**Module 6**

**JAVASCRIPT BASIC & DOM**

**(Basic logic Question)**

**Q.1 What is JavaScript. How to use it?**

JavaScript is a versatile programming language primarily used for web development. It enables dynamic content on web pages, such as interactive elements, animations, and real-time updates, making web experiences more engaging for users.

Here’s basic overview of JavaScripw1 t and how to use it:

**1.What is JavaScript?**

JavaScript is a high-level, interpreted Programming language that conforms to the ECMAScript specification. It is commonly used for client-side scripting in web development, where it runs within the user's web browser.

**2 How to Use JavaScript:** To use JavaScript in web development, you typically include JavaScript code within your HTML files or link to external JavaScript files.

**Inline JavaScript**: You can include JavaScript directly within HTML using the **<script>** tag. For example:

**Q.2 How many type of Variable in JavaScript?**

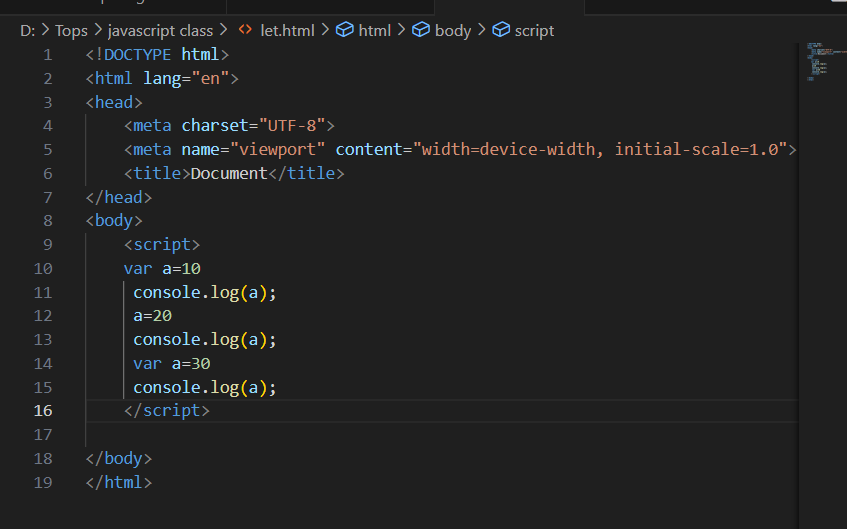
JavaScript variables are used to store data that can be changed later on. The variables can be thought of as named containers. You can place data into these and then refer to the data simply by naming the container.

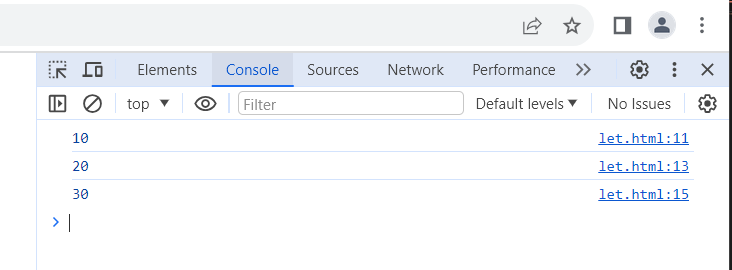
Before you use a variable in a JavaScript program, you must declare it. In JavaScript, you can declare the variable in 4 ways-

* Without using any keywords.
* Using the ‘var’ keyword.
* Using the ‘let’ keyword.
* Using the ‘const’ keyword.

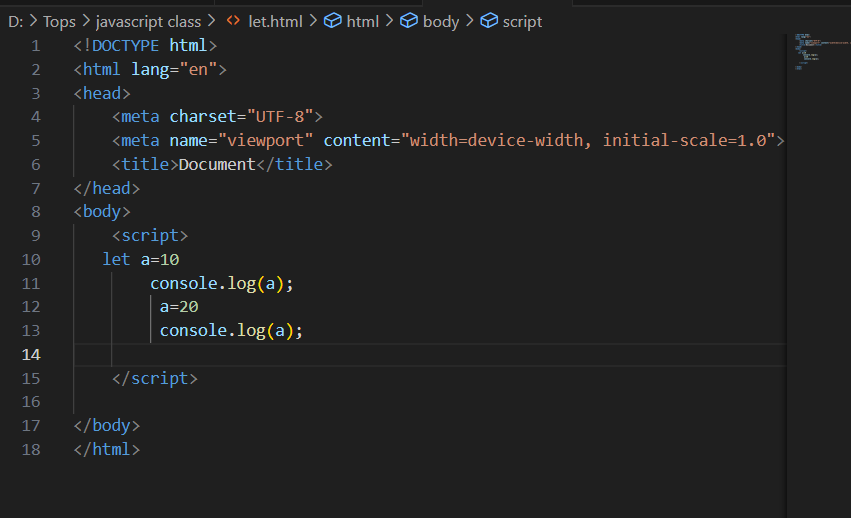
The let and const keywords were introduced to JavaScript. In this section, we will discuss ‘var’ keyword. We will cover the ‘let’ and ‘const’ keywords in subsequent chapters.

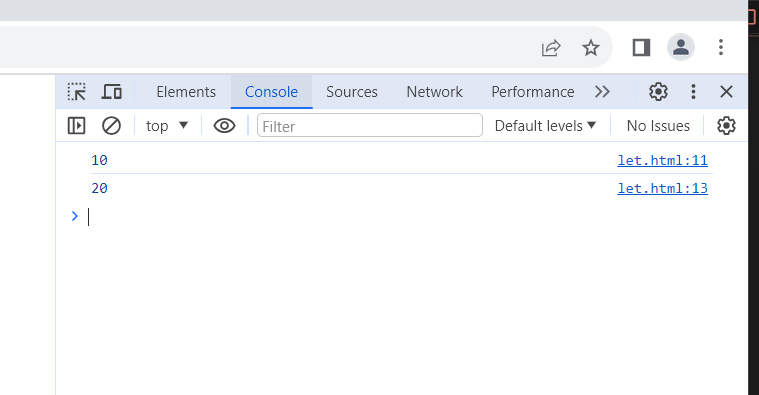
Var Keyword



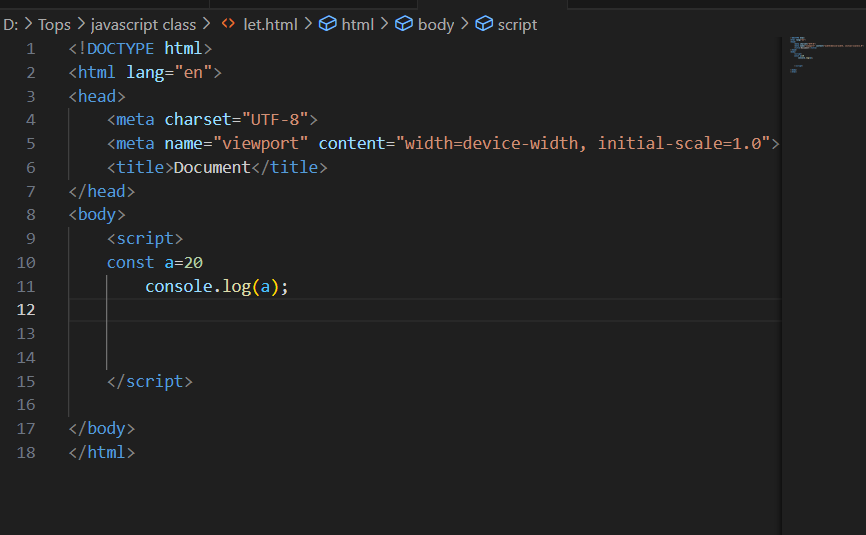
****

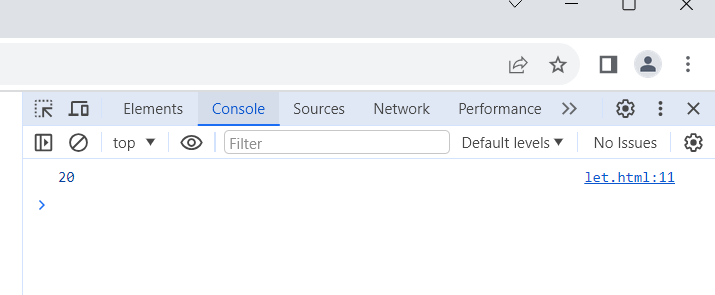
**Let keyword**

****

****

**Const keyword**

****

****

**Q.3 Define a Data Types in js?**

Data types in JavaScript define the data type that a variable can store. JavaScript includes primitive and non-primitive data types. The Primitive data types in JavaScript include string, number, Boolean, undefined, null, and symbol. The non-primitive data type includes the object.

JavaScript Data Types

Primitive Data Types

These are the lowest level of data value in the JavaScript programming language. They define immutable values- which can’t be changed.

String

A string data type is a group of characters or textual enclosed by single Quotes(‘’)or double-quotes(“”), or tick signs(‘’).

For Example:

Var Str 1 = ‘Hello, World!’;

Number

A number data type is a numeric value. It can be an integer (positive or negative), floating-point, or exponential.

For Example:

Var a=10;

Var b=25.5;

Var c= -10;

Boolean

Boolean is a logical data type used to compare two variables or check a condition. It has only two values, namely, true and false.

For Example – Comparing two variables

Var a=10;

Var b=9;

a=b // return false

**Q.4 Write a mul Function Which will Work Properly When invoked With Following Syntax.**

The **MUL function** is a miniature of the multiplication function. In this function, we call the function that required an argument as a first number, and that function calls another function that required another argument and this step goes on.

The first function’s argument is x, the second function`s argument is y and the third is z, so the return value will be xyz.

Syntax:

Function mul(x) {

Return function (y) {

Return Function (z) {

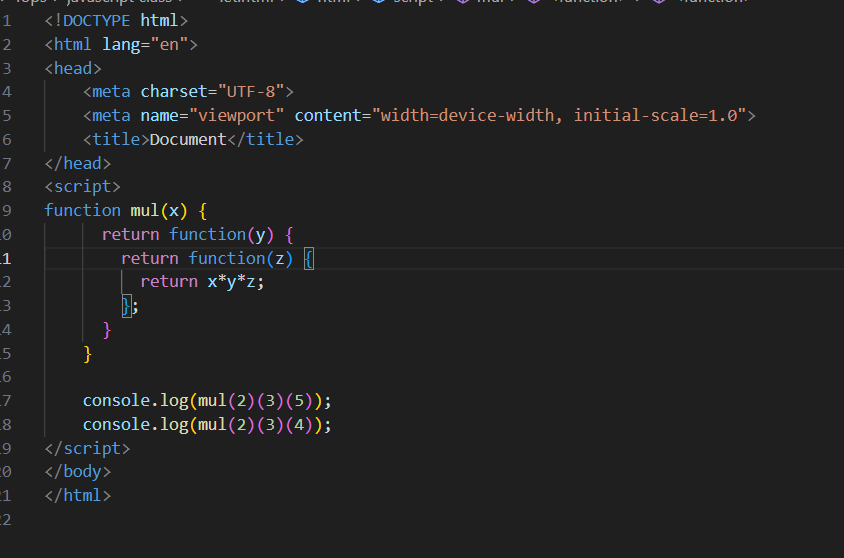
Return x \* y \* z;

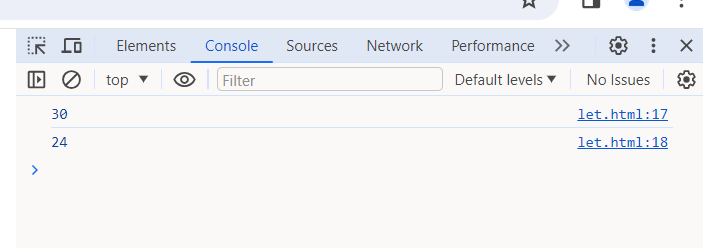
};

};

}

Example of mul function in JavaScript





**Q.5 What the deference between undefined and undeclare in JavaScript?**

Undefined:

It occurs when a variable has been declared but has not been assigned any value. Undefined is not keyword.

Undeclared: It occurs when we try to access any variable that is not initialized or declared earlier using the var or const

keyword.

If we use “typeof’ operator to get the value of an undeclared variable, we will face the runtime error with the return value as “undefined”. The scope of the undeclared variable is always global.

**Q.6 Using console.log() print out the following statement: The quote 'There is no exercise better for the heart than reaching down and lifting people up.' by John Holmes teaches us to help one another. Using console.log() print out the following quote by Mother Teresa:**

**Q.7 Check if typeof '10' is exactly equal to 10. If not make it exactly equal?**

**Q.8 Write a JavaScript Program to find the area of a triangle?**

**Q.9 Write a JavaScript program to calculate days left until next Christmas?**

**Q.10 What is Condition Statement?**

**Q.11 Find circumference of Rectangle formula: C = 4 \* a?**

**Q.12 WAP to convert years into days and days into years?**

**Q.13 Convert temperature Fahrenheit to Celsius? (Conditional logic Question)**

**Q.14 Write a JavaScript exercise to get the extension of a filename.?**

**Q.15 What is the result of the expression (5 > 3 && 2 < 4)?**

The result of the expression (5 > 3 && 2 < 4) is true.

In JavaScript, the && (logical AND) operator returns true if both of its operands are true; otherwise, it returns false.

Let's break down the expression:

5 > 3: This evaluates to true because 5 is greater than 3.

2 < 4: This evaluates to true because 2 is less than 4.

Since both operands are true, the && operator returns true.

Therefore, the overall result of the expression is true.

**Q.16 What is the result of the expression (true && 1 && "hello")?**

The result of the expression (true && 1 && "hello") is "hello".

In JavaScript, when using the logical AND (&&) operator, the expression evaluates to the last truthy value if all operands are truthy, or to the first falsy value encountered.

Let's break down the expression:

true && 1: Both operands are truthy, so the expression evaluates to the last truthy value, which is 1.

1 && "hello": Both operands are truthy, so the expression evaluates to the last truthy value, which is "hello".

Therefore, the overall result of the expression is "hello".

**Q.17 What is the result of the expression true && false || false && true?**

The expression true && false || false && true is evaluated based on the precedence and associativity of logical operators.

In JavaScript, the logical AND (&&) operator has higher precedence than the logical OR (||) operator. Additionally, the logical AND (&&) operator is left-associative, while the logical OR (||) operator is also left-associative.

Here's the step-by-step evaluation of the expression:

true && false: This evaluates to false because both operands must be true for the AND operation to result in true.

false || false: This evaluates to false because both operands are false, and only one operand needs to be true for the OR operation to result in true.

false && true: This evaluates to false because both operands must be true for the AND operation to result in true.

So, the final result of the expression is false.

**Q.18 What is a Loop and Switch Case in JavaScript define that ?**

In JavaScript, a loop is a programming construct that allows you to repeatedly execute a block of code until a specified condition is met. There are several types of loops available in JavaScript, including **for**, **while**, and **do...while** loops. Here's a brief explanation of each:

**for Loop**:

* The **for** loop is used when you know the number of iterations you want to perform.
* It consists of three parts: initialization, condition, and increment/decrement.
* The loop iterates as long as the condition evaluates to true.
* Example:

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

**console.log(i);**

**}**

**while Loop**:

* The **while** loop is used when you want to execute a block of code while a condition is true.
* It only evaluates the condition before executing the block of code.
* Example:

**let i = 0;**

**while (i < 5) {**

**console.log(i);**

**i++;**

}

do...while Loop:

The do...while loop is similar to the while loop, but it always executes the block of code at least once before checking the condition.

It evaluates the condition after executing the block of code.

Example:

**let i = 0;**

**do {**

**console.log(i);**

**i++;**

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

A switch case is another control flow statement in JavaScript used to perform different actions based on different conditions. It's often used as an alternative to long chains of **if...else** statements when you have multiple conditions to evaluate.

Here's how a switch case statement works:

**let color = "red";**

**switch (color) {**

**case "red":**

**console.log("The color is red.");**

**break;**

**case "green":**

**console.log("The color is green.");**

**break;**

**case "blue":**

**console.log("The color is blue.");**

**break;**

**default:**

**console.log("Unknown color.");**

}

The switch statement evaluates the expression (in this case, the variable color) and compares it to each case value.

If a case value matches the expression, the corresponding block of code is executed.

The break statement is used to exit the switch block after a match is found. If omitted, execution will continue to the next case.

If no case value matches the expression, the code inside the default block is executed. This block is optional.

**Q.19 What is the use of is Nan function?**

The isNaN() function in JavaScript is used to determine whether a value is "Not-a-Number" (NaN).

Here's how it works:

If the argument of the isNaN() function is not a number (i.e., it cannot be converted to a number), true is returned.

If the argument is a number or can be converted to a number, false is returned.

Example usage:

**isNaN(NaN); // true**

**isNaN(123); // false**

**isNaN('hello'); // true (cannot be converted to a number)**

**isNaN('123'); // false (can be converted to a number)**

**isNaN(true); // false (can be converted to a number: 1)**

Keep in mind that **isNaN()** attempts to convert the argument to a number before determining if it is NaN. If the argument is a non-numeric string that cannot be converted to a number, **isNaN()** returns **true**. However, if the argument is a numeric string that can be converted to a number, **isNaN()** returns **false**. Therefore, **isNaN()** is not always reliable for checking whether a value is a number or not, especially when dealing with strings. In cases where you need to determine if a value is a valid number, you might consider using other methods like **typeof** or **Number.isNaN()** (introduced in ECMAScript 2015) for more accurate results.

**Q.20 What is the difference between && and || in JavaScript?**

In JavaScript, ‘&&’ and ‘||’ are logical operators used for Boolean operations. Here’s the difference between them:

1 ‘&&’ (Logical AND):

The ‘&&’ operands are true if both operands are **true**, and **false** otherwise.

It follows the short-circuit evaluation rule:

If the left operand evaluates to false, the expression short-circuits, and the right operand is not evaluated. The result is false.

If the left operand evaluates to true, the right operand is evaluated, and the result depends on the value of the right operand.

|| (Logical OR):

The || operator returns true if at least one of its operands is true, and false otherwise.

It also follows the short-circuit evaluation rule:

If the left operand evaluates to true, the expression short-circuits, and the right operand is not evaluated. The result is true.

If the left operand evaluates to false, the right operand is evaluated, and the result depends on the value of the right operand.

Here are some key differences between && and ||:

&& returns true only if both operands are true, whereas || returns true if at least one operand is true.

If the left operand of && is false, the right operand is not evaluated. Similarly, if the left operand of || is true, the right operand is not evaluated. This behavior is known as short-circuit evaluation.

Both && and || operators are often used in conditional statements to combine multiple conditions.

**Q.21 What is the use of Void (0)?**

In JavaScript, void(0) is an expression that evaluates to undefined. The void operator in JavaScript evaluates the expression within its parentheses and then returns undefined. The purpose of using void(0) is typically to create a JavaScript snippet that evaluates to undefined without causing any side effects or navigation.

One common use of void(0) is in anchor (<a>) elements' href attributes to create non-navigating links or to prevent the default behavior of a link when clicked. By setting the href attribute to javascript:void(0), clicking the link does not cause the browser to navigate to a new page. This is useful in situations where you want to attach an event handler to a link without actually navigating to a different page.

This technique is often used when you want to attach an event handler to the link programmatically using JavaScript, without actually navigating away from the current page.

**Q.22 Check Number Is Positive or Negative in JavaScript?**

**Q.23 Find the Character Is Vowel or Not ?**

**Q.24 Write to check whether a number is negative, positive or zero?**

**Q.25 Write to find number is even or odd using ternary operator in JS?**

**Q.26 Write find maximum number among 3 numbers using ternary operator in JS?**

**Q.27 Write to find minimum number among 3 numbers using ternary operator in JS?**

**Q.28 Write to find the largest of three numbers in JS?**

**Q.29 Write to show i. Monday to Sunday using switch case in JS? ii. Vowel or Consonant using switch case in JS? (Conditional looping logic Question)**

**Q.30 What are the looping structures in JavaScript? Any one Example?**

**Q.31 Write a print 972 to 897 using for loop in JS?**

To print numbers from 972 to 897 using a **for** loop in JavaScript, you can start with the initial value of 972 and decrement the value by 1 in each iteration until it reaches 897. Here's how you can do it:

**for (let i = 972; i >= 897; i--) {**

**console.log(i);**

**}**

In this **for** loop:

* **i** is initialized to **972**.
* The loop continues as long as **i** is greater than or equal to **897**.
* In each iteration, **i** is decremented by 1.
* The loop prints the value of **i** in each iteration, starting from 972 and ending at 897.

**Q.32 Write to print factorial of given number?**

**Q.33 Write to print Fibonacci series up to given numbers?**

**Q.34 Write to print number in reverse order e.g.: number = 64728 ---> reverse =82746 in JS?**

**Q.35 Write a program make a summation of given number (E.g., 1523 Ans: - 11) in JS?**

**Q.36 Write a program you have to make a summation of first and last Digit. (E.g., 1234 Ans: 5) in JS?**

**Q.37 Use console.log() and escape characters to print the following pattern in JS?**

**Q.38 Use pattern in console.log in JS?**

**Q.39 Accept 3 numbers from user using while loop and check each numbers palindrome? (Array and object Question)**

**Q.40 Write a JavaScript Program to display the current day and time in the following format. Sample Output: Today is Friday. Current Time is 12 PM: 12: 222?**

**Q.41 Write a JavaScript program to get the current date?**

**Q.42 Write a JavaScript program to compare two objects?**

**Q.43 Write a JavaScript program to convert an array of objects into CSV string?**

**Q.44 Write a JavaScript program to capitalize first letter of a string?**

**Q. 45 Write a JavaScript program to determine if a variable is array?**

**Q.46 Write a JavaScript program to clone an array?**

If you want to perform some operations on an array, such as sorting, filtering, or mapping, but you don’t want to modify the original array, you can create a clone of the original array and perform the operations on the clone instead.

* When passing an array to a function as an argument, you may want to ensure that the function does not modify the original array. In this case, you can pass a clone of the array instead.
* If you want to preserve the original array for future reference, you can create a clone of the original array and use the clone for further processing or manipulation.
* If you have an array that contains objects or arrays as elements, and you want to avoid modifying the original objects or arrays, you can create a clone of the array to work with, so that changes to the objects or arrays in the clone do not affect the original objects or arrays.

Thus,**cloning an array in JavaScript** is a useful technique for working with arrays in a way that preserves the integrity of the original array and its elements.

**Q.48 Print the length of the string on the browser console using console.log()? hcapital letters using toUpperCase() method?**

To print the length of a string and its uppercase version on the browser console using **console.log()**, you can write JavaScript code like this:

**const str = "Hello, World!";**

**// Print the length of the string**

**console.log("Length of the string:", str.length);**

**// Print the string in uppercase**

**console.log("Uppercase version of the string:", str.toUpperCase());**

When you execute this code in a browser environment and check the console, you will see the length of the string and its uppercase version printed as follows:

**Q.50 What is the drawback of declaring methods directly in JavaScript objects?**

In JavaScript, declaring methods directly in objects can lead to potential drawbacks, primarily related to memory usage and performance optimization. Here are some drawbacks:

1. **Memory Usage**: When you declare methods directly in JavaScript objects, each instance of the object will have its own copy of the methods.
2. **Performance Impact**: Since each instance of the object has its own copy of methods, it can impact the performance of your application, especially in scenarios where you have many instances of the object or when the methods are complex and consume significant CPU resources.
3. **Inflexibility**: Declaring methods directly in objects can make your code less modular and harder to maintain. It becomes challenging to reuse the methods across different objects or to update them without affecting all instances of the object.
4. **Difficulty in Managing Prototypes**: When methods are directly declared in objects, it becomes more difficult to leverage JavaScript's prototype-based inheritance.

To mitigate these drawbacks, it's often recommended to use prototype-based inheritance or to define methods outside of the object and attach them to the object's prototype. Additionally, it makes your code more modular and easier to maintain in the long run.

Top of Form

**Q.51 Write a JavaScript program to get the current date. Expected Output : mm-dd-yyyy, mm/dd/yyyy or dd-mm-yyyy, dd/mm/yyyy?**

**Q.52 Use indexOf to determine the position of the first occurrence of a in 30 Days Of JavaScript?**

You can use the **indexOf()** method in JavaScript to determine the position of the first occurrence of a specific substring within a string. Here's how you can use it to find the position of the first occurrence of **'a'** in the string **'30 Days Of JavaScript'**:

**const str = '30 Days Of JavaScript';**

**const position = str.indexOf('a');**

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

In this code:

* **str** is the string '30 Days Of JavaScript'.
* **str.indexOf('a')** returns the position of the first occurrence of **'a'** within the string. The index is zero-based, so the first character has an index of **0**, the second character has an index of **1**, and so on.

So, in the given string **'30 Days Of JavaScript'**, the first occurrence of **'a'** is at position **8**.

**Q,53 Use lastIndexOf to determine the position of the last occurrence of a in 30 Days Of JavaScript?**

You can use the **lastIndexOf()** method in JavaScript to determine the position of the last occurrence of a specific substring within a string. Here's how you can use it to find the position of the last occurrence of **'a'** in the string **'30 Days Of JavaScript'**:

**const str = '30 Days Of JavaScript';**

**const position = str.lastIndexOf('a');**

**console.log(position); // Output: 19**

In this code:

* **str** is the string **'30 Days Of JavaScript'**.
* **str.lastIndexOf('a')** returns the position of the last occurrence of **'a'** within the string. The index is zero-based, so the first character has an index of **0**, the second character has an index of **1**, and so on.
* If the substring **'a'** is found within the string, **lastIndexOf()** returns its position (index), which is **19** in this case.
* If the substring is not found, **lastIndexOf()** returns **-1**.

So, in the given string **'30 Days Of JavaScript'**, the last occurrence of **'a'** is at position **19**.

**Q.54 Form Validtion in JS?**

**Q.55 Form in Email, number, Password, Validation?**

* The form has input fields for email, phone number, and password, along with a submit button.
* When the form is submitted, the **validateForm()** function is called to perform validation.
* The **validateForm()** function retrieves the values of the input fields and performs validation checks for each field.
* Email validation checks if the email field is not empty and matches a basic email format using a regular expression.
* Phone number validation checks if the phone number field is not empty and matches a specific format (e.g., xxx-xxx-xxxx).
* Password validation checks if the password field is not empty and has a minimum length of 8 characters.
* If any validation check fails, an alert message is displayed, and the form submission is prevented by returning **false**. Otherwise, the form is considered valid, and the submission proceeds.

**Q.56 Dynamic Form Validation in JS? Q.57 how many type of JS Event? How to use it ?**

In JavaScript, there are several types of events that can occur during the lifecycle of a web page or web application. These events represent various user interactions, browser actions, and system events. Here are some common types of JavaScript events:

1. **Mouse Events**:
   * **click**: Fires when a mouse button is clicked.
   * **mouseover**: Fires when the mouse pointer enters the area of an element.
   * **mouseout**: Fires when the mouse pointer leaves the area of an element.
   * **mousedown**: Fires when a mouse button is pressed down on an element.
   * **mouseup**: Fires when a mouse button is released over an element.
   * **mousemove**: Fires when the mouse pointer is moved over an element.
2. **Keyboard Events**:
   * **keydown**: Fires when a key is pressed down.
   * **keyup**: Fires when a key is released.
   * **keypress**: Fires when a key that produces a character value is pressed down.
3. **Form Events**:
   * **submit**: Fires when a form is submitted.
   * **change**: Fires when the value of an input element changes.
   * input: Fires when the value of an input element changes (similar to change, but with broader support and fires immediately).
   * focus: Fires when an element receives focus.
   * blur: Fires when an element loses focus.
4. **Window Events**:
   * **load**: Fires when the entire page (including all images and scripts) is loaded.
   * **unload**: Fires when the user navigates away from the page.
   * **resize**: Fires when the browser window is resized.
   * **scroll**: Fires when the user scrolls the page.
5. **Touch Events**:
   * **touchstart**: Fires when a touch point is placed on the touch surface.
   * **touchmove**: Fires when a touch point is moved along the touch surface.
   * **touchend**: Fires when a touch point is removed from the touch surface.
   * **touchcancel**: Fires when a touch event is interrupted (e.g., by the browser).

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

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

**<title>JavaScript Events</title>**

**</head>**

**<body>**

**<button id="myButton">Click me</button>**

**<script>**

**// Get the button element**

**const button = document.getElementById('myButton');**

**// Add a click event listener to the button**

**button.addEventListener('click', function(event) {**

**console.log('Button clicked!');**

**});**

**</script>**

**</body>**

**</html>**

In this example, the **addEventListener()** method is used to attach a click event listener to the button element. When the button is clicked, the anonymous function passed to **addEventListener()** is executed, logging a message to the console.

**Q.59 What is Bom vs Dom in JS?**

In JavaScript, BOM and DOM are two distinct but related concepts:

1. **BOM (Browser Object Model)**:
   * The Browser Object Model (BOM) represents everything in the browser environment that is not directly related to the document being displayed. It provides JavaScript interfaces for interacting with the browser window and its components, such as the history, location, navigator, screen, and more.
   * Some commonly used objects and properties in the BOM include **window**, **document**, **location**, **navigator**, **history**, **screen**, **localStorage**, **sessionStorage**, etc.
   * The BOM allows JavaScript to interact with and control various aspects of the browser window and the browsing environment, such as opening and closing windows, navigating to different URLs, storing data locally, detecting browser properties, and handling user interactions.
2. **DOM (Document Object Model)**:
   * The Document Object Model (DOM) represents the hierarchical structure of an HTML or XML document as a tree of objects. It provides a way for scripts to dynamically access, manipulate, and update the content, structure, and style of web pages.
   * In the DOM, each HTML element in a web page is represented as a node in the document tree, and the relationship between elements is expressed through parent-child relationships.
   * The DOM provides JavaScript interfaces for accessing and manipulating elements and their attributes, properties, and content. Common methods and properties include **getElementById()**, **querySelector()**, **innerHTML**, **textContent**, **setAttribute()**, **appendChild()**, etc.
   * The DOM allows JavaScript to dynamically modify the content and structure of web pages in response to user actions, events, or application logic.

In summary, the BOM represents the browser window and its components, while the DOM represents the structure and content of an HTML or XML document. Both BOM and DOM are essential for building interactive and dynamic web applications with JavaScript.

**Q.60 Array vs object defences in JS?**

In JavaScript, both arrays and objects are used for storing and organizing data, but they serve different purposes and have different characteristics. Here's a comparison of arrays and objects in JavaScript:

### Arrays:

1. **Ordered Collection**:
   * Arrays are ordered collections of values. Each value in an array is identified by an index, starting from **0** for the first element.
2. **Use Cases**:
   * Arrays are typically used when you need to store a collection of values that are related and ordered, such as a list of items or a series of values.
3. **Accessing Elements**:
   * You can access elements in an array using square brackets (**[]**) notation with the index of the element.
4. **Length Property**:
   * Arrays have a **length** property that indicates the number of elements in the array.
5. **Methods**:
   * Arrays come with a variety of built-in methods for manipulating and working with the elements, such as **push()**, **pop()**, **shift()**, **unshift()**, **slice()**, **splice()**, **forEach()**, **map()**, etc.

### Objects:

1. **Key-Value Pairs**:
   * Objects are collections of key-value pairs, where each key is a unique string (or symbol) that maps to a value.
2. **Use Cases**:
   * Objects are used when you need to store data in a key-value format or when you need to represent entities with properties and values.
3. **Accessing Properties**:
   * You can access properties in an object using dot notation (**object.property**) or bracket notation (**object['property']**).
4. **No Ordering**:
   * Unlike arrays, objects do not guarantee any particular order for their properties. The order of properties in an object is not guaranteed to be the same as the order in which they were defined.
5. **Methods**:
   * Objects do not have built-in methods like arrays do. However, you can iterate over the properties of an object using **for...in** loop or use methods like **Object.keys()**, **Object.values()**, and **Object.entries()** to work with object properties.

### Defenses:

1. **Array**:
   * Arrays are suitable for storing homogeneous data or a collection of similar items. They provide easy access to elements using indices.
   * Arrays are best used when you need ordered data or when you need to perform operations like sorting or filtering based on index.
   * Use arrays when you need to iterate through items in a specific order or when you need to perform operations like adding, removing, or updating elements in a collection.
2. **Object**:
   * Objects are suitable for storing heterogeneous data or a collection of key-value pairs. They provide easy access to values using keys.
   * Objects are best used when you need to represent entities with properties or when you need to access data based on specific keys.
   * Use objects when you need to store data in a structured format or when you need to perform operations like looking up values based on keys or adding/removing properties dynamically.

Choosing between arrays and objects depends on the specific requirements of your application and the type of data you need to store and manipulate. In many cases, you may find that you need to use both arrays and objects together to effectively manage your data.

**Q.61 Split the string into an array using split() Method?**

In JavaScript, you can use the **split()** method to split a string into an array of substrings based on a specified delimiter. Here's how you can use it:

**let str = "Hello World";**

**let arrayOfWords = str.split(" ");**

**console.log(arrayOfWords); // Output: ["Hello", "World"]**

In this example, **split(" ")** splits the string **"Hello World"** into an array of substrings using the space character (" ") as the delimiter. As a result, **arrayOfWords** contains **["Hello", "World"]**, where each word from the original string is an element in the array.

**Q.62 Check if the string contains a word Script using includes() method?**

You can use the **includes()** method in JavaScript to check if a string contains a specific substring. Here's how you can use it to check if the string contains the word "Script":

**let str = "30 Days of JavaScript";**

**let wordToCheck = "Script";**

**if (str.includes(wordToCheck)) {**

**console.log("The string contains the word 'Script'.");**

**} else {**

**console.log("The string does not contain the word 'Script'.");**

**}**

In this example, **includes()** is used to check if the string **str** contains the word **"Script"**. If the word is found within the string, **includes()** returns **true**, and the message "The string contains the word 'Script'." is logged to the console. Otherwise, it returns **false**, and the message "The string does not contain the word 'Script'." is logged.

**Q.63 Change all the string characters to lowercase letters using toLowerCase() Method.**

In JavaScript, you can use the **toLowerCase()** method to convert all characters in a string to lowercase. Here's how you can use it:

**let str = "Hello World";**

**let lowerCaseStr = str.toLowerCase();**

**console.log(lowerCaseStr); // Output: hello world**

In this example, **toLowerCase()** is applied to the string **"Hello World"**, and it returns a new string where all characters are converted to lowercase, resulting in **"hello world"**. The original string **str** remains unchanged.

**Q.64 What is Character at index 15 in ’30 Days of JavaScript’ string? Use charAt() method.**

In JavaScript, you can use the charAt() method to retrieve the character at a specific index in a string. The charAt() method returns the character at the specified index as a new string.

Here's how you can use charAt() to find the character at index 15 in the string '30 Days of JavaScript':

**let str = '30 Days of JavaScript';**

**let characterAtIndex15 = str.charAt(15);**

**console.log(characterAtIndex15); // Output: J**

In this example, charAt(15) returns the character **'J'**, which is at index 15 in the string '30 Days of JavaScript'.

Top of Form

**Q.65 copy to one string to another string in JS?**

To copy the content of one string to another string in JavaScript, you can simply assign the value of one string variable to another string variable. Here's an example:

**let sourceString = "This is the source string.";**

**let destinationString;**

**// Copy the content of the source string to the destination string**

**destinationString = sourceString;**

**console.log(destinationString); // Output: This is the source string.**

In this example, the content of **sourceString** is copied to **destinationString** by assigning **sourceString** to **destinationString**. After the assignment, **destinationString** contains the same content as **sourceString**

**Q.66 Find the length of a string without using libraryFunction?**

**• What is JavaScript?**

JavaScript is a high-level, interpreted programming language primarily used for building dynamic and interactive websites. It is one of the core technologies of web development along with HTML and CSS. JavaScript allows developers to add functionality, and behavior to web pages.

Here some key points about JavaScript:

1. Client-side scripting: JavaScript is mainly used as a client-side scripting language, meaning it runs in the web browser of the user.
2. Versatility: JavaScript is a versatile language that can be used for various purpose. Apart from web development, it can be used for server-side development(with platforms like Node.js), mobile app development(with framework like React Native), desktop app development, game development and more.
3. Object-oriented and functional programming: JavaScript supports both object-oriented programming (OOP) and functional programming paradigms.
4. Dynamic typing: JavaScript is dynamically typed, meaning you don’t need to declare the data type of variables explicitly.
5. Large ecosystem: JavaScript has a vast ecosystem of libraries, frameworks, and tools that simplify and accelerate web development.

Overall, JavaScript plays a crucial role in modern web development, enabling developers to create dynamic and interactive web applications that provide a rich user experience.

**• What is the use of isNaN function?**

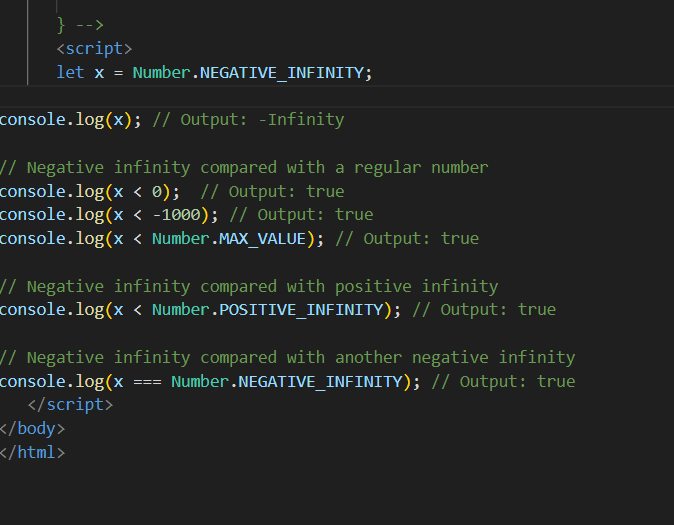
**• What is negative Infinity?**

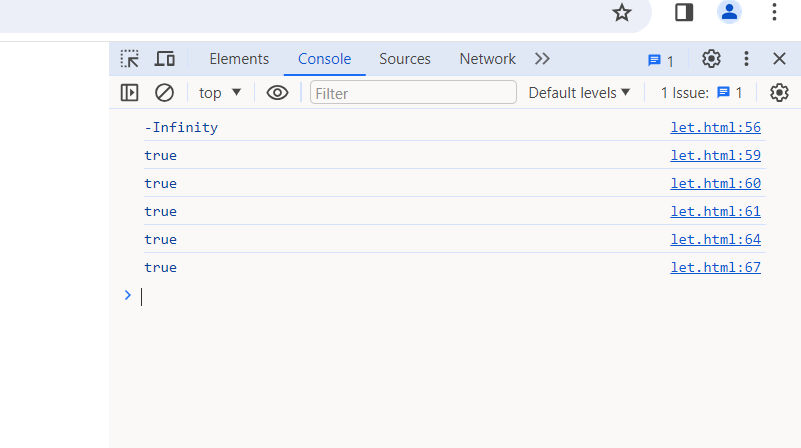
Negative infinity is a special value in JavaScript that represents the mathematical concept of negative infinity. It is the lowest possible numerical value, indicating a value that is smaller (or “less than”) any other number, including negative numbers.

In JavaScript, negative infinity is represented by the global property

‘Number.NEGATIVE\_INFINITY’. It is used to denote values that are too small or too negative to be represented by regular numerical values.

Here’s an example of how negative infinity is used in JavaScript:





In JavaScript, operations involving negative infinity generally behave as expected from a mathematical standpoint. For example, any number compared with negative infinity will evaluate to true if the number is greater than negative infinity.

Negative infinity is also less than positive infinity, and negative infinity is equal to itself.

**• Which company developed JavaScript?**

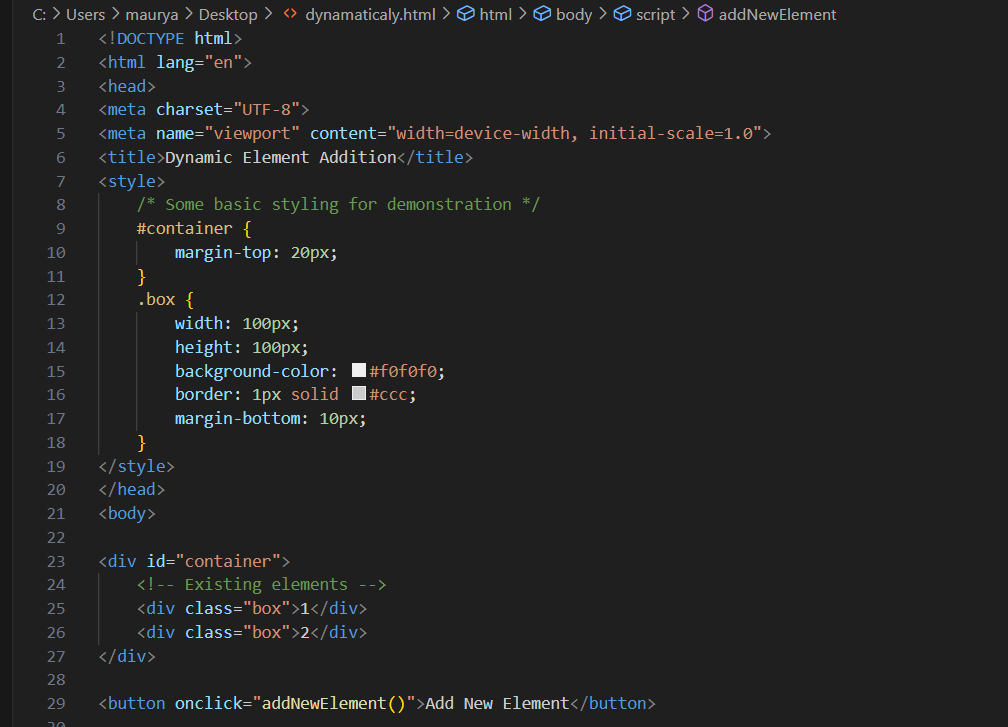
JavaScript was developed by Netscape Communications Corporation, a company founded in 1994 by Marc Andreessen and Jim Clark. Netscape originally developed JavaScript under the name "Mocha" and later "LiveScript" before settling on the name JavaScript. The initial development of JavaScript was led by Brendan Eich, who implemented the language in a span of 10 days in May 1995. JavaScript was first introduced in Netscape Navigator 2.0, released in September 1995. Later, Netscape submitted JavaScript to the Ecma International standards organization, leading to the creation of the ECMAScript specification, which serves as the standard for JavaScript implementations.

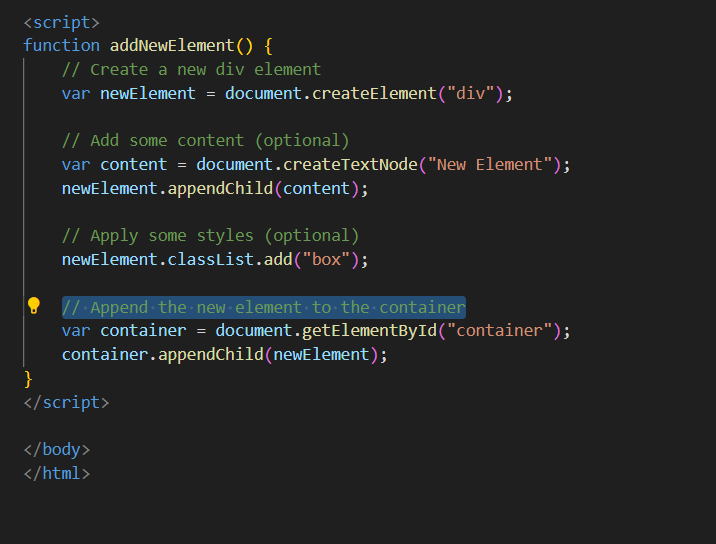
**• What is the use of isNaN function?**

**variables?**

**• Write the code for adding new elements dynamically?**

Certainly! Below is an example of how you can add new elements dynamically to a webpage using JavaScript:







In this example:

The HTML document contains a container div (‘<div id=’’containers’’>’) where new elements will be added dynamically.

Inside the container, there are initially two div elements with the class “box” for demonstration purpose.

You can click the “Add new elements” button to dynamically add new div elements with the content “New Element” to the container.

**• What is the difference between ViewState and SessionState?**

**ViewState** and **SessionState** are both mechanisms used in ASP.NET for preserving state information across multiple requests in a web application, but they serve different purposes and have different scopes

**ViewState**:

* **ViewState** is used to preserve the state of a single page across multiple requests. It is specific to a single web page.
* **ViewState** is maintained as a hidden field on the page and is used to store the state of controls and other page-specific information.
* **ViewState** is client-side, meaning the data is stored in the HTML sent to the client's browser and is returned to the server with each subsequent request.

**SessionState**:

* **SessionState** is used to preserve user-specific data across multiple requests for the duration of a user's session.
* **SessionState** is stored on the server and is unique to each user session.
* It can store any serializable object, such as user preferences, shopping cart contents, or user authentication details.
* **SessionState** can be configured to use different storage mechanisms, such as in-process (stored in memory), out-of-process (stored in a separate process, like State Server or SQL Server), or using custom providers.
* It's important to note that **SessionState** can have performance implications, especially when storing large amounts of data or when using out-of-process storage, as it may involve serialization and network overhead.

In summary, **ViewState** is used for maintaining state within a single web page, while **SessionState** is used for maintaining state across multiple requests for a specific user session.

**• What is === operator?**

The **===** operator is a comparison operator used in various programming languages, including JavaScript. It is known as the "strict equality operator."

In JavaScript specifically:

* The **===** operator checks for both equality of value and equality of type. It returns **true** if the operands are equal and of the same type, and **false** otherwise.
* It does not perform type coercion. This means that if the operands are of different types, they are considered unequal, even if their values are similar after type coercion.
* For example, **1 === 1** would return **true**, but **1 === '1'** would return **false** because the types are different.

Here's a quick comparison between **==** and **===** in JavaScript:

* **==** is known as the "abstract equality operator" or "loose equality operator." It performs type coercion if the operands are of different types before comparing their values.
* **===** is known as the "strict equality operator." It does not perform type coercion and checks both value and type.

Using **===** is generally recommended over **==** because it avoids unexpected behavior due to type coercion and leads to more predictable code.

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

In web development, you can change the style or class of an HTML element using various methods. Here are some common ways:

**Inline Style Attribute**:

You can directly set the style of an element using the **style** attribute. This allows you to apply CSS styles directly to the element. For example:

**Using JavaScript**:

You can change the style of an element dynamically using JavaScript. You can access the element using its ID, class, or other selectors, and then modify its **style** property. For example:

<div id = “mydiv”>This is a div.</div>

<script>

Var element = document.getelementById(“myDiv”);

element.style.color = “blue”

element.style.fontsize = “18px”;

</script>

**Adding/Removing Classes**:

You can define CSS classes with predefined styles and then add or remove these classes to/from elements using JavaScript. For example:

**Using CSS Selectors**:

You can use CSS selectors to target elements and apply styles using an external CSS file or within a **<style>** block in your HTML. For example:

<style>

.myclass{

Color:green;

Font-size: 20px;

}

</style>

<div class =” myclass”>This is a div.</div>

In this case, the style is applied based on the class name **myClass**.

These are some of the primary methods for changing the style or class of an HTML element. The choice of method depends on the specific requirements and structure of your web application.

**• How to read and write a file using JavaScript?**

In JavaScript, you can read and write files using different methods depending on the environment in which your code is running. In a web browser environment, JavaScript typically does not have direct access to the file system for security reasons. However, you can interact with files indirectly, such as handling file uploads and downloads via user interactions. In a server-side environment like Node.js, you have more direct access to the file system.

Here are examples of how to read and write files in both a browser environment and a Node.js environment:

**Reading a File:**

#### **Browser Environment:**

In a browser environment, you can read files using the FileReader API. Here's an example of how to read a text file:

<input type=”file” id=”fileInput”>

<script>

Document.getElementById(‘fildInput’).addEvenListener(‘change’,function(event){

Var file=event.target.files[0];

Var reader = new FileReader():

reader.onload = function(event){

console.log(event.target,result);

});

</script>

**• What are all the looping structures in JavaScript?**

In JavaScript, there are several looping structures available for iterating over collections, arrays, or executing a block of code repeatedly. Here are the main looping structures:

1 for Loop:

The ‘for’ loop is one of the most commonly used looping structures. It allows you to execute a block of code a specified number of times.

**for (initialization; condition; iteration) {**

**// Code to be executed**

**}**

2 While Loop:

The ‘while’ loop repeats a block of codes as long as a specified condition evaluates

To ‘true’.

**while (condition) {**

**// Code to be executed**

**}**

3 do…while Loop

The ‘do…while’ loop is similar to the ‘while’ loop, but it guarantees that the code block is executed at least one before checking the condition.

**do {**

**// Code to be executed**

**} while (condition);**

**For…in loop:**

The ‘for…in’ loop iterates over the enumerable properties of an object. It’s commonly used for iterating over object properties.

**for (variable in object) {**

**// Code to be executed**

**}**

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

**• What is the function of the delete operator?**

**• What are all the types of Popup boxes available in JavaScript?**

In JavaScript, there are three types of popup boxes commonly used for interacting with users:

**Alert Box**: An alert box is used to display a message to the user. It typically contains an OK button, and when the user clicks it, the alert box disappears.

**alert("This is an alert box!");**

**Confirm Box**: A confirm box is used to ask the user for confirmation. It typically contains OK and Cancel buttons, and the user can choose either option. It returns **true** if the user clicks OK and **false** if the user clicks Cancel.

**let result = confirm("Are you sure you want to delete this item?");**

**if (result) {**

**// Delete the item**

**} else {**

**// Cancel the operation**

**}**

**Prompt Box**: A prompt box is used to prompt the user to enter some input. It contains a text input field along with OK and Cancel buttons. It returns the value entered by the user if OK is clicked, and **null** if Cancel is clicked.

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

**if (name) {**

**console.log("Hello, " + name + "!");**

**} else {**

**console.log("No name entered.");**

**}**

These popup boxes are useful for obtaining user input or providing information in a user-friendly manner. However, they should be used judiciously, as excessive use of popup boxes can disrupt the user experience.

**• What is the use of Void (0)?**

The **void** operator in JavaScript is used to evaluate an expression and return **undefined**. The expression following the **void** keyword is evaluated, but its result is discarded, and **undefined** is returned.

The common use of **void(0)** is to create a JavaScript snippet that evaluates to **undefined** without causing any side effects or navigation. It's often used in **href** attributes of **<a>** elements to create non-navigating links or prevent the default behavior of a link.

**<a href="javascript:void(0)">Click me</a>**

In this example, clicking the link does not cause the browser to navigate to a new page because the expression **void(0)** evaluates to **undefined**. This is often used in situations where you want to attach an event handler to a link without actually navigating to a different page.

It's worth noting that using **void(0)** in modern JavaScript code is less common due to improvements in event handling and the availability of other techniques such as **event.preventDefault()** or **return false** in event handlers. Additionally, using inline JavaScript in HTML attributes is considered bad practice for separating concerns and maintaining clean, maintainable code.

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

In JavaScript, you can force a page to load another page by setting the **window.location** property to the URL of the page you want to load. This causes the browser to navigate to the specified URL.

Here's how you can do it:

**// Redirect to another page**

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

This line of code will redirect the current page to "[https://www.example.com](https://www.example.com/)". The browser will load the new page, replacing the current page in the browser history.

You can also use other properties of window.location to control navigation behavior:

window.location.href: Sets the entire URL of the page.

window.location.assign(url): Loads the specified URL.

window.location.replace(url): Replaces the current page in the browser history with the specified URL.

window.location.reload(): Reloads the current page.

It's important to note that when using JavaScript to force a page to load another page, it typically happens immediately, so any JavaScript code after the redirection will not be executed. If you want to perform some action before redirecting, you can delay the redirection using **setTimeout()** or handle it asynchronously. Additionally, you should consider the implications of redirecting users without their explicit action, as it can disrupt their browsing experience.

**• What are the disadvantages of using innerHTML in JavaScript?**

While **innerHTML** in JavaScript provides a convenient way to manipulate the content of HTML elements, it also comes with some disadvantages and potential risks:

**Security Vulnerabilities**: Using **innerHTML** to insert or update HTML content can expose your application to cross-site scripting (XSS) attacks if the inserted content is not properly sanitized. If the content being inserted includes untrusted data from user input, it could contain malicious scripts that get executed within your application's context.

**Performance Overhead**: Manipulating **innerHTML** can be less performant compared to other DOM manipulation methods, especially when dealing with large HTML strings. This is because setting **innerHTML** involves parsing and rendering the entire HTML content of the element, which can be slow for complex content structures.

**Potential Memory Leaks**: When using **innerHTML** to replace or append content repeatedly, it can lead to memory leaks in some browsers, especially in older versions of Internet Explorer.

Loss of Event Handlers and Data: When setting innerHTML to replace the content of an element, any event handlers or data associated with the existing content may be lost

Limited Control over Content: Using innerHTML to manipulate content involves overwriting the entire content of an element. If you need more fine-grained control over individual elements or attributes within the content, **innerHTML** may not be the most suitable approach.

Not Supported in XML Documents: While innerHTML is widely supported in HTML documents, it is not supported in XML documents. If you're working with XML content, you'll need to use other DOM manipulation methods.

Despite these disadvantages, innerHTML remains a powerful tool for quickly updating HTML content in certain situations. However, it's important to be aware of its limitations and potential risks, and to use it judiciously, especially when dealing with untrusted data. In many cases, using safer alternatives such as DOM manipulation methods (createElement, appendChild, etc.) or templating libraries can help mitigate these issues.