MODULE (JAVASCRIPT BASIC & DOM) -- 4

01) What is JavaScript?

JavaScript is a high-level, interpreted programming language primarily used for creating interactive and dynamic content on web pages. It is one of the core technologies of the World Wide Web, alongside HTML and CSS. JavaScript allows developers to add interactivity, behavior, and functionality to web pages, making them more engaging and responsive to user actions.

02) What is the use of isNaN function?

The isNaN() function in JavaScript is used to determine whether a value is NaN (Not-a-Number) or not. NaN represents a value that is not a valid number, typically resulting from arithmetic operations that involve non-numeric values or undefined results.

The isNaN() function takes one argument and returns a boolean value:

If the argument is NaN or cannot be converted to a number, isNaN() returns true.

If the argument is a valid number, isNaN() returns false.

Eg:-

```
console.log(isNaN("Hello"));

console.log(isNaN(42));

console.log(isNaN(NaN));

console.log(isNaN(undefined));
```

03) What is negative Infinity?

Negative Infinity is a special value in JavaScript that represents the mathematical concept of negative infinity. It is used to denote a value that is smaller (i.e., less than) any other numeric value, including negative numbers.

In JavaScript, Negative Infinity is represented by the global property Number.NEGATIVE_INFINITY. It is used to indicate values that are smaller than the smallest

representable number in JavaScript, or as a result of arithmetic operations that produce values that are too small to be represented.

Here's how you can use Negative Infinity in JavaScript:

```
console.log(Number.NEGATIVE_INFINITY);

console.log(-1 / 0);


console.log(Number.NEGATIVE_INFINITY < -100);
console.log(Number.NEGATIVE_INFINITY > Number.MIN_SAFE_INTEGER);
```

04) Which company developed JavaScript?

JavaScript was developed by Netscape Communications Corporation, a company founded in 1994 that played a significant role in the early days of the internet. Brendan Eich, a programmer at Netscape, created JavaScript in 1995 under the name "Mocha," which was later renamed to "LiveScript" and finally "JavaScript" due to its close association with Java. JavaScript was introduced in Netscape Navigator 2.0, released in December 1995, as a scripting language for client-side web development. Since then, JavaScript has evolved into one of the most widely used programming languages for web development, with implementations in all major web browsers and a large ecosystem of frameworks, libraries, and tools.

05) What are undeclared and undefined variables?

1.Undeclared Variables:

Undeclared variables are variables that have been used in code without being declared using the var, let, or const keywords.

When you attempt to use an undeclared variable, JavaScript will either implicitly create a global variable (if not in strict mode) or throw a ReferenceError (if in strict mode).

Using undeclared variables is generally considered bad practice because it can lead to unexpected behavior and makes code harder to maintain.

Example of using an undeclared variable:

```
// Undeclared variable
myVar = 10;
```

2.Undefined Variables:

Undefined variables are variables that have been declared but not assigned a value, or variables that have been accessed before they are assigned a value.

When you access an undefined variable, JavaScript returns the special value undefined, which indicates that the variable has been declared but does not currently have a value.

Example of accessing an undefined variable:

```
let myVar;
console.log(myVar); // Output: undefined

let anotherVar;
console.log(anotherVar); // Output: undefined
anotherVar = 20;
```

06) What is the difference between ViewState and SessionState?

1. ViewState:

ViewState is used to persist the state of a web page across postbacks, i.e., between consecutive requests for the same page.

It is implemented as a hidden field on the page that stores the values of certain controls and their properties.

ViewState is specific to a single page and is maintained on the client side.

It is primarily used to maintain the state of controls (such as textboxes, checkboxes, etc.) between postbacks without storing the data on the server.

The data stored in ViewState is encoded and embedded within the HTML of the page, so it increases the size of the page and can potentially impact performance if used excessively.

2.SessionState:

SessionState is used to store user-specific data across multiple requests during a user's session on the website.

It is stored on the server side, either in memory, in-process, or out-of-process (e.g., in a SQL Server database or a state server).

SessionState is available across multiple pages in a web application and is accessible to all users with an active session.

It is commonly used to store user authentication status, user preferences, shopping cart contents, and other user-specific data.

Unlike ViewState, SessionState data is not transmitted between the client and server with each request, reducing the size of page payloads.

07) What is === operator?

The === operator, known as the strict equality operator, is used in JavaScript to compare two values for equality without performing type coercion. It checks both the value and the data type of the operands.

Here's how it works:

If the operands are of the same data type and have the same value, === returns true.

If the operands are of different data types or have different values, === returns false.

For example:

```
console.log(5 === 5);
console.log('5' === 5);
console.log('5' === '5');
console.log(true === true);
console.log(null === null);
console.log(undefined === undefined);
console.log(0 === false);
```

08) How can the style/class of an element be changed?

To change the style or class of an HTML element dynamically using JavaScript, you can manipulate the properties of the element directly. There are several ways to achieve this:

= Changing Inline Styles:

You can directly modify the inline style of an element using the style property.

```
// Change background color of an element with id "myElement"
document.getElementById("myElement").style.backgroundColor = "blue";
```

= Adding or Removing Classes:

You can add or remove classes from an element using the classList property.

```
// Add a class to an element with id "myElement"
document.getElementById("myElement").classList.add("newClass");

// Remove a class from an element with id "myElement"
document.getElementById("myElement").classList.remove("oldClass");
```

=Toggle Classes:

You can toggle the presence of a class on an element using the classList.toggle() method.

```
// Toggle a class on an element with id "myElement"
document.getElementById("myElement").classList.toggle("active");
```

= Setting Multiple Styles:

You can set multiple styles at once using the cssText property or by directly assigning style properties.

```
// Using cssText
document.getElementById("myElement").style.cssText = "color: red; font-size: 16px;";

// Directly assigning style properties
const element = document.getElementById("myElement");
element.style.color = "red";
element.style.fontSize = "16px";
```

09) How to read and write a file using JavaScript?

In a web browser environment, JavaScript does not have direct access to the filesystem for security reasons. However, in server-side JavaScript environments like Node.js, you can read and write files using built-in modules like fs (file system).

10) What are all the looping structures in JavaScript?

1.for loop:

The for loop is used when you know the number of iterations in advance.

It consists of three parts: initialization, condition, and iteration.

Example:

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

```
    console.log(i);
}
```

2.while loop:

The while loop is used when you don't know the number of iterations in advance, and the loop continues as long as a specified condition evaluates to true.

Example:

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

3.do...while loop:

The do...while loop is similar to the while loop, but it always executes the block of code at least once before checking the condition.

Example:

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

4.for...in loop:

The for...in loop iterates over the enumerable properties of an object.

It is often used to loop through the keys 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:

The for...of loop iterates over the iterable objects such as arrays, strings, maps, sets, etc.

It provides a more concise syntax compared to other loops when iterating over values.

Example:

```
const arr = [1, 2, 3, 4, 5];
for (let value of arr) {
   console.log(value);
}
```

6. forEach:

n JavaScript, there is no built-in forEach loop. However, there is a forEach() method available for arrays, which is often referred to as a "for each" loop due to its functionality. The forEach() method is used to iterate over the elements of an array and execute a provided function once for each array element.

Here's how you can use the forEach() method:

```
const array = [1, 2, 3, 4, 5];
```

```
array.forEach(function(element) {
   console.log(element);
});
```

11) What is the function of the delete operator?

The delete operator in JavaScript is used to remove a property from an object or an element from an array. Its function depends on the type of the operand:

1.Removing Properties from Objects:

When used with an object property, delete removes that property from the object.

Example:

```
const obj = { a: 1, b: 2, c: 3 };
delete obj.b;
console.log(obj);
```

2.Removing Elements from Arrays:

When used with an array element, delete removes that element from the array but leaves a hole (an empty slot) in its place. The length of the array is not affected, and subsequent elements retain their original indices.

Example:

```
const arr = [1, 2, 3, 4, 5];
delete arr[2];
console.log(arr);
```

3.Removing Variables:

The delete operator cannot be used to delete variables declared with var, let, or const, nor can it delete variables declared as function parameters or any other lexical bindings. Attempting to delete such variables will result in false, and the variable will remain unchanged.

Example:

```
let x = 10;
console.log(delete x);
```

12) What are all the types of Pop up boxes available in JavaScript?

In JavaScript, there are three types of popup boxes available:

1.Alert Box (alert()):

The alert() method displays an alert dialog with a message and an OK button.

It is commonly used to show a message to the user.

Syntax:

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

2.Confirm Box (confirm()):

The confirm() method displays a dialog box with a message, an OK button, and a Cancel button.

It is used to get a confirmation from the user.

It returns true if the user clicks OK and false if the user clicks Cancel.

Syntax:

```
const result = confirm("Do you want to continue?");
if (result) {
   // User clicked OK
} else {
   // User clicked Cancel
}
```

3.Prompt Box (prompt()):

The prompt() method displays a dialog box with a message, an input field for the user to enter text, an OK button, and a Cancel button.

It is used to prompt the user for input.

It returns the value entered by the user as a string, or null if the user clicks Cancel.

Syntax:

```
const userInput = prompt("Please enter your name:");
if (userInput !== null) {
   // User entered a value
} else {
   // User clicked Cancel
}
```

13) What is the use of Void (0)?

In JavaScript, void(0) is an expression that evaluates to the undefined value. It is commonly used in conjunction with the javascript: pseudo-protocol in HTML anchor (<a>) tags to prevent the browser from navigating to a new page when the anchor is clicked.

Here's how it works:

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

14) What are the disadvantages of using innerHTML in JavaScript?

While innerHTML is a convenient and commonly used property in JavaScript to manipulate the content of HTML elements, it does have some disadvantages:

Security Risks:

Using innerHTML to dynamically insert HTML content received from untrusted sources can expose your website to cross-site scripting (XSS) attacks. If the inserted content contains malicious scripts, they can execute in the context of your web page, potentially compromising user data or performing unauthorized actions.

Performance Overhead:

Manipulating innerHTML involves parsing and re-rendering the entire content of the targeted element and its descendants. This can be inefficient, especially for large HTML structures, as it affects performance by consuming more CPU and memory resources compared to targeted DOM manipulation methods like appendChild() or createElement().

Event Handlers and Data Loss:

When using innerHTML to update the content of an element, any existing event listeners or data associated with the element's descendants may be lost. This is because innerHTML completely replaces the existing content, including its associated JavaScript objects, which can lead to unexpected behavior or functionality loss.

Invalid HTML and Style Overwriting:

Assigning HTML strings to innerHTML may result in invalid HTML structure if the string contains incomplete or incorrect markup. Additionally, any inline styles or event handlers defined in the original HTML content may be overwritten or removed when replacing the content with innerHTML.

Accessibility Concerns:

Dynamically updating the content of elements using innerHTML without proper consideration for accessibility guidelines may result in decreased accessibility for users of assistive

technologies. For example, dynamically inserted content may not be announced by screen readers, affecting users with visual impairments.