javascript Assigment.

Q-1 What is JavaScript?

- JavaScript is a versatile programming language that is primarily used to add interactivity and dynamic behavior to websites. Originally created by Brendan Eich at Netscape in 1995, it has since become one of the most popular and widely used languages for web development.

Client-Side Language: JavaScript is mainly a client-side language, meaning it runs on the user's web browser rather than on a web server. This allows developers to create interactive features that respond to user actions without needing to constantly communicate with the server.

Versatile: JavaScript is a versatile language that can be used for a wide range of tasks, from simple tasks like form validation to complex web applications and even mobile app development (with frameworks like React Native).

Object-Oriented: It supports object-oriented programming (OOP) paradigms, allowing developers to create reusable and modular code.

Q-2 What is the use of isNaN function?

-The isNaN function in JavaScript stands for "Is Not a Number." It is used to determine whether a value is NaN (Not-a-Number) or not. Here's how it works

-syntax

isNaN(value)

If the argument is NaN or cannot be converted into a number, isNaN will return true.

If the argument is a valid number, isNaN will return false.

example-

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

if (isNaN(userInput)) {

    alert("Please enter a valid number!");

} else {

    alert("You entered a number: " + userInput);

}
```

Q-3 What is negative Infinity?

In JavaScript, Infinity is a special numeric value that represents positive infinity. Similarly, there is also a special value called -Infinity, which represents negative infinity.

Infinity represents a value that is greater than any other number.

-Infinity, on the other hand, represents a value that is smaller (more negative) than any other number, including negative numbers.

It is often used to represent results that are mathematically undefined or impossible, such as dividing a positive number by zero.

example-

```
let max = -Infinity;

let numbers = [5, 10, -20, 3];

for (let number of numbers) {

    if (number > max) {

        max = number;

    }
```

```
}

console.log("Max value:", max);    // 10
```

Q-4 Which company developed JavaScript?

-JavaScript was developed by Netscape Communications Corporation, particularly by Brendan Eich. Brendan Eich created the initial version of JavaScript in just ten days in May 1995. It was originally named "Mocha" and was later renamed "LiveScript," before finally settling on "JavaScript" due to the popularity of Java at the time. Netscape Navigator 2.0 was the first browser to incorporate JavaScript, introducing it to the world in September 1995.

Q-5 What are undeclared and undefined variables?

-Undeclared and undefined variables are two different concepts in JavaScript, each with its own meaning and implications.

An undeclared variable is a variable that has been used in code without being declared using the var, let, or const keywords. When you try to use a variable without declaring it, JavaScript will treat it as an undeclared variable.

syntax-   console.log(myVar);

- undefined variables

An undefined variable, on the other hand, is a variable that has been declared but has not been assigned a value. In JavaScript, when a variable is declared but not initialized, its default value is undefined.

ex-

```
let myVar;

console.log(myVar);
```

Q-6 Write the code for adding new elements dynamically?

-Create a new element.

Optionally, set attributes or content for the new element.

Append the new element to an existing element in the DOM (Document Object Model).

Here's an example that demonstrates how to create a new element with some text content and append it to an existing with an id of "container" when a button is clicked:

example

```
<div id="container">
    <!-- Existing content -->
</div>
<button id="addButton">Add New Element</button>
```

Q-7 What is the difference between ViewState and SessionState?

- ViewState and SessionState are concepts in web development, often associated with ASP.NET, though the general concepts are applicable to other web frameworks as well. They both serve to maintain state information in web applications, but they do so in different scopes and with different purposes.

ViewState is specific to ASP.NET Web Forms. It's used to preserve page and control values between postbacks. When a form is submitted to the server and the page reloads, ViewState helps to maintain the state of controls on the page, like textboxes, dropdowns, etc.

It stores data related to a single web form and keeps it on the client side, often as a hidden field.

SessionState is a more general concept in web development, not specific to

ASP.NET Web Forms but used in various web frameworks.

It is used to store user-specific information across multiple pages or requests during a user session.

SessionState data is stored on the server-side, typically in memory or a database, and is identified by a session ID, which is usually stored in a cookie on the client-side.

--viewstate example

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        // Initialize some control values
        TextBox1.Text = "Initial Value";
    }
}
```

--sessionstate example

```
Session["UserName"] = "JohnDoe";


// Retrieving Session variable
string userName = Session["UserName"] as string;
```

Q-8 What is === operator?

-The === operator in JavaScript is called the "strict equality" operator. It is used for comparing two values to check if they are equal in both value and data type.

If the values being compared are of different types, the === operator will return false without attempting to convert the values to the same type.

If the values are of the same type, it behaves exactly like the == operator (the "loose equality" operator), comparing their values.

```
console.log(5 === 5);

console.log('5' === 5);

console.log(5 === '5');

console.log(true === true);

console.log(true === 1);
```

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

-To change the style or class of an HTML element using JavaScript, you have a few different options depending on what exactly you want to achieve:

--using style property

example-

```
/ Get the element by ID

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

// Change background color

element.style.backgroundColor = "red";

// Change font size

element.style.fontSize = "20px";
```

--using class property

example-

```
// Get the element by ID
```

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

// Add a CSS class

element.classList.add("newClass");

// Remove a CSS class

element.-In a web brclassList.remove("oldClass");

// Toggle a CSS class (add if not present, remove if present)

element.classList.toggle("active");
```

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

owser environment (client-side JavaScript), reading and writing files directly from the user's device is restricted for security reasons. However, JavaScript can read and write files in specific environments, such as server-side with Node.js or within a web browser using the File API for file input fields. Below are examples for both scenarios:

```
  write and read with nodejs example

const fs = require('fs');

// Reading a file asynchronously

fs.readFile('example.txt', 'utf8', (err, data) => {
    if (err) {
        console.error(err);
        return;
    }
    console.log(data);
});
```

--API example

```
<input type="file" id="fileInput">
<button onclick="readFile()">Read File</button>
<div id="fileContents"></div>
```

--write browser exmple

```
function downloadFile() {
    const content = "Hello, World!";
    const blob = new Blob([content], { type: 'text/plain' });
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
    a.download = 'example.txt';
    document.body.appendChild(a);
    a.click();
    URL.revokeObjectURL(url);    // Release the object URL
}
```

Q-11 What are all the looping structures in JavaScript?

- JavaScript offers several looping structures that allow you to execute a block of code repeatedly. Here are the main looping structures in JavaScript

for Loop:

The for loop is one of the most commonly used loops in JavaScript. It is used when

you know the number of iterations.

--example

```
for (let i = 0; i < 5; i++) {
    console.log(i);
}
```

while Loop:

The while loop continues to execute a block of code as long as the condition is true. It is used when the number of iterations is not known beforehand.

--example

```
let i = 0;
while (i < 5) {
    console.log(i);
    i++;
}
```

do...while Loop:

Similar to the while loop, the do...while loop also executes a block of code while a specified condition is true. The difference is that the do...while loop always executes the block of code at least once, even if the condition is false.

example--

```
let i = 0;
do {
    console.log(i);
    i++;
```

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

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

- In JavaScript, you can convert a string representation of a number in any base (binary, octal, hexadecimal, etc.) to an integer using the parseInt function along with specifying the radix (base) as the second argument. The parseInt function takes two arguments: the string to convert and the radix (base).

example--

```
let binaryString = "101010";

let integerValue = parseInt(binaryString, 2);

console.log(integerValue);    // Output: 42
```

Q-13 What is the function of the delete operator?

- In JavaScript, the delete operator is used to remove a property from an object. Its main function is to delete a specific property of an object or an element at a specified index in an array. Here's how it works

example-

```
let car = {
    brand: 'Toyota',
    model: 'Camry',
    year: 2020
};

console.log(car);    // Before deletion

// Deleting the 'year' property
```

```
delete car.year;

console.log(car);    // After deletion
```

Deleting Array Elements:

When used with an array, delete removes an element at a specified index. However, this leaves a hole in the array, meaning that the length of the array does not change, but the element at the deleted index becomes undefined.

example-

```
let numbers = [10, 20, 30, 40, 50];

console.log(numbers);    // Before deletion

// Deleting element at index 2

delete numbers[2];

console.log(numbers);    // After deletion (index 2 is now undefined)
```

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

- Alert Box (alert()):

The alert() function displays a simple dialog box with a message and an "OK" button. It is used to give the user some information or to display an alert.

ex-

```
alert("Hello! This is an alert box.");
```

Confirm Box (confirm()):

The confirm() function creates a dialog box with a message and two buttons: "OK" and "Cancel". It is often used when you want the user to confirm or cancel an

action.

ex-

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

if (result) {

    // User clicked "OK"

    deleteItem();

} else {

    // User clicked "Cancel"

    // Do nothing or provide feedback

}
```

Prompt Box (prompt()):

The prompt() function displays a dialog box with a message, an input field for the user to enter text, and "OK" and "Cancel" buttons. It is used when you want the user to input some value.

ex-

```
let name = prompt("Please enter your name:", "John Doe");

if (name !== null) {

    // User entered a name

    alert("Hello, " + name + "!");

} else {

    // User clicked "Cancel" or closed the prompt

    alert("You did not enter your name.");
```

}

Q-15 What is the use of Void (0)?

In JavaScript, void is an operator that takes an expression as an argument and evaluates it, returning undefined. When void is followed by (0), it essentially evaluates 0 and returns undefined. This can be used in various scenarios:

In this example, the href attribute is set to javascript:void(0);. When the user clicks the link, it executes the doSomething() function but does not navigate to a new page. This is often used to create "dummy" links that trigger JavaScript actions without changing the page.

If someFunction() returns a value but you're not interested in using that value, you can use void to explicitly discard it. This is especially useful if you want to call a function for its side effects without capturing its return value.

example-

```
function calculateSquareRoot(num) {
    if (num >= 0) {
        return Math.sqrt(num);
    } else {
        alert("Cannot calculate square root of a negative number.");
        return void 0; // Explicitly return undefined
    }
}
let result1 = calculateSquareRoot(25); // Result: 5
let result2 = calculateSquareRoot(-10); // Alert shown, result2 is undefined
```

Q-16 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 new page. This effectively redirects the browser to the specified URL. There are a few different ways to achieve this redirection:

Using window.location.href:

ex

// Redirect to a new page

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

--Using a Click Event:

You can also achieve page redirection by simulating a click on an anchor (<a>) element.

ex html

Click to Redirect

js

// Simulate click event on the anchor element

document.getElementById("redirectLink").addEventListener("click", function(event) {

    event.preventDefault(); // Prevent the default behavior of the link

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

});

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

While the innerHTML property in JavaScript is a convenient and commonly used way to manipulate HTML content within an element, it does have some disadvantages and potential risks, especially in certain contexts. Here are some of the disadvantages of using innerHTML:

1. Security Risks (Cross-Site Scripting - XSS):

One of the biggest concerns with using innerHTML is the potential for Cross-Site Scripting (XSS) vulnerabilities.

If the content being inserted via innerHTML includes untrusted or user-generated data, it can be a security risk. If the data contains malicious scripts, they will be executed when inserted into the DOM.

2. Performance Impact:

When you use innerHTML to modify a large portion of the DOM, it can cause the browser to re-parse and re-render the entire contents of the affected element and its children.

This can lead to a performance hit, especially on complex pages or when manipulating large amounts of content.

3. Overwriting Event Listeners and Data:

When you set innerHTML, it completely replaces the content inside the targeted element.

If the existing content includes event listeners, data, or state that you want to preserve, setting innerHTML can inadvertently remove these.

This can lead to unexpected behavior if you are not careful.