**Module (JAVASCRIPT BASIC & DOM) – 4**

1. **What is JavaScript?**

**Ans**.

JavaScript is a versatile language that enables developers to add functionality, manipulate the Document Object Model (DOM), handle events, and create responsive user interfaces within web browsers.

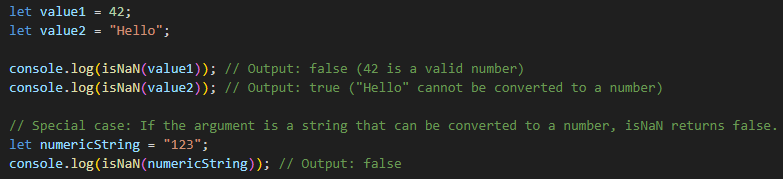
JavaScript is not limited to web browsers; it is also used on the server side (Node.js) and in various other contexts, such as mobile app development and game development.

1. **What is the use of is NaN function?**

**Ans.**

The isNaN function in JavaScript is used to determine whether a value is NaN (Not-a-Number). NaN is a special value in JavaScript that represents the result of an invalid or undefined mathematical operation, such as dividing zero by zero.

**Exm :-**



It's important to note that isNaN performs type coercion, meaning it attempts to convert non-numeric values to numbers before determining if they are NaN. If you want to check for NaN without type coercion, you may use Number.isNaN:



Unlike isNaN, Number. isNaN does not convert non-numeric values to numbers before checking for NaN, making it more strict in its evaluation.

1. **What is negative Infinity?**

**Ans.**

In JavaScript, Negative Infinity is often encountered in certain mathematical operations or as a result of specific computations. For example, dividing a negative number by zero or subtracting Infinity from a finite number may result in Negative Infinity:

let result1 = -5 / 0; // Result is Negative Infinity

let result2 = 10 - Infinity; // Result is Negative Infinity

console.log(result1); // Output: -Infinity

console.log(result2); // Output: -Infinity

It's important to note that Negative Infinity is distinct from the regular JavaScript variable Infinity, which represents positive infinity. Positive Infinity is the result of operations like dividing a positive number by zero or adding Infinity to a finite number. Both Positive Infinity and Negative Infinity serve as placeholders for extremely large or extremely small values that cannot be accurately represented by finite numbers in JavaScript.

1. **Which company developed JavaScript?**

**Ans.**

JavaScript was developed by Netscape Communications Corporation, a company that played a significant role in the early days of the World Wide Web. The language was created by Brendan Eich, and it was initially released in 1995 as part of the Netscape Navigator web browser.

While Netscape was the original developer of JavaScript, the language's standardization process was later taken over by the European Computer Manufacturers Association (ECMA). The standardized version is known as ECMAScript, and subsequent versions of JavaScript are based on this standard.

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

**Ans.**

**Undeclared Variables:**

An undeclared variable is a variable that has been used in code without being declared using the var, let, or const keyword.

If you try to use a variable without declaring it, JavaScript will create a global variable in non-strict mode. In strict mode, referencing an undeclared variable results in an error.

Using undeclared variables is generally discouraged, as it can lead to unexpected behavior and make the code more difficult to understand.

**Undefined Variables:**

An undefined variable is a variable that has been declared but has not been assigned a value, or a variable that is used as an object property, but the property itself is undefined.

When you declare a variable without assigning a value to it, or if you try to access an object property that doesn't exist, the variable is automatically assigned the value undefined.

It's important to note that undefined is a special value in JavaScript, and it represents the absence of a value. When you see undefined, it generally means that the variable has been declared but not yet assigned a value, or you are trying to access a property that does not exist on an object.

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

**Ans.**

In JavaScript, we can dynamically add new elements to the HTML document using the DOM (Document Object Model). Below is an example code snippet that demonstrates how to create a new HTML element (in this case, a paragraph) and add it to the document dynamically:



In this example:

We create a new <p> (paragraph) element using document.createElement("p").

We set the text content of the new paragraph using newParagraph.textContent.

We get a reference to an existing container element where we want to append the new paragraph. In this case, the container has the id="container".

We use container.appendChild(newParagraph) to append the new paragraph as a child of the container.

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

**Ans.**

**ViewState:**

ViewState is specific to ASP.NET web forms and is used to persist state information about an individual web page across postbacks.

It is used to store page-specific information that needs to be retained between round-trips to the server and back to the client.

ViewState is maintained on the client side as a hidden field within the web page. It is encrypted and sent to the client, and the client sends it back to the server on subsequent postbacks.

The information stored in ViewState is only available for the specific page where it is used.

**SessionState:**

SessionState is a broader concept that is not limited to ASP.NET web forms; it is used in various web development frameworks.

SessionState is used to store user-specific information that needs to be maintained across multiple requests and pages during a user's visit to a website.

Unlike ViewState, which is specific to a single page, SessionState is shared across multiple pages in a user's session.

SessionState can be stored on the server or externally, and it is identified by a session ID sent between the server and the client.

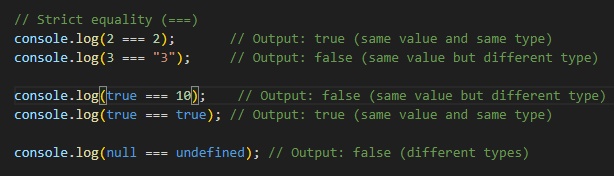
ViewState is page-specific and is used to maintain state information between postbacks for a single page, while SessionState is used to store information across multiple pages during a user's session on a website.

1. **What is === operator?**

**Ans.**

The === operator is a strict equality operator in JavaScript. It is used to compare two values for equality without performing type coercion. Unlike the loose equality operator (==), which allows for type coercion, the strict equality operator checks both the values and their types.

If the types of the two operands are different, the === operator immediately returns false without attempting any type coercion.

If the types are the same, it then compares the values of the operands.

In the first example, 2 === 2 evaluates to true because both the value and the type are the same. In the second example, 3 === "3" evaluates to false because the values are the same, but the types are different.

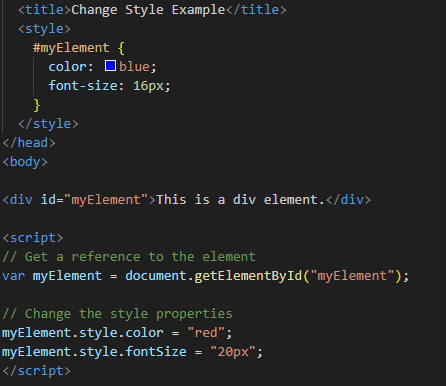
1. **How can the style/class of an element be changed?**

**Ans.**

In JavaScript, We can change the style or class of an HTML element by accessing its properties through the DOM (Document Object Model).

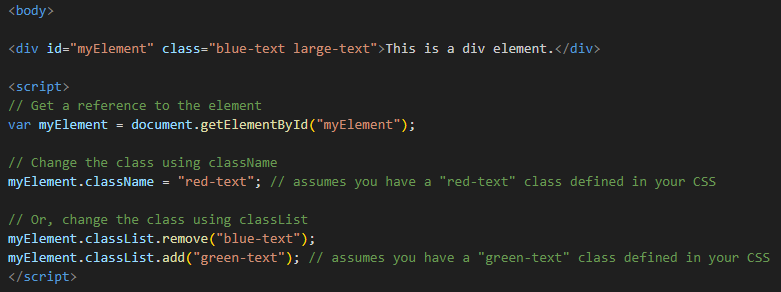
**Changing Style:**

We can directly manipulate the style property of an element to change its CSS styles. Here's an example:



**Changing Class:**

We can also change the class of an element by manipulating its className or using the classList property. Here's an example:



**In this example, the JavaScript code changes the class of the myElement div, removing the "blue-text" class and adding the "green-text" class.**

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

**Ans.**

In a browser environment, JavaScript does not have direct access to read and write files on the user's machine for security reasons. However, JavaScript can interact with files in various ways within certain contexts, such as working with the File API for handling files selected by users through file input fields. Additionally, if you're working with Node.js on the server side, JavaScript has built-in modules to handle file operations.

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

**Ans.**

**for Loop:**

The for loop is one of the most commonly used loops in JavaScript.

It consists of an initialization, a condition, and an iteration statement.

The loop continues to execute as long as the condition is true.

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

console.log(i);

}

**while Loop:**

The while loop repeats a block of code as long as a specified condition is true.

The condition is evaluated before the execution of the loop.

let i = 0;while (i < 5) {

console.log(i);

i++;

}

**do-while Loop:**

**The do-while loop is similar to the while loop, but the condition is evaluated after the execution of the loop.**

**This guarantees that the code inside the loop will run at least once.**

let i = 0;do {

console.log(i);

i++;

} while (i < 5);

**for...in Loop:**

The for...in loop is used to iterate over the enumerable properties of an object.

It is often used with objects to loop through their keys.

const person = {

name: 'John',

age: 30,

job: 'Developer'

};

for (let key in person) {

console.log(key, person[key]);

}

**for...of Loop:**

The for...of loop is introduced in ECMAScript 6 and is used to iterate over iterable objects like arrays, strings, and collections.

It simplifies the process of iterating through values.

const numbers = [1, 2, 3, 4, 5];

for (let number of numbers) {

console.log(number);

}

**forEach Method:**

While not a traditional loop, the forEach method is often used to iterate over elements of an array.

It provides a concise way to perform an action on each element of the array.

const fruits = ['apple', 'banana', 'orange'];

fruits.forEach((fruit) => {

console.log(fruit);

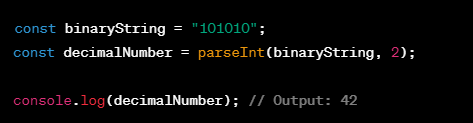
});

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

**Ans.**

In JavaScript, we can convert a string representing a number in any base (binary, octal, hexadecimal, etc.) to an integer using the parseInt function. The parseInt function takes two arguments: the string to be converted, and the radix (base) of the numeral system.

**Here's an example:**



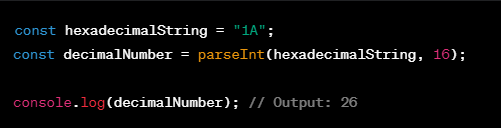
In this example, parseInt(binaryString, 2) converts the binary string "101010" to its decimal (base 10) equivalent, resulting in the number 42.

Similarly, We can use parseInt with different radices to convert strings representing numbers in other bases.

**For example:**

**For octal:** parseInt(octalString, 8)

**For hexadecimal:** parseInt(hexadecimalString, 16)



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

**Ans.**

The delete operator in JavaScript is used to remove a property from an object or an element from an array. Its primary function is to delete a specific property or element, which can be useful in certain scenarios.

**It's important to note that delete has some limitations:**

* It cannot delete variables or function names.
* It does not affect variables or functions.
* It does not delete variables or functions declared with var.
* It is not recommended for use on predefined JavaScript object properties.

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

**Ans.**

In JavaScript, there are three types of pop-up boxes commonly used for user interaction:

**Alert Box:**

* The alert() function displays an alert dialog with a specified message and an OK button.
* It is used for providing information to the user.

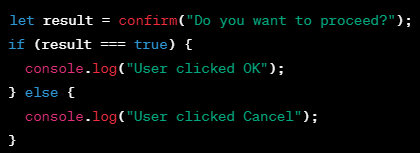
**Example:**



**Confirm Box:**

* The confirm() function displays a dialog box with a specified message, along with OK and Cancel buttons.
* It is used for obtaining user confirmation for an action.

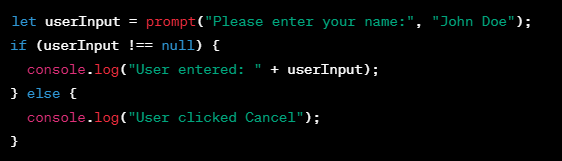
**Example:**



**Prompt Box:**

* The prompt() function displays a dialog box that prompts the user for input.
* It takes two arguments: the message to display, and a default value for the input field.
* The user can enter a value, and it returns the entered value or null if the user clicks Cancel.

**Example:**



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

**Ans.**

**Accessibility:** Using javascript:void(0) as a link can interfere with accessibility features and may not be the best practice for users relying on assistive technologies.

**Script Blocking:** Modern JavaScript development encourages unobtrusive JavaScript and separating behavior from structure. Placing JavaScript directly in HTML attributes is not in line with this approach.

**Inline Event Handlers:** Inline event handlers, like onclick="myFunction()", are generally discouraged due to the separation of concerns and maintainability issues.

**Href Attribute Value:** Modern practices usually set the href attribute to "#" for similar purposes without using javascript:void(0).

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

**Ans.**

To force a page to load another page using JavaScript, we can use the window.location object to manipulate the current URL. There are a few ways to achieve this:

**Setting window.location.href:**

The window.location.href property represents the entire URL of the current page. You can assign a new URL to it to navigate to a different page.

**Redirect to another page**

window.location.href = "https://www.example.com";

**Using window.location.assign():**

The assign() method of the window.location object can also be used to navigate to a different page.

**Redirect to another page**

window.location.assign("https://www.example.com");

**Using window.location.replace():**

The replace() method of the window.location object can be used to load a new page and replace the current entry in the browser's history.

**Redirect to another page and replace the current history entry**

window.location.replace("https://www.example.com");

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

**Ans.**

While the innerHTML property in JavaScript is a powerful and convenient way to manipulate the content of HTML elements, it comes with certain disadvantages and considerations:

**Performance Impact:**

* Manipulating innerHTML often involves re-parsing and rebuilding the entire content of an element, which can be less performant compared to more granular DOM manipulation methods, such as using createElement and appendChild. If you're making frequent updates to a large portion of the DOM, performance might be a concern.

**Event Handlers and Data Binding:**

* If the innerHTML contains elements with attached event handlers or data bindings, overwriting the innerHTML may remove existing event handlers and break data bindings. This can result in unintended behavior, especially in complex applications using frameworks like React or Angular.

**Limited Error Handling:**

* When parsing and rendering HTML using innerHTML, the browser handles errors silently. If there are syntax errors or malformed HTML, the browser may not provide detailed error messages, making it harder to identify and fix issues.

**Content Overwriting:**

* Setting innerHTML replaces the entire content of an element. If you only need to modify a small part of the content, using more targeted DOM manipulation methods (like createElement and appendChild) might be more efficient.

While innerHTML is widely used and appropriate in many situations, developers should be mindful of its potential drawbacks and consider alternative approaches, especially when dealing with user input or when performance is a critical factor. Proper input validation and sanitation should always be applied to prevent security vulnerabilities.