## Module (JAVASCRIPT BASIC & DOM) – 4

**Que.1 What is JavaScript?**

**Ans.** JavaScript is a high-level programming language that is commonly used for creating interactive and dynamic web content. It is primarily used for adding functionality to websites, such as validating forms, creating interactive elements, and manipulating web page content. JavaScript can be executed on both the client-side (in the user's web browser) and the server-side (on the web server). It is widely supported by modern web browsers and has a large ecosystem of libraries and frameworks that make it easier to develop complex web applications.

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

**Ans.** The isNaN function in JavaScript is used to determine whether a value is NaN (Not-a-Number) or not. It is a built-in function that takes a single argument and returns a boolean value.

The primary use of the isNaN function is to check if a value is a valid number or not. It is often used to validate user input, especially when dealing with numeric values. If the argument passed to the isNaN function is not a number or cannot be converted into a number, it will return true. Otherwise, if theargument is a valid number or can be converted into a number**,** it will return false.

Here's an example of how the isNaN function can be used:

javascript

let userInput = prompt("Enter a number:");

if (isNaN(userInput)) {

console.log("Invalid input. Please enter a valid number.");

} else {

console.log("The input is a valid number.");

}

In this example, if the user enters a non-numeric value, the isNaN function will return true, indicating that the input is invalid. If the user enters a numeric value, the isNaN function will return false, indicating that the input is valid.

**Que.3 What is negative Infinity?**

**Ans.** Negative Infinity, denoted as -Infinity, is a special value in JavaScript that represents a number that is infinitely small or negative infinity. It is the result of certain mathematical operations or calculations that yield a value that is beyond the range of representable numbers.

Negative Infinity is often used to indicate an underflow condition or when a value is smaller than the minimum representable number. It is typically encountered when dividing a negative number by zero or when subtracting apositive number from a negative number that is already very large.

Here's an example to illustrate the concept of negative Infinity:

javascript

let result = -1 / 0;

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

In this example, dividing -1 by 0 results in a value that is infinitely small or negative infinity, which is represented by -Infinity in JavaScript.

It's important to note that operations involving negative Infinity can have specific behaviors and may differ from regular arithmetic operations.

**Que.4 Which company developed JavaScript?**

**Ans.** JavaScript was developed by Netscape Communications Corporation, a now-defunct software company. The initial development of JavaScript was led by Brendan Eich in 1995. It was originally created as a scripting language for the Netscape Navigator web browser. Over time, JavaScript gained popularity and became widely adopted as a standard scripting language for web development. Today, JavaScript is supported by all major web browsers and has evolved into a versatile programming language used for both client-side and server-side development.

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

**Ans.** In JavaScript, undeclared and undefined variables are two different concepts.

**1. Undeclared Variables:**

Undeclared variables are variables that have not been declared or defined using the `var`, `let`, or `const` keywords. When you try to use an undeclared variable, JavaScript will throw a ReferenceError, indicating that the variable is not defined.

Here's an example of an undeclared variable:

javascript

console.log(x); // ReferenceError: x is not defined

In this example, the variable `x` has not been declared or assigned a value before trying to access it, resulting in a ReferenceError.

**2. Undefined Variables:**

Undefined variables, on the other hand, are variables that have been declared but have not been assigned a value. When you declare a variable without assigning a value to it, JavaScript automatically initializes it with the value `undefined`.

Here's an example of an undefined variable:

javascript

let y;

console.log(y); // Output: undefined

In this example, the variable `y` has been declared using the `let` keyword, but it has not been assigned a value. As a result, when you try to access it, it will have the value `undefined`.

It's important to note that `undefined` is a special value in JavaScript that represents the absence of a value or the uninitialized state of a variable.

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

**Ans**. Certainly! Here's an example of JavaScript code that adds new elements dynamically to an HTML document:

javascript

// Create a new element

const newElement = document.createElement("div");

// Set attributes or properties for the new element

newElement.textContent = "This is a new element";

newElement.classList.add("new-element");

// Find the parent element where you want to append the new element

const parentElement = document.getElementById("parent-element");

// Append the new element to the parent element

parentElement.appendChild(newElement);

In this example, we create a new `<div>` element using the `document.createElement()` method. We then set attributes or properties for the new element, such as its text content and CSS class, using the `textContent` and `classList` properties.

Next, we find the parent element where we want to append the new element. In this example, we assume there is an element with the id "parent-element" in the HTML document. You can modify this to match the id of your desired parent element.

Finally, we use the `appendChild()` method to append the new element to the parent element, making it a child of the parent element.

You can repeat this code or modify it as needed to add multiple elements dynamically or to add different types of elements (e.g., `<p>`, `<span>`, etc.).

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

**Ans.** ViewState and SessionState are both state management techniques used in web applications, but they serve different purposes and have distinct characteristics.

ViewState:

- ViewState is a client-side state management technique in ASP.NET.

- It is used to store and manage the state of individual web controls or the entire page across postbacks.

- ViewState data is stored as a hidden field in the HTML markup of the page and is sent back and forth between the server and the client.

- ViewState is specific to a single user and is not shared between different users or sessions.

- It is primarily used to maintain the state of controls, such as their values, visibility, and other properties, during postbacks.

- ViewState is suitable for storing small amounts of data, but it can increase the size of the page, leading to slower performance and increased bandwidth usage.

SessionState:

- SessionState is a server-side state management technique in ASP.NET.

- It is used to store and manage user-specific data across multiple requests and pages within a session.

- SessionState data is stored on the server and is associated with a unique session identifier (usually stored in a cookie or URL).

- SessionState is shared between different pages and requests made by the same user within a session.

- It can store larger amounts of data compared to ViewState, making it suitable for storing user-specific information, such as user preferences, shopping cart contents, or authentication details.

- SessionState can be configured to use different storage mechanisms, such as in-memory, database, or out-of-process storage, depending on the application's requirements.

In summary, ViewState is used to maintain the state of individual controls or the entire page during postbacks, while SessionState is used to store and manage user-specific data across multiple requests and pages within a session. ViewState is client-side and specific to a single user, while SessionState is server-side and shared between pages and requests within a session.

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

**Ans.** The === operator in JavaScript is a strict equality operator. It is used to compare two values for equality, both in terms of value and data type.

When using the === operator, JavaScript checks if the operands are of the same type and have the same value. If both the type and value are identical, the operator returns true. Otherwise, it returns false.

Here's an example to demonstrate the usage of the === operator:

Javascript

let num1 = 5;

let num2 = "5";

console.log(num1 === num2); // Output: false

In this example, num1 is a number (data type: number) and num2 is a string (data type: string) that represents the number 5. Even though their values are the same, the === operator returns false because they are of different data types.

By using the === operator, you can perform strict equality checks in JavaScript, ensuring that both the value and data type are considered during the comparison.

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

**Ans.** To change the style or class of an element in JavaScript, you can use the DOM (Document Object Model) API. Here are a few common approaches:

**1. Changing inline styles:**

You can directly modify the inline styles of an element using the `style` property. For example, to change the background color of an element with the id "myElement" to red, you can do:

javascript

let element = document.getElementById("myElement");

element.style.backgroundColor = "red";

**2. Modifying class names:**

You can add, remove, or toggle classes on an element using the `classList` property. For example, to add a class "highlight" to an element with the id "myElement", you can do:

javascript

let element = document.getElementById("myElement");

element.classList.add("highlight");

To remove a class, you can use `remove` instead of `add`:

javascript

element.classList.remove("highlight");

To toggle a class (add if it doesn't exist, remove if it exists), you can use `toggle`:

javascript

element.classList.toggle("highlight");

**3. Modifying element attributes:**

You can also directly modify the attributes of an element using the `setAttribute` method. For example, to change the"src" attribute of an image element with the id "myImage", you can do:

javascript

let image = document.getElementById("myImage");

image.setAttribute("src", "new-image.jpg");

These are just a few examples of how you can change the style or class of an element using JavaScript. The specific approach you choose depends on your requirements and the structure of your HTML document.

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

**Ans**. In JavaScript, file reading and writing operations are typically performed in the context of web development, where JavaScript runs in a web browser. The File API and the FileReader API provide the necessary functionality to read files from the user's local system and perform file-related operations. However, it's important to note that these APIs have certain limitations due to security restrictions imposed by web browsers.

To read a file using JavaScript, you can follow these steps:

1. Create an HTML input element of type "file" in your web page to allow the user to select a file:

html

<input type="file" id="fileInput">

2. Add an event listener to the input element to handle the file selection:

javascript

const fileInput = document.getElementById('fileInput');

fileInput.addEventListener('change', handleFileSelect, false);

3. Implement the `handleFileSelect` function to read the contents of the selected file:

javascript

function handleFileSelect(event) {

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

const reader = new FileReader();

reader.onload = function(event) {

const fileContent = event.target.result;

console.log(fileContent); // Do something with the file content

};

reader.readAsText(file);

}

In this example, the `handleFileSelect` function retrieves the selected file using `event.target.files[0]`. It then creates a `FileReader` object and sets its `onload` event handler to process the file content once it is loaded. Finally, the `readAsText` method is called on the `FileReader` object to read the file as text.

To write a file using JavaScript, it is not directly possible within the browser environment due to security restrictions. However, if you are working in a server-side JavaScript environment (e.g., Node.js), you can use the built-in `fs` module to perform file writing operations. Here's a basic example:

javascript

const fs = require('fs');

const content = 'This is the content to be written to the file.';

fs.writeFile('example.txt', content, function(err) {

if (err) {

console.error(err);

} else {

console.log('File written successfully.');

}

});

In this example, the `writeFile` function from the `fs` module is used to write the specified content to a file named "example.txt". The callback function is invoked once the operation is complete, indicating whether an error occurred or the file was written successfully.

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

**Ans.** In JavaScript, there are three main looping structures: "for" loops, "while" loops, and "do-while" loops.

1. "for" loops: These loops are commonly used when you know the number of iterations in advance. They consist of three parts: initialization, condition, and increment/decrement. The loop will continue executing as long as the condition is true.

Example:

javascript

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

// code to be executed

}

2. "while" loops: These loops are used when the number of iterations is not known in advance. The loop will continue executing as long as the specified condition is true.

Example:

javascript

let i = 0;

while (i < 5) {

// code to be executed

i++;

}

3. "do-while" loops: Similar to "while" loops, but the condition is checked after the loop body is executed. This guarantees that the loop will execute at least once, even if the condition is initially false.

Example:

javascript

let i = 0;

do {

// code to be executed

i++;

} while (i < 5);

These looping structures allow you to repeat a block of code multiple times, providing flexibility in handling different scenarios.

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

**Ans.** In JavaScript, you can convert a string of any base to an integer using the `parseInt()` function. The `parseInt()` function takes two arguments: the string to be converted and the base of the number system.

Here's an example of converting a string of base 2 (binary) to an integer:

javascript

const binaryString = "101010";

const decimalNumber = parseInt(binaryString, 2);

console.log(decimalNumber); // Output: 42

In the example above, the `parseInt()` function is used to convert the `binaryString` to a decimal number by specifying the base as 2. The resulting decimal number is stored in the `decimalNumber` variable.

Similarly, you can convert strings of other bases, such as octal (base 8) or hexadecimal (base 16), by specifying the appropriate base in the `parseInt()` function.

Note that if the string cannot be parsed as a valid number in the specified base, the `parseInt()` function will return `NaN` (Not a Number).

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

**Ans.** The `delete` operator in JavaScript is used to remove a property from an object or to delete an element from an array. Its function depends on the context in which it is used:

1. Deleting object properties:

javascript

const obj = { name: "balvant", age: 21 };

delete obj.age;

console.log(obj); // Output: { name: "balvant" }

In this example, the `delete` operator is used to remove the `age` property from the `obj` object.

2. Deleting array elements:

javascript

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

delete arr[2];

console.log(arr); // Output: [1, 2, empty, 4, 5]

Here, the `delete` operator is used to delete the element at index 2 in the `arr` array. However, note that the array length remains the same, and the deleted element is replaced with an `empty` slot.

It's important to note that the `delete` operator does not directly free up memory or affect variables. It only removes the reference to the property or element, making it eligible for garbage collection if there are no other references to it.

**Que.14 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 displaying messages or gathering user input: "alert", "confirm", and "prompt".

1. "alert" pop-up box: It is used to display a message to the user. It only contains an "OK" button and does not allow any user input.

Example:

javascript

alert("This is an alert message.");

2. "confirm" pop-up box: It is used to display a message and prompt the user to confirm or cancel an action. It contains "OK" and "Cancel" buttons.

Example:

javascript

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

if (result) {

// Code to delete the item

} else {

// Code to cancel the deletion

}

3. "prompt" pop-up box: It is used to display a message and prompt the user to enter some input. It contains an input field for the user to enter text and "OK" and "Cancel" buttons.

Example:

javascript

const balvant = prompt("balvant:");

if (name) {

console.log("Hello, " + balvant + "!");

} else {

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

}

These pop-up boxes provide a way to interact with the user and gather information or confirm actions within a JavaScript application.

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

**Ans.** In JavaScript, `void(0)` is often used as a way to prevent the browser from navigating to a new page when clicking on a hyperlink or button.

When `void` is used with the value `0`, it creates an expression that evaluates to `undefined`. By using `void(0)` as the value of an `href` attribute or an `onclick` event handler, it essentially tells the browser to do nothing when the associated element is interacted with.

Here's an example of using `void(0)` in an HTML anchor tag:

html

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

In this example, clicking on the anchor tag will not cause the browser to navigate to a new page because the `href` attribute is set to `javascript:void(0)`. It effectively prevents the default behavior of a hyperlink.

It's worth noting that with modern JavaScript and best practices, the use of `void(0)` is less common. Instead, event listeners and JavaScript functions are typically used to handle interactions and prevent default actions.

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

**Ans.** In JavaScript, you can force a page to load another page by using the `window.location` object. There are a few different ways to achieve this:

1. Using `window.location.href`:

javascript

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

This will set the `href` property of `window.location` to the desired URL, causing the browser to navigate to that page.

2. Using `window.location.replace()`:

javascript

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

This method replaces the current page in the browser's history with the new page, effectively redirecting to the specified URL.

3. Using `window.location.assign()`:

javascript

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

This method also navigates to the specified URL, similar to `window.location.href`, but it allows the user to navigate back to the original page using the browser's back button.

All three methods will trigger the browser to load the new page specified by the URL. Choose the method that best suits your requirements based on whether you want to replace the current page in the history or allow the user to navigate back.

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

**Ans.** While the `innerHTML` property in JavaScript provides a convenient way to manipulate HTML content, it does have some disadvantages:

1. Security risks: Using `innerHTML` to dynamically insert or update HTML content can expose your application to security vulnerabilities like cross-site scripting (XSS) attacks. If the content being inserted contains user-generated or untrusted data, it can execute malicious scripts unintentionally.

2. Performance impact: When you use `innerHTML` to modify HTML content, the entire HTML structure within the targeted element is re-parsed and re-rendered. This can be inefficient, especially when dealing with large or complex HTML structures. In contrast, more targeted DOM manipulation methods like `createElement` and `appendChild` can be faster and more efficient.

3. Event listener detachment: If you use `innerHTML` to replace or modify HTML content that has attached event listeners, those event listeners will be detached. This means you'll need to reattach the event listeners manually after using `innerHTML`, which can be cumbersome and error-prone.

4. Limited control over DOM structure: When using `innerHTML`, you're essentially replacing the entire content of an element with a new HTML string. This can lead to unintended consequences if you need fine-grained control over the existing DOM structure or if you want to preserve any existing event listeners or data associated with the elements.

To mitigate these disadvantages, it is often recommended to use alternative approaches like DOM manipulation methods (`createElement`, `appendChild`, etc.), or libraries/frameworks that provide more secure and efficient ways to update HTML content, such as React or Vue.js.