MODULE – JAVASCRIPT(BASIC & DOM)

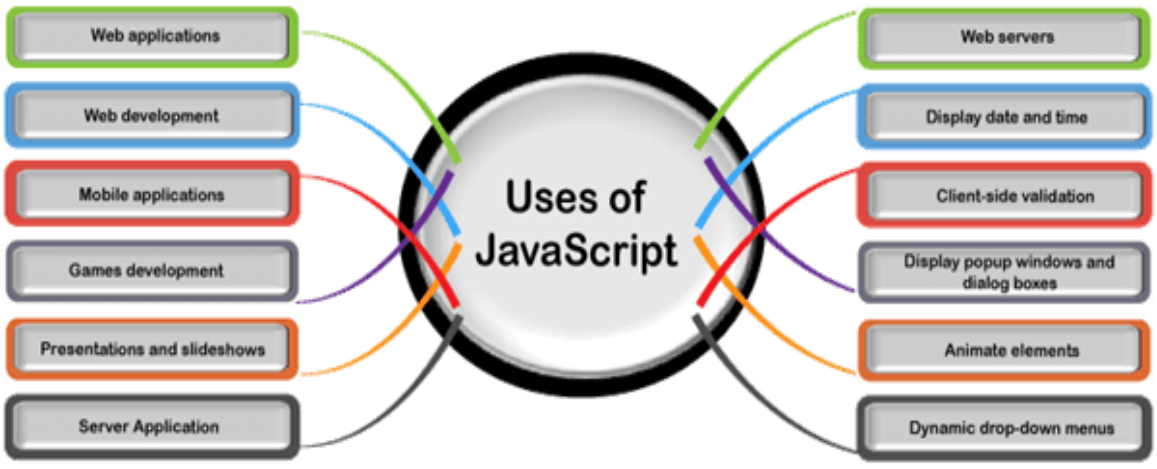
1.What is JavaScript?

JavaScript is a scripting or programming language that allows you to implement complex features on web pages

— every time a web page does more than just sit there and display static information for you to look at

— displaying timely content updates, interactive maps, animated 2D/3D graphics, scrolling video jukeboxes, etc.

— you can bet that JavaScript is probably involved. It is the third layer of the layer cake of standard web technologies, two of which (HTML and CSS) we have covered in much more detail in other parts of the Learning Area.



2.What is the use of isNaN function?

-The isNaN() function determines whether a value is NaN when converted to a number. Because coercion inside the isNaN() function can be surprising, you may alternatively want to use Number.isNaN()

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title>title</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
  <script>
    var denominator = 0;
    var number = 0;

    if ( isNaN( number / denominator ) ) {
       document.write('NaN detected. Bad math operation. ');
    } else {
       document.write('Answer is ' + String(number / denominator) );
    }

  </script>
  </body>
</html>
```

3.What is negative Infinity?

-Number.NEGATIVE_INFINITY is a special numeric value that is returned when an arithmetic operation or mathematical function generates a negative value greater than the largest representable number in JavaScript (i.e., more negative than -Number.MAX_VALUE).

-JavaScript displays the NEGATIVE_INFINITY value as -Infinity. This value behaves mathematically like infinity; for example, anything multiplied by infinity is infinity, and anything divided by

infinity is zero. In ECMAScript v1 and later, you can also use -Infinity instead of Number.NEGATIVE_INFINITY.

```
<!DOCTYPE html>

<html>

<head>

    <title> JavaScriptSQRTFunction </title>

</head>

<body>

  <h1> JavaScriptSQRTFunction </h1>

  <p id = "Pos"></p>

  <p id = "Neg"></p>

  <p id = "Dec"></p>

  <p id = "Neg_Dec"></p>

  <p id = "Str"></p>

  <p id = "Null"></p>

  <script>

    document.getElementById("Pos").innerHTML = Math.sqrt(10);

    document.getElementById("Neg").innerHTML = Math.sqrt(-10);

    document.getElementById("Dec").innerHTML = Math.sqrt(14.05);

    document.getElementById("Neg_Dec").innerHTML = Math.sqrt(-6.05);

    document.getElementById("Str").innerHTML = Math.sqrt("JavaScript");

    document.getElementById("Null").innerHTML = Math.sqrt(null);

  </script>

</body>

</html>
```

4.Which company developed JavaScript?

JavaScript was invented by **Brendan Eich** in 1995.

It was developed for **Netscape 2**, and became the **ECMA-262** standard in 1997.

After Netscape handed JavaScript over to ECMA, the Mozilla foundation continued to develop JavaScript for the Firefox browser. Mozilla's latest version was 1.8.5. (Identical to ES5).

Internet Explorer (IE4) was the first browser to support ECMA-262 Edition 1 (ES1).

5.What are undeclared and undefined variables?

Undefined variable means a variable has been declared but does not have a value. Undeclared variable means that the variable does not exist in the program at all.

Undefined:
It occurs when a variable has been declared but has not been assigned with any value. Undefined is not a keyword.

Undeclared:
It occurs when we try to access any variable that is not initialized or declared earlier using *var* or *const keyword*. If we use *'typeof'* operator to get the value of an undeclared variable, we will face the *runtime error* with return value as "undefined"**. The scope of the undeclared variables is always global.**

EXAMPLE:

```
// Declared
let x;
console.log(x);

// Declared & Assigned
let y = null;
console.log(y);
```

6.Write the code for adding new elements dynamically?

New elements can be dynamically created in JavaScript with the help of createElement() method. The attributes of the created element can be set using the setAttribute() method. The examples given below would demonstrate this approach.

EXAMPLE

```
<head>
    <script type="text/javascript" src="js/createElement.js"></script>
    <style type="text/css">
        #specialDiv {
            color: red;
        }

        .colorBlue {
            color: blue;
        }

        .buttons {
            padding: 10px;
            border: 1px solid grey;
             border-radius: 4px;
        }
    </style>
</head>
```

7.What is the difference between ViewState and SessionState?

-The basic difference between these two is that the ViewState is to manage state at the client's end, making state management easy for end-user while SessionState manages state at the server's end, making it easy to manage content from this end too.

ViewState	SessionState
▪ Maintained at page level only.	▪ Maintained at session level.
▪ View state can only be visible from a single page and not multiple pages.	▪ Session state value availability is across all pages available in a user session.
▪ It will retain values in the event of a postback operation occurring.	▪ In session state, user data remains in the server. Data is available to user until the browser is closed or there is session expiration.
▪ Information is stored on the client's end only.	▪ Information is stored on the server.
▪ used to allow the persistence of page-instance-specific data.	▪ used for the persistence of user-specific data on the server's end.
▪ ViewState values are lost/cleared when new page is loaded.	▪ SessionState can be cleared by programmer or user or in case of timeouts.

Usage:

- **SessionState: It can be used to store information that you wish to access on different web pages.**
- **ViewState It can be used to store information that you wish to access from same web page.**

8.What is === operator?

The strict equality (===) operator checks whether its two operands are equal, returning a Boolean result. Unlike the equality operator, the strict equality operator always considers operands of different types to be different.

Strict equality (===)

The strict equality (===) operator checks whether its two operands are equal, returning a Boolean result. Unlike the equality operator, the strict equality operator always considers operands of different types to be different.

```
<html>
<body>
    Demonstrating === operator in javascript
    </br>
    <script type="text/javascript">
      var var1 = 0;
      var var2 = false;
      var result = var1 === var2; // Strict comparison of integer and boolean
      document.write("Triple equals operator for integer and boolean: " + result);
      document.write("</br>");
      document.write("Triple equals operator integer and string: " + (2 === '2'));
// Strict comparison of integer and string
      document.write("</br>");
      document.write("Tripe equals operator integer and integer: " + (2 === 2)); //
// Strict comparison of integer and integer
    </script>
</body>

</html>
```

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

The document.getElementById() method is used to return the element in the document with the "id" attribute and the "className" attribute can be used to change/append the class of the element.

Syntax:
document.getElementById('myElement').className = "myclass";

Example 1: In this code change the class of the button from "default" to "changedClass" using the onclick event which in turn changes the background color of the button from RED to GREEN.

```
style>
    .default{
        background-color: red;
    }
    .changedClass{
        background-color: green;
    }
</style>
```

```
<body>
    <h1 style="color:green">
        GeeksforGeeks
    </h1>
    <h2>
        Change class name of element
    </h2>
    <button class="default" onclick="changeClass()" id="myButton">
        Click Here!</button><br>
    <p id="myPara">
        Old class name: default
    </p>
    <script>
        function changeClass() {
            document.getElementById('myButton')
                    .className = "changedClass";
            var button_class = document.getElementById('myButton')
                                       .className;
            document.getElementById('myPara')
                    .innerHTML = "New class name: "
                               + button_class;
        }
    </script>
</body>
```

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

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.

Parameters:

- **file_name**: It's a string, a buffer, a URL, or a file description integer that specifies the location of the file to be written. When you use a file descriptor, it will function similarly to the fs. write() method.
- **data**: The data that will be sent to the file is a string, Buffer, TypedArray, or DataView.
- **options:** It's a string or object that may be used to indicate optional output options. It includes three more parameters that may be selected.
- **encoding**: It's a string value that indicates the file's encoding. 'utf8' is the default setting.

- **mode**: The file mode is specified by an integer number called mode. 0o666 is the default value.
- **flag**: This is a string that indicates the file-writing flag. 'w' is the default value.
- **callback**: This function gets invoked when the method is run.
- **err**: If the process fails, this is the error that will be thrown.

EXAMPLE:

```
var fs = require("fs");
console.log(" Writing into an file ");

// Sample.txt is an empty file
fs.writeFile(
  "sample.txt",
  "Let's write a few sentences in the file",
  function (err) {
    if (err) {
      return console.error(err);
    }
    // If no error the remaining code executes
    console.log(" Finished writing ");
    console.log("Reading the data that's written");

    // Reading the file
    fs.readFile("sample.txt", function (err, data) {
      if (err) {
        return console.error(err);
      }
      console.log("Data read : " + data.toString());

    });
  }
);
```

11.What are all the looping structures in JavaScript?

-Loops can execute a block of code a number of times.

JavaScript Loops

-Loops are handy, if you want to run the same code over and over again, each time with a different value.Often this is the case when working with arrays:

JavaScript supports different kinds of loops:

- **for - loops through a block of code a number of times**
- **for/in - loops through the properties of an object**
- **for/of - loops through the values of an iterable object**
- **while - loops through a block of code while a specified condition is true**
- **do/while - also loops through a block of code while a specified condition is true**

```
<!DOCTYPE HTML>
<HTML>
        <HEAD>
                <TITLE>For Loops</TITLE>

                <SCRIPT LANGUAGE = "Javascript">

                var answer=0;
                var start_value = 1;
                var end_value = 11;

                for ( start_value; start_value < end_value; start_value++ ) {
                        answer = answer + start_value;

                }

                document.write("answer= " + answer);

                </SCRIPT>
        </HEAD>

        <BODY>


        </BODY>
</HTML>
```

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

In JavaScript **parseInt() function (or a method) is used to convert the passed-in string parameter or value to an integer value itself**. This function returns an integer of the base which is specified in the second argument of the parseInt() function.

Example 1: In this example, we will convert a string value to an integer using the parseInt() function with no second parameter.

- javascript

```
function convertStoI() {
    var a = "100";
    var b = parseInt(a);
    console.log("Integer value is" + b);
    var d = parseInt("3 11 43");

    console.log('Integer value is ' + d);
}
convertStoI();
```

Output:
Integer value is 100

Integer value is 3

13.What is the function of the delete operator?

The **delete** operator removes a property from an object. If the property's value is an object and there are no more references to the object, the object held by that property is eventually released automatically.

EXAMPLE:

```
> var employeeDetails = {
      name: "Mayank",
      age: 20,
      designation: "Architect"
  }

  delete window.employeeDetails;
< false
```

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

JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

Alert Box

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

Syntax

window.alert("*sometext*");

The window.alert() method can be written without the window prefix.

Example

```
alert("I am an alert box!");
```

Confirm Box

A confirm box is often used if you want the user to verify or accept something.

When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns **true**. If the user clicks "Cancel", the box returns **false**.

Syntax

window.confirm("*sometext*");

The window.confirm() method can be written without the window prefix.

Example

```
if (confirm("Press a button!")) {
  txt = "You pressed OK!";
} else {
  txt = "You pressed Cancel!";
}
```

Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

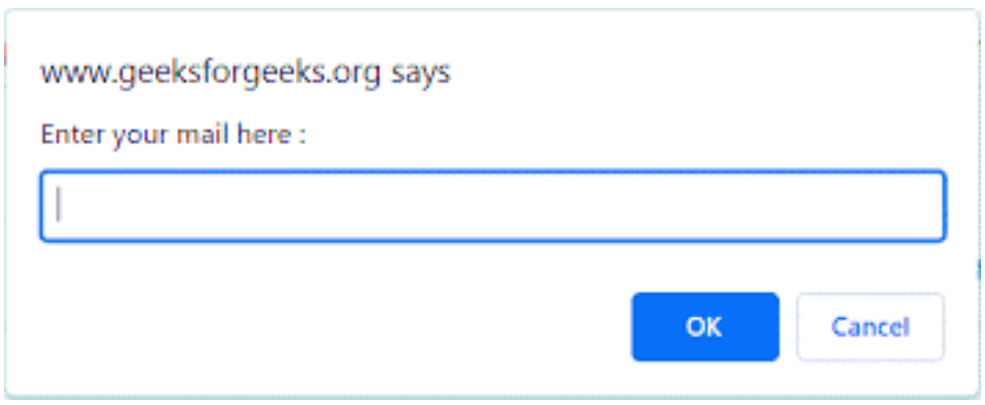
Syntax

window.prompt("*sometext*","*defaultText*");

The window.prompt() method can be written without the window prefix.

Example

```
let person = prompt("Please enter your name", "Harry Potter");
let text;
if (person == null || person == "") {
  text = "User cancelled the prompt.";
} else {
  text = "Hello " + person + "! How are you today?";
}
```



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

JavaScript void 0 means returning undefined (void) as a primitive value. You might come across the term "JavaScript:void(0)" while going through HTML documents. It is used to prevent any side effects caused while inserting an expression in a web page.

If we split this up, we have javascript: and void(0). Let's look at each part in more detail.

javascript:

This is referred to as a Pseudo URL**. When a browser receives this value as the value of href on an anchor tag, it interprets the JS code that follows the colon (:) rather than treating the value as a referenced path.**
For example:

```
<a href="javascript:console.log('javascript');alert('javascript')">Link</a>
```

When "Link" is clicked, here is the result:

As seen above, the browser does not treat href as a referenced path. Instead, it treats it as some JavaScript code starting after "javascript:" and separated by semi-colons.

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

Approach: **We can use *window.location* property inside the *script* tag to forcefully load another page in Javascript. It is a reference to a Location object that is it represents the current location of the document. We can change the URL of a window by accessing it.**

Syntax:

```
<script>
   window.location = <Path / URL>
</script>
```

EXAMPLE

```
<!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">
    <title> New Page </title>
</head>

<body>
    <h3>This is the new loaded page</h3>
</body>

</html>
```

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

Inner HTML is slow

Inner HTML is slow because when we use the inner HTML property in the code it allows us to change using the JavaScript language. It is very slow because as inner HTML already parses the content even we have to parse the content again so that's why it takes time.

Event handlers attached to any DOM element are preserved

When we have used the event handlers then the event handlers are not automatically attached to the new elements created by innerHTML. To change that, we have to track the event handlers and manually attach them to a new element.

It means that first, we have to fetch the element property through innerHTML, and then we have to attach them to a new element.

Example

Let's try to understand with an appropriate example −

```
<!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">
   <title>Basic of HTML</title>
</head>
<body>
   <p id= 'demo'>Hello</p>
   <button onclick="Change()">Change</button>
   <script>
      function Change()
      {
         let p = document.getElementById('demo');
         p.innerHTML = '<span>Hello World</span>';
      }
   </script>
</body>
</html>
```

As you can see in the above program first we fetch the element which is already preserved and then we manually attach them to the new element **span.** you can see the changes in the output screenshots before the event handler and after the event handler.

Some other disadvantages of HTML are −

Replacement is done everywhere

When innerHTML property is used to modify, all the DOM nodes will have to be parsed and created again.

It is not possible to append innerHTML

In JavaScript, '+=' is commonly used for appending. However, when using innerHTML to append to an HTML tag, the entire tag is re-parsed.

Example

```
<spam id="tp">Tutorials Point</p>
subject = document.getElementById('#tp')

// The whole "tp" tag is reparsed
subject.innerHTML += '<span> Tutorix </span>'
```

Thank you!!!!!