***Frontend Assignment***

***MODULE: 1 (*JAVASCRIPT BASIC & DOM*)***

1. What is JavaScript?

JavaScript is a popular programming language that is primarily used for creating dynamic and interactive web pages. It was initially developed by Brendan Eich at Netscape Communications in 1995 and has since become one of the fundamental languages for web development.

JavaScript is a client-side scripting language, which means it runs directly in the web browser of the user. It enables web developers to add interactive elements, perform actions, and manipulate the content of web pages in response to user interactions. With JavaScript, you can create features like image sliders, form validations, interactive maps, and much more.

Over the years, JavaScript has evolved to become a versatile language that can be used not only in web browsers but also on server-side platforms (Node.js) and for developing mobile and desktop applications. It has a wide range of frameworks and libraries, such as React, Angular, and Vue.js, that make web development more efficient and powerful.

JavaScript is a high-level, interpreted language, meaning that it is executed line by line and does not require compilation before running. It is dynamically typed, allowing variables to hold values of any type without specifying the type explicitly.

The syntax of JavaScript is similar to other programming languages like C++ and Java, making it relatively easy to learn for those familiar with those languages. It supports a variety of programming paradigms, including procedural, object-oriented, and functional programming.

1. What is the use of isNaN function?

The isNaN() function in JavaScript is used to determine whether a value is NaN (Not-a-Number). NaN is a special value in JavaScript that represents an invalid number.

The isNaN() function takes a single argument and checks if it is NaN. It performs the following steps:

(i)If the argument passed to isNaN() is not of the type "number", it is first converted to a Number type. If the conversion is not possible, it returns true. For example:

isNaN("Hello"); // true

(ii)If the argument is of type "number" or successfully converted to a number, it checks if the value is NaN. If the value is NaN, it returns true; otherwise, it returns false. For example:

isNaN(42); // false

isNaN(NaN); // true

isNaN(10 / "Two"); // true

The isNaN() function is often used to validate user input and determine if a given value is a valid number. It can be used to handle scenarios where a calculation or conversion results in an unexpected or invalid numeric value.

1. What is negative Infinity?

Negative Infinity is a special value in JavaScript that represents the mathematical concept of negative infinity. It is denoted by the constant Number.NEGATIVE\_INFINITY or simply -Infinity.

Infinity in mathematics represents a concept of a number that is larger than any finite number. Similarly, negative infinity represents a value that is smaller than any finite number.

Negative Infinity is often encountered in JavaScript when performing mathematical operations or comparisons that result in an infinitely small or infinitely negative value.

For example:

console.log(Number.NEGATIVE\_INFINITY); // -Infinity

console.log(-1 / Infinity); // -0

console.log(-10 / 0); // -Infinity

In the above code snippets, dividing a negative number by positive Infinity or dividing a negative number by zero will result in negative infinity.

Negative Infinity is considered a special value because it behaves differently than regular numbers in some cases. For example, any positive number multiplied by negative infinity will result in negative infinity:

console.log(10 \* Number.NEGATIVE\_INFINITY); // -Infinity

1. Which company developed JavaScript?

JavaScript was initially developed by Brendan Eich at Netscape Communications Corporation. Netscape, a now-defunct company, was one of the leading web browser companies in the 1990s. Brendan Eich created JavaScript in 1995 while working at Netscape, and it was originally named "Mocha" but was quickly renamed to "LiveScript" before finally being named "JavaScript." The language was developed as a scripting language for web browsers to add interactivity to web pages.

Netscape's release of JavaScript created a significant impact on web development, and JavaScript quickly gained popularity due to its ability to provide dynamic and interactive features on websites. Today, JavaScript is supported by all major web browsers and has become a fundamental technology in web development.

5.What are undeclared and undefined variables?

Undeclared and undefined variables are related but distinct concepts in JavaScript.

(i)Undeclared variables: An undeclared variable is a variable that has not been declared using the var, let, or const keywords before it is used. When you attempt to use an undeclared variable, JavaScript assumes it to be a global variable, regardless of the scope in which it is referenced. If a variable is undeclared and you try to read its value or assign a value to it, JavaScript will throw a reference error. For example:

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

To avoid undeclared variables, it's considered good practice to always declare variables explicitly before using them, using the appropriate keyword (var, let, or const).

(ii)Undefined variables: An undefined variable is a variable that has been declared, but it does not have a value assigned to it or has been explicitly assigned the value undefined. When you reference an undefined variable, JavaScript does not throw a reference error but returns the value undefined. For example:

let undefinedVariable;

console.log(undefinedVariable); // undefined

In this case, the variable undefinedVariable has been declared but not assigned a value, so its value is undefined.

It's important to note that undefined is a special value in JavaScript that indicates the absence of a value or the lack of an assigned value. It is not the same as an undeclared variable.

6.Write the code for adding new elements dynamically?

<!DOCTYPE html>

<html>

<head>

<title>Dynamic Element Addition</title>

</head>

<body>

<button onclick="addNewElement()">Add New Element</button>

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

<script>

function addNewElement() {

// Create a new element

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

// Set the content of the new element

newElement.textContent = "This is a new element added dynamically!";

// Get the container element

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

// Append the new element to the container

container.appendChild(newElement);

}

</script>

</body>

</html>

In this example, we have an HTML page with a button and a container div element. When the button is clicked, the addNewElement() JavaScript function is invoked.

Inside the function:

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

(ii)We set the content of the new element using newElement.textContent property.

(iii)We retrieve the container element using document.getElementById("container").

(iv)We append the new element to the container using container.appendChild(newElement).

Each time the button is clicked, a new paragraph element with the specified content will be added dynamically to the container div.

7.What is the difference between ViewState and SessionState?

(i) ViewState: ViewState is a feature in ASP.NET that allows the preservation of page state across postbacks. It is used to store and retrieve the values of controls and other page-specific data. ViewState is maintained as a hidden field on the web page, and it is encrypted and sent to the client browser with each response. When the page is posted back to the server, the ViewState is sent back to the server, allowing the server to restore the state of controls and data.

(ii) SessionState: SessionState is another feature in ASP.NET that enables the storage and retrieval of user-specific data across multiple requests and pages during a user session. SessionState stores data on the server side and associates it with a unique session identifier, usually stored in a cookie or passed in the URL. This allows the server to maintain user-specific information, such as user preferences, shopping cart items, login status, etc., throughout the user's session.

8.What is === operator?

The === operator in JavaScript is known as the strict equality operator. It is used to compare two values for equality without performing any type coercion. In other words, it checks whether the values being compared are not only equal in value but also of the same data type.

Here's how the === operator works:

(i)If the operands being compared have different types, the === operator immediately returns false without attempting any further comparison.

(ii)If the operands have the same type, the === operator performs a strict equality comparison by comparing their values. It returns true if the values are exactly the same, and false otherwise.

For example:

console.log(5 === 5); // true

console.log("5" === 5); // false (different types)

console.log(5 === "5"); // false (different types)

console.log(true === true); // true

console.log(null === undefined); // false (different types)

console.log(0 === -0); // true

console.log(NaN === NaN); // false (NaN is a special case)

In the examples above, the === operator compares values of different types, such as numbers and strings, and returns false due to the strict equality requirement. It returns true when comparing values of the same type and identical value.

8.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 style property or the classList property of the element object. Here's how you can do it:

(i)Changing the style of an element using the style property:

The style property allows you to access and modify individual CSS properties of an element. You can directly assign new values to specific style properties. For example:

// 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 = "20px";

In this example, the backgroundColor and fontSize properties of the element's style object are modified to change the background color and font size, respectively.

(ii)Changing the class of an element using the classList property:

The classList property provides methods to manipulate the classes of an element. You can add, remove, or toggle classes using the add(), remove(), and toggle() methods, respectively. For example:

// Get the element by its ID

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

// Add a class

element.classList.add("myClass");

// Remove a class

element.classList.remove("oldClass");

// Toggle a class

element.classList.toggle("active");

In this example, the add() method adds the class "myClass" to the element, the remove() method removes the class "oldClass", and the toggle() method toggles the class "active" (i.e., adds it if it's not present and removes it if it's already present).

9.How to read and write a file using JavaScript?

(i)Reading and writing files in a web browser (client-side):

JavaScript running in a web browser has limited access to the user's file system for security reasons. However, you can still interact with files indirectly using the File API and HTML5 features like the FileReader API.

To read a file selected by the user, you can use the following approach:

// HTML

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

<button onclick="handleFile()">Read File</button>

// JavaScript

function handleFile() {

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

var file = fileInput.files[0];

var reader = new FileReader();

reader.onload = function (event) {

var fileContent = event.target.result;

console.log(fileContent);

// Do something with the file content

};

reader.readAsText(file);

}

(ii) Reading and writing files in a server-side environment like Node.js:

In Node.js, you have access to the file system module (fs) that provides methods for reading and writing files. Here's an example of reading and writing files using Node.js:

const fs = require("fs");

// Reading a file

fs.readFile("path/to/file.txt", "utf8", function (err, data) {

if (err) {

console.error(err);

return;

}

console.log(data);

// Do something with the file content

});

// Writing a file

const content = "This is the content to be written.";

fs.writeFile("path/to/newfile.txt", content, function (err) {

if (err) {

console.error(err);

return;

}

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

});

10.What are all the looping structures in JavaScript?

(i) for loop: The for loop allows you to execute a block of code for a specified number of iterations. It consists of an initialization statement, a condition, an iteration expression, and the code block to be executed. For example:

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

console.log(i);

}

(ii) while loop: The while loop repeatedly executes a block of code as long as a specified condition is true. It checks the condition before each iteration. For example:

var i = 0;

while (i < 5) {

console.log(i);

i++;

}

(iii) do...while loop: The do...while loop is similar to the while loop, but it checks the condition after executing the code block. This guarantees that the code block is executed at least once. For example:

var i = 0;

do {

console.log(i);

i++;

} while (i < 5);

(iv) for...in loop: The for...in loop is used to iterate over the properties of an object. It iterates through each enumerable property of an object. For example:

var person = {

name: "John",

age: 30,

city: "New York"

};

for (var prop in person) {

console.log(prop + ": " + person[prop]);

}

(v) for...of loop: The for...of loop is introduced in ECMAScript 6 and is used to iterate over iterable objects such as arrays, strings, or collections. It provides an easy way to iterate through the elements of an iterable. For example:

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

for (var num of numbers) {

console.log(num);

}

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

Here's how you can use parseInt() to convert a string of any base to an integer:

var binaryString = "101010"; // Binary string

var octalString = "52"; // Octal string

var hexadecimalString = "1A"; // Hexadecimal string

var binaryNumber = parseInt(binaryString, 2); // Convert binary string to integer

var octalNumber = parseInt(octalString, 8); // Convert octal string to integer

var hexadecimalNumber = parseInt(hexadecimalString, 16); // Convert hexadecimal string to integer

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

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

console.log(hexadecimalNumber); // Output: 26

In this example, parseInt() is used to convert the strings binaryString, octalString, and hexadecimalString to integers. The second argument specifies the base of the number system used in the string (2 for binary, 8 for octal, and 16 for hexadecimal).

The parseInt() function parses the string and returns an integer representation of the string in the specified base. The resulting integers (binaryNumber, octalNumber, hexadecimalNumber) can be used in further calculations or operations as needed.

Note that if the input string starts with "0x", it is automatically interpreted as a hexadecimal string, so you don't need to specify the base explicitly in that case. For example:

var hexadecimalString = "0x1A"; // Hexadecimal string with "0x" prefix

var hexadecimalNumber = parseInt(hexadecimalString); // Automatically interpreted as base 16

console.log(hexadecimalNumber); // Output: 26

12.What is the function of the delete operator?

(i) Deleting object properties:

When used with an object, the delete operator can be used to remove a property from the object. It allows you to delete both own properties and inherited properties.

var person = {

name: "John",

age: 30

};

console.log(person); // Output: { name: "John", age: 30 }

delete person.age;

console.log(person); // Output: { name: "John" }

In this example, the delete operator is used to remove the age property from the person object. After the deletion, the person object only contains the name property.

Note that the delete operator only removes the property itself, not its prototype chain. In other words, if the deleted property is inherited from a prototype, the property will still be accessible through the prototype chain.

(ii)Deleting array elements:

When used with an array, the delete operator can be used to remove an element from the array. However, it does not update the length of the array or reindex the remaining elements.

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

console.log(numbers); // Output: [1, 2, 3, 4, 5]

delete numbers[2];

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

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

(i)

alert(): The alert() function displays a simple dialog box with a message and an OK button. It is commonly used to show informative messages to the user.

alert("Hello, World!");

(ii) confirm(): The confirm() function displays a dialog box with a message and two buttons: OK and Cancel. It is used to get a confirmation from the user and returns a boolean value indicating the user's choice.

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

if (result) {

// User clicked OK

} else {

// User clicked Cancel

}

(iii) prompt(): The prompt() function displays a dialog box with a message, an input field for the user to enter text, and two buttons: OK and Cancel. It is used to prompt the user for input and returns the entered value as a string.

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

if (name !== null) {

// User entered a value

} else {

// User clicked Cancel

}

14. What is the use of Void (0)?

In JavaScript, void(0) is a special expression that evaluates to the undefined value. It is often used in HTML anchor (<a>) tags as the href attribute to prevent the page from navigating to a new URL when the anchor is clicked. The use of void(0) essentially creates a "no-op" action

Here's an example of how void(0) can be used in an anchor tag:

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

In this example, when the anchor is clicked, the doSomething() function will be executed. However, since the href attribute is set to javascript:void(0), the page won't navigate or refresh.

The void operator in JavaScript takes an expression and evaluates it, but discards the resulting value and always returns undefined. By using void(0), you ensure that the anchor tag's href attribute is set to an expression that doesn't have any side effects and doesn't navigate the page.

<a href="#" onclick="doSomething(event)">Click me</a>

function doSomething(event) {

event.preventDefault();

// Perform actions or navigate programmatically

}

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

(i)Using window.location.href:

You can set the href property of window.location to the desired URL, and the browser will navigate to that URL. Here's an example:

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

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

The assign() method of the window.location object can also be used to load another page. It works similarly to setting the href property. Here's an example:

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

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

The replace() method of the window.location object can be used to load another page and replace the current page in the browser's history. This means the user won't be able to navigate back to the original page using the browser's back button. Here's an example:

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

16. What are the disadvantages of using innerHTML in JavaScript?

(i). Security risks: Using `innerHTML` can expose your web application to potential security vulnerabilities, such as cross-site scripting (XSS) attacks. If you're not careful, inserting user-generated content directly into the `innerHTML` property can execute arbitrary scripts and compromise the security of your application. It's crucial to properly sanitize and validate user input before using `innerHTML`.

(ii). Performance impact: Modifying the `innerHTML` property of an element causes the browser to reparse and re-render the entire content of that element. If you frequently update large portions of the HTML using `innerHTML`, it can lead to performance degradation, especially in complex or large web pages. In such cases, using more targeted DOM manipulation methods, like creating and appending individual elements, can be more efficient.

(iii). Event listener detachment: When you modify the `innerHTML` property of an element, any event listeners attached to the existing child elements will be lost. This happens because the old elements are removed from the DOM and new elements are inserted. If you have event handlers attached to the child elements, you'll need to reattach them after updating the `innerHTML`.

(iv). Limited manipulation: While `innerHTML` allows you to set or retrieve the HTML content of an element, it doesn't provide granular control over individual elements or attributes within that content. If you need to modify specific elements, attributes, or manipulate the DOM structure, using the DOM manipulation methods like `createElement`, `appendChild`, `setAttribute`, etc., provide more flexibility and control.

(v). Accessibility issues: When you modify the `innerHTML` property, assistive technologies like screen readers may not interpret the changes correctly. This can result in a less accessible experience for users who rely on such technologies. It's important to ensure that any dynamic content modifications maintain proper accessibility standards.

17. Create password field with show hide functionalities

HTML

<label for="password">Password:</label>

<input type="password" id="password" placeholder="Enter your password">

<input type="checkbox" id="showPassword">

<label for="showPassword">Show Password</label>

JAVASCRIPT

var passwordInput = document.getElementById("password");

var showPasswordCheckbox = document.getElementById("showPassword");

showPasswordCheckbox.addEventListener("change", function () {

if (showPasswordCheckbox.checked) {

passwordInput.type = "text";

} else {

passwordInput.type = "password";

}

});