Module 6) JAVASCRIPT BASIC & DOM

1) What is JavaScript?

→ JavaScript is the world's most popular programming language.

→ JavaScript is a scripting language that enables to create dynamically updating content, control multimedia, animate images, and pretty much everything else in web pages.

→ JavaScript is primarily used for building interactive and dynamic content on websites. Originally created to make web pages more interactive.

→ JavaScript can be used for Client-side developments as well as Server-side developments.

→ JavaScript allows you to add features like user authentication, form validation, and dynamic content updates without requiring the user to refresh the page.

2) What is the use of isNaN function?

→ The isNaN() function can be used to test whether a value is NaN or not. It takes one argument and returns either true or false.

→ It returns true if the value is a NaN else returns false.

→ For example, if you have a variable x that contains the string "Hello", you can use isNaN() to check if x is NaN like this:

```
let x = "Hello"
isNaN(x); // returns true
```

→ If x was a number, such as 42, the isNaN() function would return false.

```
let x = 42;
isNaN(x) // return false
```

3) What is negative Infinity?

→ The negative infinity in JavaScript is a constant value that is used to represent a value that is the lowest available.
→ This means that no other number is lesser than this value. It can be generated using a self-made function or by an arithmetic operation.
→ JavaScript shows the NEGATIVE_INFINITY value as -Infinity.

4) Which company developed JavaScript?

→ JavaScript was developed by Netscape Communications Corporation.
→ Brendan Eich, a programmer at Netscape, created JavaScript in 1995.
→ The initial name for the language was "Mocha," but it was later renamed JavaScript to capitalize on the popularity of Java, another programming language at the time.

5) What are undeclared and undefined variables?

→ Undeclared variable :
→ An undeclared variable is a variable that has been used in the code without being declared or defined before its usage.
→ Using a variable without declaring it first can lead to errors.
→ in JavaScript , if we use x variable without being declared it shows error like this,

```
console.log(x); // Error: x is not defined
```

→ Undefined Variables:

→ An undefined variable is a variable that has been declared but has not been assigned a value.

→ Variables are automatically assigned a default value (which is often undefined) when they are declared but not explicitly assigned.

→ if we declared a variable name without assigned its value it shows not a error but when we check its value by using typeof operator it shows undefined as showed in below example..

→
```
let name;
console.log(typeof name)
        result : undefined
```

6) Write the code for adding new elements dynamically?

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
</head>

<body>
  <div class="button">
    <button id="addTask">Add task</button>
  </div>
  <div class="tasks"></div>
  <script >
    let task = document.getElementsByClassName("tasks");
```

```
        let addTask = document.getElementById("addTask");

        addTask.addEventListener("click", function () {

          for (let i = 0; i < task.length; i++) {

            let newDiv = document.createElement("div");

            newDiv.innerText = "New Div Element created";

            task[i].append(newDiv);
          }
        });
      </script>
    </body>
  </html>
```

7) What is the difference between ViewState and SessionState?

→ **ViewState:**

→ It is maintained at only one level that is page-level. Changes made on a single page are not visible on other pages.

→ Information that is gathered in view state is stored for the clients only and cannot be transferred to any other place.

→ It can be used to store information that you wish to access from same web page.

→ **SessionState:**

→ It is maintained at session-level and data can be accessed across all pages in the web application.
→ The information is stored within the server and can be accessed by any person that has access to the server where the information is stored.
→ It can be used to store information that you wish to access on different web pages.

8) What is === operator?

→ Triple equals (===), also referred to as "strict equality",
→ It is used to compare two values for equality without performing type coercion.
→ If the operands are of different types, === returns false without attempting to convert the operands.
→ If the operands are of the same type and have the same value, === returns true.
→ If the operands are of the same type but have different values, === returns false.
→ console.log(5 === '5'); // false (different types)
→ console.log('hello' === 'hello'); // true (both are strings with the same value).

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

Changed in style:-

→ **1.Direct Style Property Modification:**

```
var myElement = document.getElementById('myElementId');

// Change the background color directly
```

```
myElement.style.backgroundColor = 'red';
```

→ **2.Using `setAttribute`:**

```
var myElement = document.getElementById('myElementId');

// Set the style attribute directly

myElement.setAttribute('style', 'background-color: blue; color:
white;');
```

Changed in class:-

→ **1. Using classList property:**

```
<div id="myElement">Hello, World!</div>

<script>
  var element = document.getElementById('myElement');

  // Add a class
  element.classList.add('myClass');

  // Remove a class
  element.classList.remove('otherClass');

 // Toggle a class (add if not present, remove if present)
 myElement.classList.toggle('toggleClass');
</script>
```

→ **2.Direct Class Property Modification:**

```
<div id="myElement">Hello, World!</div>
```

```
// Get the element by its ID
var element = document.getElementById('myElement');
// Add a class
element.className = 'newClass';
```

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

→ On the client side, you can't read or write files in JavaScript browsers.

→ The fs module in Node.js may be used to accomplish this on the server-side. It has methods for reading and writing files on the file system that are both synchronous and asynchronous.

→ The fs.readFile() and rs.writeFile() methods are used to read and write of a file using javascript. The file is read using the fs.readFile() function, which is an inbuilt method. This technique reads the full file into memory and stores it in a buffer.

11) What are all the looping structures in JavaScript?

→ Here are the main looping structures in JavaScript:

1. for Loop:

→ The for loop is a common loop that consists of three expressions: initialization, condition, and iteration.

```
for (initialization; condition; iteration) {
  // code to be repeated
}
```

2. while Loop:

→ The while loop repeats a block of code as long as a specified condition is true.

```
while (condition) {
  // code to be repeated
}
```

3. do-while Loop:

→ Similar to the `while` loop, but the block of code is executed at least once, even if the condition is initially false.

```
do {
 // code to be repeated
} while (condition);
```

4. for...in Loop:

→ Iterates over the enumerable properties of an object. It's typically used for iterating over the keys of an object.

```
for (variable in object) {
// code to be repeated
}
```

5. for...of Loop:

→ Introduced in ECMAScript 2015 (ES6), the for...of loop iterates over the values of iterable objects (arrays, strings, etc.).

```
for (variable of iterable) {
// code to be repeated
}
```

6. forEach loop:

→ ForEach loop is a functional loop.

→ Available for arrays, the `forEach` method executes a provided function once for each array element.

```
array.forEach(function(element) {
// code to be repeated
});
```

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

→ In JavaScript, you can use the parseInt function to convert a string representation of a number from any base to an integer.

→ The parseInt function takes two arguments: the string to be converted and the base of the numeral system.

→ Example:

```
var a = "1101";
var Number =Number. parseInt(a);
```

13) What is the function of the delete operator?

→ This article is going to discuss the delete operator available in JavaScript.

→ Delete is comparatively a lesser-known operator in JavaScript.

→ This operator is more specifically used to delete JavaScript object properties.

→ The delete operator only works on objects and not on variables or functions.

→ Example:

```
let emp = {
    firstName: "Raj",
    lastName: "Kumar",
    salary: 40000
}
```

```
console.log(delete emp.salary);
console.log(emp);

result:
        true
        {"firstName":"Raj","lastName":"Kumar"}
```

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

→ There are three types of pop-up boxes in JavaScript namely Alert Box, Confirm Box and Prompt Box.

→ In Javascript, popup boxes are used to display the message or notification to the user.

→ **1. Alert Box:** It is used when a warning message is needed to be produced. When the alert box is displayed to the user, the user needs to press ok and proceed.

→ Syntax:

```
alert("your Alert here")
```

→ **2. Prompt Box:** It is a type of pop up box which is used to get the user input for further use.

→ After entering the required details user have to click ok to proceed next stage else by pressing the cancel button user returns the null value.

→ Syntax:

```
prompt("your Prompt here")
```

→ **3. Confirm Box:** It is a type of pop-up box that is used to get authorization or permission from the user. The user has to press the ok or cancel button to proceed.

→ Syntax:

```
confirm("are you sure to delete this item")
```

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

→ JavaScript void 0 means returning undefined (void) as a primitive value.

→ Suppose you insert a link and want to call some JavaScript through it. Usually, when you click on a link, the browser will either reload or open a new page.

→ If you just want to call JavaScript through that link, you would not want the entire page to refresh. This is where the JavaScript:void(0) will come in handy.

→ When you use JavaScript void 0, it will return an undefined primitive value. This will prevent the browser from opening a new or reloading the web page and allowing you to call the JavaScript through it.

16) How can a page be forced to load another page in JavaScript?

→ In JavaScript, we can use window.location object to force a page to load another page. We can use the location object to set the URL of a new page.

→ There are different ways – window.location.href property, window.location.assign() and window.location.replace() methods, to set the URL of a new page using the location object.

→ **window.location.href property:**

→ The first way is to use the window.location.href property. This property contains information about the current URL of the page, and it can be used to redirect the user to a new page.
→ Syntax: window.location.href = "new_url";
→ **window.location.replace property:**
→ Another way for redirecting forcefully to another page is to use the window.location.replace property. This method helps in replacing the current page in the browser's history with another page, but here the user will not be able to return back to the original page.
→ Syntax: window.location.replace("new_url");
→ **window.location.assign method:**
→ In this method, we use the window.location.assign method which is used to add a new page to the browser's history allowing the users to return back to the original page browsed by the user.
→ Syntax: window.location.assign("new_url");

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

→ **The use of innerHTML very slow:** The process of using innerHTML is much slower as its contents as slowly built, also already parsed contents and elements are also re-parsed which takes time.
→ **Can break the document:** There is no proper validation provided by innerHTML, so any valid HTML code can be used. This may break the document of JavaScript. Even broken HTML can be used, which may lead to unexpected problems.
→ **Security Risks:** Using innerHTML` to manipulate content with user input can expose your application to cross-site scripting (XSS)

vulnerabilities. If you're injecting user-generated content into the HTML using `innerHTML`, you need to ensure that the content is properly sanitized to prevent malicious scripts from being executed.

→ **Preserves event handlers attached to any DOM elements:** The event handlers do not get attached to the new elements created by setting innerHTML automatically. To do so one has to keep track of the event handlers and attach it to new elements manually. This may cause a memory leak on some browsers.

→ **Browser Inconsistencies:** While innerHTML is widely supported, there can be slight variations in behavior across different browsers. It's essential to test and ensure consistent behavior, especially when working with more complex HTML structures.