** To **convert**a **string**to an **integer parseInt()**function is used in **javascript**. **parseInt()** function returns **Nan**( not a number) when the string doesn’t contain number. If a string with a number is sent then only that number will be returned as the output. This function won't accept spaces. If any particular number with spaces is sent then the part of the number that presents before space will be returned as the output.

syntax

parseInt(value);

This function takes a **string**and **converts**it into an integer. If there is no integer present in the string, **NaN**will be the output.

Example

In the following example,**various cases of strings** such as **only strings**, **strings with numbers**, etc have been taken and sent through the**parseInt()** function. Later on, their integer values, if present, have displayed as shown in the output.

<html>

<body>

<script>

var a = "10";

var b = parseInt(a);

document.write("value is " + b);

var c = parseInt("423-0-567");

document.write("</br>");

document.write('value is ' + c);

document.write("</br>");

var d = "string";

var e = parseInt(d);

document.write("value is " + e);

document.write("</br>");

var f = parseInt("2string");

document.write("value is " + f);

</script>

</body>

</html>

Output

value is 10

value is 423

value is NaN

value is 2

**13.What is the function of the delete operator?**

**Ans.**

**Delete operator**

The delete**operator** removes a property from an object. If the property's value is an object and there are no more references to the object, the object held by that property is eventually released automatically.

const Employee = {

firstname: 'John',

lastname: 'Doe'

};

console.log(Employee.firstname);

// expected output: "John"

delete Employee.firstname;

console.log(Employee.firstname);

// expected output: undefined

Output.

> "John"

> undefined

[Syntax](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/delete#syntax)

delete object.property

delete object[property]

[Parameters](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/delete#parameters)

object

The name of an object, or an expression evaluating to an object.

property

The property to delete.

[Return value](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/delete#return_value)

true for all cases except when the property is an [own](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/hasOwn) [non-configurable](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/defineProperty#configurable_attribute) property, in which case false is returned in non-strict mode.

[Exceptions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/delete#exceptions)

[TypeError](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/TypeError)

Thrown in [strict mode](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode) if the property is an own non-configurable property.

[Description](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/delete#description)

The delete operator has the same [precedence](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence) as other unary operators like [typeof](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/typeof). Therefore, it accepts any expression formed by higher-precedence operators. However, the following forms lead to early syntax errors in [strict mode](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode):

delete identifier;

delete object.#privateProperty;

Because [classes](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes) are automatically in strict mode, and [private properties](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes/Private_class_fields) can only be legally referenced in class bodies, this means private properties can never be deleted. While delete identifier [may work](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/delete#deleting_global_properties) if identifier refers to a configurable property of the global object, you should avoid this form and prefix it with [globalThis](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/globalThis) instead.

While other expressions are accepted, they don't lead to meaningful behaviors:

delete console.log(1);

// Logs 1, returns true, but nothing deleted

The delete operator removes a given property from an object. On successful deletion, it will return true, else false will be returned. Unlike what common belief suggests (perhaps due to other programming languages like [delete in C++](https://docs.microsoft.com/cpp/cpp/delete-operator-cpp?view=msvc-170)), the delete operator has nothing to do with directly freeing memory. Memory management is done indirectly via breaking references. See the [memory management](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_Management) page for more details.

It is important to consider the following scenarios:

If the property which you are trying to delete does not exist, delete will not have any effect and will return true.

delete only has an effect on own properties. If a property with the same name exists on the object's prototype chain, then after deletion, the object will use the property from the prototype chain.

Non-configurable properties cannot be removed. This includes properties of built-in objects like [Math](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math), [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array), [Object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object) and properties that are created as non-configurable with methods like [Object.defineProperty()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/defineProperty).

Deleting variables, including function parameters, never works. delete variable will throw a [SyntaxError](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SyntaxError) in strict mode, and will have no effect in non-strict mode.

Any variable declared with [var](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var) cannot be deleted from the global scope or from a function's scope, because while they may be attached to the [global object](https://developer.mozilla.org/en-US/docs/Glossary/Global_object), they are not configurable.

Any variable declared with [let](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let) or [const](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const) cannot be deleted from the scope within which they were defined, because they are not attached to an object.

**14.What are all the types of Pop up boxes available in JavaScript?**

**Ans.** Pop up boxes are the dialogue boxes appeared on the screen when clicked on a button. These boxes are used to show some alert messages or instant messages.

In JavaScript, there are three types of pop-ups. Alert box, confirm box and prompt box.

**Alert box**

This pop-up box is used to display some information in a dialogue box when a button is clicked. For example, if a person is using a laptop and the battery percentage gets below 20 percent then a pop-up box is displayed showing a message to connect the charger.

Syntax

This is the syntax for the alert box.

alert(“The message to be displayed”)

Example 1

you execute the following script; it'll open an alert box with the content: "This is an alert" with a confirmation button.

<script>

alert("This is a alert");

</script>

Note that the alert dialog should be used for messages which do not require any response on the part of the user, other than the acknowledgement of the message.

Example 2

Following is another example for this −

<html>

<head>

</head>

<body>

<h1>Pop up Box in JavaScript-Alert box</h1>

<button onclick="jsAlert()"> Click </button>

<script>

function jsAlert(){

alert("The content is published by Tutorials Point")

}

</script>

</body>

</html>

**Confirm Box**

This pop-up box is used to take the opinion of the user on the message displayed. This confirm box has two buttons “OK” and “Cancel”. The user can press either one of the buttons.

Syntax

This is the syntax for the confirm box.

confirm(“The message to be displayed for user confirmation”)

Example 1

If you execute the following script, it'll open a confirmation box with the content: "Please confirm this action" with a confirmation button and cancellation button. This returns a boolean depending on the input provided by the user.

<script>

let bool = confirm("Please confirm this action");

console.log(bool);

</script>

If you click confirm, it'll return true. If you click cancel, it'll return false.

Note that Dialog boxes are modal windows - they prevent the user from accessing the rest of the program's interface until the dialog box is closed.

Example 2

Another example for this is −

<head>

</head>

<body>

<h1>Pop up Box in JavaScript-Confirm box>/h1>

<button onclick="jsConfirm()"> Click >/button>

<p id="a">>/p>

<script>

function jsConfirm(){

var s

if(confirm("This is confirm box in JavaScript"))

s="You pressed OK"

else

s="You pressed Cancel"

document.getElementById("a").innerHTML = s

}

</script>

</body>

</html>

**Prompt Box**

This pop-up box is to get input from the user to give further information as per the specified input. After the input is given, then the user has to press “ok” button son that the information will be displayed in the browser.

Syntax

This is the syntax for the prompt box.

prompt(“enter the message”)

Example 1

If you execute the following script, it'll open a propmt box with the content: "Please enter your name" with a confirmation button and cancellation button. This returns a String provided by the user.

<script>

let name = prompt("Please enter your name");

console.log(name);

</script>

When you give it some input in the prompt, it'll log your name to the console.

Example 2

This is another example demonstrating the usage of prompt box.

<html>

<head>

</head>

<body>

<h1>Pop up Box in JavaScript-Prompt box</h1>

<button onclick="jsPrompt()"> Click </button>

<p id="a"></p>

<script>

function jsPrompt(){

var s=prompt("Tutorials point: Rate on a scale of 1 to 5:")

if(parseInt(s)>=1 && parseInt(s)<=5)

document.getElementById("a").innerHTML = s + " is your rating"

else

document.getElementById("a").innerHTML ="Please provide valid rating"

}

</script>

</body>

</html>

**15.What is the use of Void (0)?**

**Ans.** JavaScript void 0 means returning undefined (void) as a primitive value. You might come across the term “JavaScript:void(0)” while going through [HTML](https://www.simplilearn.com/tutorials/html-tutorial/what-is-html) documents. It is used to prevent any side effects caused while inserting an expression in a web page. For instance, URLs or hyperlinks are the common examples of using JavaScript void 0. Suppose you insert a link and want to call some JavaScript through it. Usually, when you click on a link, the browser will either reload or open a new page. However, if you just want to call JavaScript through that link, you would not want the entire page to refresh. This is where the JavaScript:void(0) will come in handy.

When you use JavaScript void 0, it will return an undefined primitive value. This will prevent the browser from opening a new or reloading the web page and allowing you to call the JavaScript through it.

Preventing a Page to Reload Using JavaScript Void 0

We can use JavaScript void 0 to prevent reloading of a page. Let’s understand this with the two examples given below. The first example does not use JavaScript:void(0), but the second uses it to prevent reloading. We will also use the ondblclick event handler so that nothing happens on a single click and the action is handled on double click.

<!DOCTYPE html>

<html>

<head>

<title>without using JavaScript:void(0)</title>

</head>

<body align="center">

    <h2>This is without using JavaScript:void(0)</h2>

<a href="#" ondblclick="alert('Task completed!')">Double Click Me!</a>

</body>

</html>

JavaScript Void 0 Alternatives

Although JavaScript void 0 is an easy solution to prevent a page from reloading, you might sometimes want to use alternatives. Some of these alternatives include:

You can use event.preventDefault with the event handler as an alternative to JavaScript:void(0). Here’s an example:

<!DOCTYPE html>

<html>

<head>

<title>JavaScript void 0 alternatives</title>

</head>

<body align="center">

    <h2>Using .preventDefault()</h2>

<a href="https://www.simplilearn.com" onclick="event.preventDefault();" ondblclick="alert('Task completed')">Double Click Me!</a>

</body>

</html>

**16.How can a page be forced to load another page in JavaScript?**

**Ans**. We can use [window.location](https://www.geeksforgeeks.org/javascript-window-location-and-document-location-objects/) property inside the *script* tag to forcefully load another page in Javascript. It is a reference to a Location object that is it represents the current location of the document. We can change the URL of a window by accessing it.

Syntax:

<script>

window.location = <Path / URL>

</script>

Example:

<script>

window.location = "https://www.geeksforgeeks.org/"

</script>

So in the above example, we see that by changing the window.location Object inside Javascript we can change the URL of our window and thus successfully load any page forcibly from our Javascript without any href tag. We will build a small working sample to learn it practically.

Below is the step by step implementation:

Step 1: Create a file named *index.html*. Add a heading and two buttons to it. One button forcefully loads a page with a live URL and the other button loads a local HTML page. In the *<script>* tag we have two functions, one loads gfg home page, and the second loads a local HTML page using *window.location* property.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible"

content="IE=edge">

<meta name="viewport" content=

"width=device-width, initial-scale=1.0">

</head>

<body>

<h3>This is the original page</h3>

<br>

<button onclick="force\_load\_gfg()">

Force Load GFG Page

</button>

<br><br>

<button onclick="force\_load\_local()">

Force Load Local HTML page

</button>

<script>

function force\_load\_gfg() {

window.location =

"https://www.geeksforgeeks.org/"

}

function force\_load\_local() {

window.location =

"F:/gfg/PageRedirect/newPage.html"

}

</script>

</body>

</html>

**Step 2:** Create a file named ***newPage.html***. This is the local HTML page that would be loaded by Javascript.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible"

content="IE=edge">

<meta name="viewport" content=

"width=device-width, initial-scale=1.0">

<title> New Page </title>

</head>

<body>

<h3>This is the new loaded page</h3>

</body>

</html>

**17.What are the disadvantages of using innerHTML in JavaScript?**

**Ans.** HTML stands for Hyper Text Markup Language, through the HTML we can design a block of webpages. Html is a frontend markup language that is used to build the content of frontend pages. It means that we can build a structure of web pages.

Through HTML we can design the content of any website. It means that we can create headings, buttons, paragraphs, headers, footers, links, etc for any website.

Example

let’s try to understand to implement a program −

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Basic of HTML</title>

</head>

<body>

<h1>H1 Heading </h1>

<p>This is a paragraph </p>

<a href = "#">Link</a>

<br><br>

<button>Button</button>

<footer>This is a footer</footer>

</body>

</html>

In the above program, we have created a heading, paragraph, link, break tag, button, and footer. As you can see in the above program, we can write an HTML program, in the following ways.

Inner HTML

Inner HTML is just like a property of every HTML element. It means that let’s say we have created a div tag in HTML and inside the div tag, we have created the paragraph tag then we can say that the paragraph tag is an inner HTML and the whole div is an outer HTML.

Also, we can use innerHTML in JavaScript to fetch the property of any tag by using is’t tag name, id, or by class name.

Following is the snippet of code to show an inner HTML −

<div><p>Hello world</p><div>

In the above snippet of code, you can say that the whole div tag is an outer HTML and the whole p tag is an inner HTML.

Example

Let’s try to understand with a suitable example −

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Basic of HTML</title>

</head>

<body>

<div>Hello <strong>HTML</strong></div>

</body>

</html>

In the above program, as you can see initially, we created a div tag and inside the div tag, we created the strong tag. So the div tag will be the outer HTML and whatever is written inside the div tag is the inner HTML.

Disadvantages of innerHTML

Following are the disadvantages of using inner HTML −

Inner HTML is slow

Inner HTML is slow because when we use the inner HTML property in the code it allows us to change using the JavaScript language. It is very slow because as inner HTML already parses the content even we have to parse the content again so that’s why it takes time.

Event handlers attached to any DOM element are preserved

When we have used the event handlers then the event handlers are not automatically attached to the new elements created by innerHTML. To change that, we have to track the event handlers and manually attach them to a new element.

It means that first, we have to fetch the element property through innerHTML, and then we have to attach them to a new element.

Example

Let’s try to understand with an appropriate example −

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Basic of HTML</title>

</head>

<body>

<p id= 'demo'>Hello</p>

<button onclick="Change()">Change</button>

<script>

function Change()

{

let p = document.getElementById('demo');

p.innerHTML = '<span>Hello World</span>';

}

</script>

</body>

</html>

As you can see in the above program first we fetch the element which is already preserved and then we manually attach them to the new element span. you can see the changes in the output screenshots before the event handler and after the event handler.

Some other disadvantages of HTML are −

Replacement is done everywhere

When innerHTML property is used to modify, all the DOM nodes will have to be parsed and created again.

It is not possible to append innerHTML

In JavaScript, ‘+=’ is commonly used for appending. However, when using innerHTML to append to an HTML tag, the entire tag is re-parsed.

Example

<spam id="tp">Tutorials Point</p>

subject = document.getElementById('#tp')

// The whole "tp" tag is reparsed

subject.innerHTML += '<span> Tutorix </span>'

Breaks the document

InnerHTML does not provide proper validation, so any valid HTML code can be used. This has the potential to break the JavaScript document. Even broken HTML can be used, which can cause unexpected issues.

**Used for Cross-site Scripting**

The text and images or elements in the webpage can used by hackers or malicious users to change the text or data and show some different undesired or threatful content by the other HTML element tag. This leads to change of sensitive and confidential information.