6/28/2023

Module JAVASCRIPT BASIC & DOM – 4

**Vijay Ratnotar**

JAVASCRIPT Assignment - VijayRatnotar

TOPS Technologies

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

**Name:-** **Vijay Ratnotar Mo.No.9429099686**

1. **What is JavaScript?**

JavaScript is a popular computer language for producing interactive and dynamic online content. It enables website developers to add features such as form validation, animations, and HTML element manipulation. JavaScript is an interpreted high-level programming language that is mostly used to create interactive and dynamic online content. Netscape created it, and it is currently standardised under the ECMAScript specification.

JavaScript is client-side, which means it executes in the user's web browser rather than on the server. It is a flexible language that most recent web browsers support.

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

the 'isNaN()' function determines if a value is NaN (Not a Number) or not. It accepts a single parameter and returns 'true' if it is NaN and 'false' otherwise. This function is useful when you need to determine whether a value is a legitimate number before conducting any mathematical operations on it. You may use 'isNaN()' to check user input in a form, for example, to confirm that a numeric number is inputted.

1. **What is negative Infinity?**

A number that is endlessly tiny, or, to put it another way, a number that is smaller than any other number, is represented by the special value known as negative infinity in JavaScript. The constant "Number.NEGATIVE\_INFINITY" is used to identify it.

1. **Which company developed JavaScript?**

In the middle of the 1990s, Brendan Eich created JavaScript while working for Netscape Communications Corporation. When Brendan Eich was working at Netscape in May 1995, he developed JavaScript in about 10 days. The original name of the language was "Mocha," which was then modified to "LiveScript" and eventually to "JavaScript" to capitalise on Java's then-current level of popularity.

JavaScript was created by the now-defunct software business Netscape Communications Corporation. Brendan Eich oversaw the original development of JavaScript in 1995.

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

Undeclared variables are variables that have not been declared using the `var`, `let`, or `const` keywords. They are not recognized by the JavaScript interpreter and will result in a reference error if you try to use them.

Undefined variables, on the other hand, are variables that have been declared but have not been assigned a value. They have a default value of `undefined` until a value is assigned to them.

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

Here is an example of code for adding new elements dynamically in JavaScript:

```javascript

// Create a new element

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

// Set some properties for the new element

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

newElement.style.color = "red";

// Append the new element to an existing element in the DOM

var parentElement = document.getElementById("parent");

parentElement.appendChild(newElement);

To add new elements dynamically, you can use JavaScript to manipulate the DOM (Document Object Model) and add elements to your HTML page. Here's an example code snippet that demonstrates how to add a new paragraph element dynamically:

HTML:

```html

<!DOCTYPE html>

<html>

<head>

<title>Add Elements Dynamically</title>

</head>

<body>

<button id="addButton">Add New Paragraph</button>

<div id="container"></div>

<script src="script.js"></script>

</body>

</html>

```

JavaScript (script.js):

javascript

// Get the button and container elements

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

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

// Add event listener to the button

addButton.addEventListener("click", addNewParagraph);

// Function to add a new paragraph

function addNewParagraph() {

// Create a new paragraph element

const newParagraph = document.createElement("p");

// Set the text content of the paragraph

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

// Append the new paragraph to the container

container.appendChild(newParagraph);

}

```

In this example, a button with the id "addButton" and a container div with the id "container" are defined in the HTML. The JavaScript code adds an event listener to the button, and when clicked, it calls the `addNewParagraph` function.

The `addNewParagraph` function creates a new `<p>` element using the `createElement` method, sets its text content using the `textContent` property, and appends it to the container using the `appendChild` method.

1. **What is the difference between View State and Session State?**

Web development, notably in the context of ASP.NET, is connected to the ideas of view state and session state.

View State describes a web page's or control's state that is maintained through postbacks. The values of controls on a web page may be saved and retrieved using it, allowing them to keep their state even after a postback.

Session State, on the other hand, refers to the state of a user's session on a website. It is used to store and retrieve user-specific data throughout multiple requests and pages. Session State is typically stored on the server and can be accessed by the user during their session.

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

In JavaScript, the `=` operator is used for assignment. It assigns the value on the right-hand side of the operator to the variable on the left-hand side. For example:

```javascript

var x = 5;

```

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

To change the style or class of an element in JavaScript, you can use the `classList` property and the `style` property.

To change the class of an element, you can use the `classList` property's methods such as `add()`, `remove()`, or `toggle()`. For example, to add a class to an element with the id "myElement", you can use the following code:

```javascript

document.getElementById("myElement").classList.add("newClass");

```

To change the style of an element, you can access the `style` property of the element and modify its CSS properties. For example, to change the background color of an element with the id "myElement", you can use the following code:

```javascript

document.getElementById("myElement").style.backgroundColor = "red";

```

Remember to replace "myElement" with the actual id or select the element using other methods like `querySelector()`.

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

Using the JavaScript File System API, you may read and write files. However, security limitations prevent web browsers from using the File System API. It is mostly offered in server-side JavaScript frameworks like Node.js.

Here's an example of how you can read and write a file using Node.js:

1. First, make sure you have Node.js installed on your computer.

2. Create a new JavaScript file (e.g., `fileReadWrite.js`) and open it in a text editor.

3. Use the `fs` module to read and write files. Here's an example code snippet:

```javascript

const fs = require('fs');

// Read a file

fs.readFile('input.txt', 'utf8', (error, data) => {

if (error) {

console.error(error);

return;

}

console.log(data);

});

// Write to a file

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

fs.writeFile('output.txt', content, 'utf8', (error) => {

if (error) {

console.error(error);

return;

}

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

});

```

In this example, we import the `fs` module, which provides file system-related functionality in Node.js.

The `readFile` function reads the contents of a file asynchronously. It takes the file path, the file encoding (e.g., 'utf8' for plain text files), and a callback function as parameters. The callback function is called with two arguments: an error (if any) and the file data.

The `writeFile` function writes data to a file asynchronously. It takes the file path, the data to write, the file encoding, and a callback function as parameters. The callback function is called with an error (if any) as an argument.

To run this script, open a terminal or command prompt, navigate to the directory where the script is located, and run the following command:

```

node fileReadWrite.js

```

Make sure you have an `input.txt` file in the same directory as the script to read from, and an `output.txt` file will be created or overwritten with the content specified in the `writeFile` function.

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

There are several looping structures in JavaScript:

1. for loop: It executes a block of code a specified number of times.

Example:

```

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

console.log(i);

}

```

2. while loop: It executes a block of code as long as the specified condition is true.

Example:

```

let i = 0;

while (i < 5) {

console.log(i);

i++;

}

```

3. do...while loop: It executes a block of code once, and then repeats the loop as long as the specified condition is true.

Example:

```

let i = 0;

do {

console.log(i);

i++;

} while (i < 5);

```

4. for...in loop: It iterates over the properties of an object.

Example:

```

const obj = { a: 1, b: 2, c: 3 };

for (let key in obj) {

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

}

```

5. for...of loop: It iterates over iterable objects (arrays, strings, etc.) and provides access to each element.

Example:

```

const arr = [1, 2, 3];

for (let element of arr) {

console.log(element);

}

```

6. forEach loop: It is a method available on arrays that executes a provided function once for each element in the array.

Example:

```

const arr = [1, 2, 3];

arr.forEach(function(element) {

console.log(element);

});

```

These looping structures provide different ways to iterate over data and perform repetitive tasks in JavaScript.

1. **How can you convert the string of any base to an integer 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 string. The base can be any integer between 2 and 36.

Here's an example of converting a binary string to an integer:

Javascript

const binaryString = "101010";

const decimalNumber = parseInt(binaryString, 2);

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

In this example, the binary string "101010" is converted to an integer using the base 2, which represents binary numbers. The resulting decimal number is 42.

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

Javascript

const octalString = "52";

const decimalNumber = parseInt(octalString, 8);

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

const hexString = "2A";

const decimalNumber = parseInt(hexString, 16);

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

In these examples, the octal string "52" is converted to decimal 42 using base 8, and the hexadecimal string "2A" is converted to decimal 42 using base 16.

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

The 'delete' operator in JavaScript is used to remove a property from an object. It may also be used to remove an element from an array, however this is often not advised.

The 'delete' operator requires an object and a property name as its operands when it is used with an object. Here is how to use it:

Javascript

delete objectName.propertyName;

Or

javascript

delete objectName['propertyName'];

The `delete` operator will remove the specified property from the object if it exists. After the deletion, if you try to access the property, it will return `undefined`.

Here's an example:

Javascript

const person = {

name: 'VIJAY',

age: 30,

};

console.log(person.name); // Output: 'VIJAY'

delete person.name;

console.log(person.name); // Output: undefined

It's important to note that the `delete` operator only removes the property itself, not any associated values or functions. It doesn't affect variables or functions declared with the `var`, `let`, or `const` keywords.

As mentioned earlier, using the `delete` operator to remove elements from an array is not recommended. It can leave empty slots in the array, which may cause unexpected behavior when iterating or manipulating the array. Instead, it's better to use array methods like `splice()` or reassign array elements to `undefined` if you want to indicate an empty slot.

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

JavaScript provides three types of pop-up boxes: alert(), prompt(), and confirm().

The alert() box displays a message to the user, with an "OK" button to close the box. Example:

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

The prompt() box displays a message to the user and allows them to input data. It returns the value entered by the user. Example:

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

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

The confirm() box displays a message to the user and provides two buttons - "OK" and "Cancel". It returns a boolean value indicating whether the user clicked "OK" or "Cancel". Example:

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

if (result) {

console.log("Item deleted!");

} else {

console.log("Deletion cancelled.");

}

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

In JavaScript, void(0) is a special expression that evaluates to undefined. It is often used in HTML anchor tags to prevent the default behavior of navigating to a new page when clicked.

By using href="javascript:void(0)", the link appears as inactive or disabled, allowing developers to attach custom event handlers without triggering any unwanted behavior.

For example, consider an HTML anchor tag <a> with a href attribute:

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

When this anchor is clicked, the browser will execute the JavaScript expression void(0), which evaluates to undefined. Since undefined doesn't trigger any navigation, it effectively prevents the page from reloading or changing.

Similarly, it can be used in form submissions to prevent the default form behavior, like this:

<form action="javascript:void(0)">

<!-- form elements here -->

<button type="submit">Submit</button>

</form>

In this case, when the form is submitted, the JavaScript expression void(0) is evaluated, resulting in undefined, which prevents the form from being submitted and the page from refreshing.

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

In JavaScript, you can use the `window.location` object to change the current page's URL, effectively loading a new page. There are a few different ways to achieve this:

1. Assign a new URL directly to `window.location.href`:

javascript

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

2. Use `window.location.assign()` to load a new page:

javascript

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

3. Simulate a click on an anchor element using its `click()` method:

javascript

document.getElementById('myLink').click(); // Assuming you have an anchor element with the ID 'myLink'

4. Use `window.location.replace()` to replace the current page with a new one:

javascript

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

Please be aware that altering the website in this way with JavaScript is often seen as a redirect and can ruin the user experience if done without the user's knowledge or agreement. It's typically advised to employ these strategies sparingly and carefully to prevent annoying or confused consumers.

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

There are several disadvantages of using innerHTML in JavaScript:

1. Risks to your website's security include cross-site scripting (XSS) attacks if you use innerHTML. The security of your users might be jeopardised if you are careless with the data you enter and allow an attacker to insert harmful code into your website.

2. Performance issues: Changing an element's innerHTML forces the browser to parse and re-render that element's whole content, which can be computationally expensive and cause your webpage to load slowly, especially when working with big or sophisticated DOM structures.

3. Loss of event listeners: Any event listeners affixed to an element's innerHTML are lost when the element is modified. After changing the innerHTML, you would need to reconnect event listeners if you needed to preserve them.

4. Limited control over DOM structure: innerHTML replaces the entire content of an element, so if you only want to modify a small portion of the DOM, you would need to recreate the entire structure within the innerHTML, which can be cumbersome and error-prone.

5. Accessibility problems: Altering the innerHTML might affect how accessible your website is. People with disabilities may encounter a bad user experience as a result of the modifications performed by innerHTML if screen readers and other assistive devices are unable to correctly comprehend or announce them.

6. Code maintainability: When working with complicated HTML structures, using innerHTML might make it more difficult to maintain and debug your code. It makes it challenging to separate issues and adhere to recommended practises for code organisation because it combines JavaScript and HTML code.

Overall, even while innerHTML might be useful for straightforward tasks, it is typically advised to utilise alternate techniques, such as DOM manipulation or template engines, to avoid the drawbacks listed above.