**Module 4 – JavaScript Basic and DOM**

**Q.1 What is JavaScript?**

JavaScript is a lightweight but incredibly powerful scripting language. We most frequently encounter it through our browsers, but JavaScript has snuck into everything from native applications to PDFs to e-books. Even web servers themselves can be powered by JavaScript.

JavaScript is a programming language that adds interactivity and custom behaviors to our sites. It is a client-side scripting language, which means it runs on the user’s machine and not on the server.

As a dynamic programming language, JavaScript doesn’t need to be run through any form of compiler that interprets our human-readable code into something the browser can understand. The browser effectively reads the code the same way we do and interprets it on the fly.

JavaScript is also loosely typed. All this means is that we don’t necessarily have to tell JavaScript what a variable is. If we’re setting a variable to a value of 5, we don’t have to programmatically specify that variable as a number; that is, 5 is a number, and JavaScript recognizes it as such.

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

`isNaN () ` in JavaScript is used to check if a value is NaN (Not-a-Number). It returns `true` if the value is NaN or cannot be converted to a number; otherwise, it returns `false`. It's often used to validate numeric input or check the result of mathematical operations. However, for more accurate NaN checks, it's recommended to use `Number.isNaN() ` to avoid implicit type conversion.

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

In JavaScript, negative infinity (`-Infinity`) is a special value representing a quantity smaller than any real number. It's used to denote results of operations that tend toward negative infinity or to indicate an overflow condition. For example, `Number.NEGATIVE\_INFINITY` is a representation of negative infinity in JavaScript.

In JavaScript, negative infinity is a special value representing negative infinity, which is a mathematical concept indicating a quantity that is smaller than any real number. It's denoted by the keyword `Number.NEGATIVE\_INFINITY`.

let negativeInfinity = Number.NEGATIVE\_INFINITY;

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

console.log (negativeInfinity \* 10); // Output: -Infinity

It's worth noting that operations involving negative infinity in JavaScript will generally result in negative infinity.

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

Despite its name, JavaScript has nothing to do with Java. It was created by Brendan Eich at Netscape in 1995 and originally named “LiveScript.” But Java was all the rage around that time, so for the sake of marketing, “LiveScript” became “JavaScript.”

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

**1. Undeclared Variables:**

- Variables that are used in code without being formally declared using `var`, `let`, or `const`.

- Accessing an undeclared variable may result in an error in strict mode or create an implicit global variable in non-strict mode.

**2. Undefined Variables:**

- Variables that have been declared but not assigned a value.

- All declared variables are initialized with the value `undefined` until a value is explicitly assigned.

// Undeclared variable

console.log(undeclaredVariable); // May throw an error in strict mode

// Declared but undefined variable

let undefinedVariable;

console.log(undefinedVariable); // Outputs: undefined

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

To dynamically add new elements using JavaScript, we can follow these steps:

**1. Access the Container:**

- Get a reference to the container where you want to add new elements.

var container = document. getElementById("container");

**2. Create a New Element:**

- Create a new HTML element using `document. createElement ()`.

var newElement = document. createElement("p");

**3. Set Content or Attributes:**

- Set content or attributes for the new element.

newElement.textContent = "This is a new paragraph.”;

**4. Append to the Container:**

- Append the new element to the container using `appendChild ()`.

container. appendChild(newElement);

**Putting it all together:**

var container = document. getElementById("container");

var newElement = document. createElement("p");

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

container. appendChild(newElement);

This script dynamically creates a new paragraph element, sets its content, and appends it to a container in your HTML. We can customize this code to create and add different types of elements based on needs.

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

**1. ViewState:**

- Scope: Associated with individual web pages.

- Storage: Maintains the state of controls on a single page across postbacks (subsequent requests).

- Location: Stored on the client side, usually in a hidden field within the HTML.

- Purpose: Helps retain the state of controls and data on the same page during roundtrips.

**2. SessionState:**

- Scope: Associated with the user's session.

- Storage: Maintains user-specific data across multiple pages during a user's visit to the application.

- Location: Stored on the server or out-of-process (e.g., in a database) depending on the configuration.

- Purpose: Useful for storing information that needs to persist across multiple requests within a user's session.

In short, ViewState is concerned with maintaining the state of individual pages across postbacks, often dealing with control states. SessionState, on the other hand, is focused on maintaining data across multiple pages and requests during a user's session. Both are mechanisms to manage state in a web application, but they serve different purposes and have different scopes.

**Q.8 What is “===” operator?**

The "===" operator in JavaScript is the strict equality operator. It checks for both value and type equality without performing type coercion. This means that the operands must be of the same type to be considered equal, and no automatic type conversion is done.

In contrast, the "==" operator is the loose equality operator, which performs type coercion if the operands are of different types, potentially leading to unexpected results.

Example with "===":

3 === "3" // false (number is not equal to string)

Example with "==":

3 == "3" // true (after type coercion, the values are equal)

Using "===" is generally recommended for more predictable and less error-prone code, as it avoids implicit type conversions.

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

To change the style or class of an element in JavaScript, we can use the following methods:

**1. Changing Style:**

- Using the `style` property of the element to directly manipulate its inline styles.

// Get the element by its ID

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

// Change the background color

element.style.backgroundColor = "red";

// Change the font size

element.style.fontSize = "16px";

**2. Adding/Removing Classes:**

- Using the `classList` property to manipulate the classes of an element.

// Get the element by its ID  
 var element = document.getElementById("myElement");

// Add a class

element.classList.add("newClass");

// Remove a class

element.classList.remove("oldClass");

- Alternatively, we can use `className` to directly set the entire class attribute.

// Get the element by its ID

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

// Set the class attribute

element.className = "newClass";

These methods allow us to dynamically change the style or class of HTML elements using JavaScript, making web pages more interactive and responsive.

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

Sure, let's break it down without diving into code:

**Reading a File:**

- In a browser, use the FileReader API.

- Then, creating an input element of type "file" to allow users to choose a file.

- After that, add an eventListener for the "change" event on the input element.

- Access the selected file and use FileReader to read its contents.

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

<script>

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

fileInput.addEventListener("change", (event) => {

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

if (file) {

const reader = new FileReader();

reader.onload = (e) => {

const contents = e.target.result;

console.log("File contents:", contents);

};

reader.readAsText(file);

}

});

</script>

**Writing to a File (This has to be done using Node.js(On server-end)):**

- In a Node.js environment, using the built-in fs module.

- Use functions like `writeFile` or `writeFileSync` to write content to a file.

- Specifying the file path and the content we want to write.

const fs = require("fs");

const contentToWrite = "Hello, this is content to write to a file.";

fs.writeFileSync("output.txt", contentToWrite);

console.log("File written successfully!");

Reading and Writing files involve asynchronous operations, so handling callbacks or promises is common. In the browser, security restrictions apply to file access for client-side JavaScript. In Node.js, we have more direct access to the file system.

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

JavaScript supports several looping structures:

**1. for Loop:**

- Executes a block of code a specified number of times

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

// Code to be executed

}

**2. while Loop:**

- Repeats a block of code as long as a specified condition is true.

while (condition) {

// Code to be executed

}

**3. do...while Loop:**

- Similar to the while loop, but the code block is executed at least once before the condition is checked.

do {

// Code to be executed

} while condition);

**4. for...in Loop:**

- Iterates over the enumerable properties of an object.

for (let key in object) {

// Code to be executed

}

**5. for...of Loop:**

- Iterates over the values of an iterable object (e.g., arrays, strings).

for (let value of iterable) {

// Code to be executed

}

**6. forEach Method:**

- Applies a function to each element of an array.

array. forEach(function(element) {

// Code to be executed for each element

});

These looping structures provide flexibility for different scenarios. The choice of loop depends on the specific requirements and the type of data you are working with.

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

To convert a string in any base to an integer in JavaScript, we can use the `parseInt()` function. The `parseInt()` function takes two arguments: the string to be converted, and the base of the numeral system (radix). The radix specifies the base of the numeral system (e.g., 10 for decimal, 16 for hexadecimal).

In short, use **`parseInt(string, radix)`** to convert a string in any base to an integer in JavaScript.

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

In JavaScript, the `delete` operator is used to remove properties from objects or elements from arrays. It can also be used to delete variables. When applied:

- For objects, it removes property.

- For arrays, it removes an element but leaves a hole.

- For variables, it deletes variables declared with `var`, `let`, or `const`.

Keep in mind that it doesn't work on variables declared without `var`, `let`, or `const`. Also, it might not delete non-configurable properties, and in the case of arrays, it doesn't re-index remaining elements. It returns `true` if the deletion is successful, `false` otherwise.

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

In JavaScript, there are three types of popup boxes commonly used for interactive user messages:

**1. Alert Box (`alert`):**

- Displays a simple message box with a specified message and an OK button.

- Used for providing information to the user.

alert("This is an alert!");

**2. Confirm Box (`confirm`):**

- Displays a dialog box with a specified message, OK, and Cancel buttons.

- Returns `true` if the user clicks OK and `false` if Cancel.

var result = confirm("Do you want to proceed?");

**3. Prompt Box (`prompt`):**

- Displays a dialog box with a specified message, an input field, OK, and Cancel buttons.

- Returns the entered text if the user clicks OK, and `null` if Cancel.

var userInput = prompt("Please enter your name:", "John Doe");

These popup boxes are simple ways to interact with users and gather information or confirm actions on a webpage. They are, however, somewhat limited in terms of customization and styling.

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

In JavaScript, `void(0)` is often used as a concise way to create an expression that evaluates to `undefined`. The `void` operator in JavaScript takes an operand and evaluates it to `undefined`.

Using `void(0)` is a common idiom when we want to create a hyperlink or a button that doesn't perform any action when clicked, and we want to avoid changing the current page or causing any unintended behavior.

For example, in an HTML link:

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

In this case, clicking the link triggers the `myFunction()` JavaScript function without navigating to a new page, thanks to the use of `void(0)` in the `href`.

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

In JavaScript, we can force a page to load another page by changing the `window.location` property. This property represents the current URL of the browser, and modifying it causes the browser to navigate to the specified URL.

Here's a simple example:

// Force the page to load another page

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

This line of code sets the `href` property of `window.location` to the desired URL, triggering a page load to that URL. We can use this approach in response to user actions, events, or as part of your application's logic to navigate to different pages dynamically.

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

Using `innerHTML` in JavaScript has drawbacks:

**1. Security Risk:** It can expose your application to XSS if user input is not sanitized.

**2. Performance Impact:** Less efficient for large content due to parsing and rendering.

**3. Event Handlers Removal:** Replacing content with `innerHTML` can remove existing event handlers.

**4. Limited Control:** Offers less control over the DOM compared to other methods.

**5. Markup Issues:** Improperly formatted HTML may lead to unexpected results.

**Advance JavaScript : Module 4 – New Request**

**Q.1 What is JSON?**

JSON (JavaScript Object Notation) is a lightweight data interchange format that is easy for humans to read and write, and easy for machines to parse and generate. It uses a simple text format to represent data structures, primarily based on key-value pairs. JSON is widely used for data transmission between a server and a web application, as well as for configuration files and data storage. It closely resembles the syntax of JavaScript object literals but is language-independent, making it a common choice for exchanging data between different programming languages.

**Q.2 What is promises?**

Promises in JavaScript are a way to handle asynchronous operations more cleanly. They represent the result of an asynchronous task, which can be either a successful value or an error. Promises have states (pending, fulfilled, or rejected) and provide a structured approach with methods like `then` and `catch` for handling the outcomes of asynchronous tasks.

**Q.3 What is JavaScript Output method?**

In JavaScript, the primary methods for outputting information are `console.log ()` and `document.write()`. `console.log ()` is commonly used for debugging and logging messages to the browser's console, while `document.write()` is used to write directly to the HTML document. Both are essential tools for displaying information and interacting with users in JavaScript.

**Q.4 How to used JavaScript Output method?**

In JavaScript, you can use various methods to output information. Here are two commonly used methods: `console.log()` and `document.write()`.

**1. console.log() method:**

- Used for printing messages and values to the browser's console.

- Helpful for debugging and development purposes.

var message = "Hello, World!";

console.log(message);

**2. document.write() method:**

- Writes content directly to the HTML document.

- Be cautious when using it, as it can overwrite the entire document if called after the document has fully loaded.

document.write("This text will be written to the document.");

**Q.5 How to use JavaScript Events to do all examples?**

In JavaScript, events are used to respond to user interactions or other occurrences on a webpage. Here's a brief overview of how we can use JavaScript events for the examples:

**1. Using JavaScript events with `console.log()`:**

- Creating an HTML element (e.g., a button) with a specific ID.

- Using JavaScript to add an event listener to that element, specifying the event we want to listen for (e.g., "click").

- In the event listener, writing a function that includes the code we want to execute (e.g., logging a message to the console).

document.getElementById("buttonId").addEventListener("click", function() {

console.log("Button clicked!");

});

**2. Using JavaScript events with the DOM to modify content:**

- Creating HTML elements (e.g., a paragraph and a button) with specific IDs.

- Using JavaScript to add an event listener to the button, listening for a relevant event (e.g., "click").

- In the event listener, writing a function that modifies the content of an HTML element (e.g., using `innerHTML`)

document.getElementById("buttonId").addEventListener("click", function() {

var newMessage = "New content added!";

document.getElementById("outputId").innerHTML = newMessage;

});

These event-driven approaches allow us to create interactive and dynamic web pages using JavaScript.