UNIT 5: JAVASCRIPT

CS-04 NETWORKING & INTERNET ENVIRONMENT

- **INTRODUCTION TO JAVASCRIPT**
- **VARIABLES**
- **OPERATORS**
- **CONDITIONAL STATEMENTS**
- **LOOPING STATEMENTS**
- **BREAK AND CONTINUE STATEMENTS**
- **DIALOG BOXES**
- **ARRAYS**
- **USER DEFINED FUNCTIONS(UDF)**
- **BUILT IN FUNCTIONS**
- **EVENTS**
- **DOM OBJECT**
- **HISTORY OBJECT**
- **FORM AND EMAIL VALIDATION**

: ASSIGNMENT UNIT 4:

1. **Explain javascript with suitable example and its usage**
2. **Explain variable and operators(any 3 types) in JavaScript**
3. **Explain conditional statement in JS.**
4. **Explain Looping statement in JS.**
5. **Explain Break and continue statement with suitable example**
6. **Explain types of dialog box in JS**
7. **Explain UDF with its type and example**
8. **Explain String Funtions in JS**
9. **Explain Math Function in JS**
10. **Explain Date function in JS**
11. **Explain any 4 events with suitable example.**
12. **Explain DOM.**
13. **Explain History Example.**

INTRODUCTION TO JAVASCRIPT

- JavaScript is the most popular scripting language on the internet, and works in all major browsers, such as Internet Explorer, Firefox, Chrome, Opera, and Safari.
- JavaScript is a language that is used to make web pages interactive.
- It runs on your user's computer and so does not require constant downloads from your web site.
- JavaScript was designed to add interactivity to HTML pages.
- JavaScript is a scripting language.
- A scripting language is a lightweight programming language.
- JavaScript is usually embedded directly into HTML pages.
- JavaScript is an interpreted language (means that scripts execute without preliminary
- compilation).

- Everyone can use JavaScript without purchasing a license.
- Using JavaScript user can add events on your website.
- Using JavaScript user can check the data before it is submitted to the server.
- **JavaScript** is a lightweight, interpreted **programming** language. It is designed for creating network-centric applications. It is complimentary to and integrated with Java. **JavaScript** is very easy to implement because it is integrated with HTML. It is open and cross-platform.

How to write JavaScript code into html page?

- The HTML <script> tag is used to insert a JavaScript into an HTML page.
- Inside <script> tag type attribute is used to specify the scripting language.
- JavaScript can write into two sections either between <HEAD> tag.
- If user writes code in <head> section then JavaScript code will be executed depends on user's requirement.
- Syntax:

```
<script type = "text/javascript">
    var money;
    var name;
</script>
```

Applications of JavaScript Programming

- JavaScript is one of the most widely used programming languages (Front-end as well as Back-end). It has it's presence in almost every area of software development. I'm going to list few of them here:
- **Client side validation** - This is really important to verify any user input before submitting it to the server and JavaScript plays an important role in validating those inputs at front-end itself.
- **Manipulating HTML Pages** - JavaScript helps in manipulating HTML page on the fly. This helps in adding and deleting any HTML tag very easily using JavaScript and modify your HTML to change its look and feel based on different devices and requirements.
- **User Notifications** - You can use JavaScript to raise dynamic pop-ups on the webpages to give different types of notifications to your website visitors.
- **Back-end Data Loading** - JavaScript provides Ajax library which helps in loading back-end data while you are doing some other processing. This really gives an amazing experience to your website visitors.
- **Presentations** - JavaScript also provides the facility of creating presentations which gives website look and feel. JavaScript provides RevealJS and BespokeJS libraries to build a web-based slide presentations.
- **Server Applications** - Node JS is built on Chrome's JavaScript runtime for building fast and scalable network applications. This is an event based library which helps in developing very sophisticated server applications including Web Servers.

JAVASCRIPT DATATYPES AND VARIABLES

- One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.
- JavaScript allows you to work with three primitive data types −
- Numbers, eg. 123, 120.50 etc.
- Strings of text e.g. "This text string" etc.
- Boolean e.g. true or false.

- JavaScript also defines two trivial data types, null and undefined, each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as object. We will cover objects in detail in a separate chapter.
- **Note** − JavaScript does not make a distinction between integer values and floating-point values. All numbers in JavaScript are represented as floating-point values. JavaScript represents numbers using the 64-bit floating-point format defined by the IEEE.

JavaScript Variables

- Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.
- Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the *var* keyword.
- Syntax:

```
<script type = "text/javascript">
    var money;
    var name;
</script>
```

- **Note** − Use the var keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.
- JavaScript is untyped language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

JavaScript Variable Scope

- The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.
- **Global Variables** – A global variable has global scope which means it can be defined anywhere in your JavaScript code.
- **Local Variables** – A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.
- Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable.

JavaScript Variable Names

- While naming your variables in JavaScript, keep the following rules in mind.
- You should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, break or boolean variable names are not valid.
- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, 123test is an invalid variable name but _123test is a valid one.
- JavaScript variable names are case-sensitive. For example, Name and name are two different variables.

JAVASCRIPT OPRATORS

- Let us take a simple expression 4 + 5 is equal to 9. Here 4 and 5 are called operands and '+' is called the operator. JavaScript supports the following types of operators.
- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Arithmetic Operators

- JavaScript supports the following arithmetic operators
- Assume variable A holds 10 and variable B holds 20.
- Note − Addition operator (+) works for Numeric as well as Strings. e.g. "a" + 10 will give "a10".

Operator & Description
+ (Addition) Adds two operands **Ex:** A + B will give 30
- (Subtraction) Subtracts the second operand from the first **Ex:** A - B will give -10
*** (Multiplication)** Multiply both operands **Ex:** A * B will give 200
/ (Division) Divide the numerator by the denominator **Ex:** B / A will give 2
% (Modulus) Outputs the remainder of an integer division **Ex:** B % A will give 0
++ (Increment) Increases an integer value by one **Ex:** A++ will give 11
-- (Decrement) Decreases an integer value by one **Ex:** A-- will give 9

Comparison Operators

JavaScript supports the following comparison operators
Assume variable A holds 10 and variable B holds 20.

Operator & Description
= = (Equal) Checks if the value of two operands are equal or not, if yes, then the condition becomes true. **Ex:** (A == B) is not true.
!= (Not Equal) Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true. **Ex:** (A != B) is true.
> (Greater than) Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true. **Ex:** (A > B) is not true.
< (Less than) Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true. **Ex:** (A < B) is true.
>= (Greater than or Equal to) Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true. **Ex:** (A >= B) is not true.
<= (Less than or Equal to) Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true. **Ex:** (A <= B) is true.

Logical Operators

JavaScript supports the following logical operators
Assume variable A holds 10 and variable B holds 20.

Operator & Description
&& (Logical AND) If both the operands are non-zero, then the condition becomes true. **Ex:** (A && B) is true.
|| (Logical OR) If any of the two operands are non-zero, then the condition becomes true. **Ex:** (A || B) is true.
! (Logical NOT) Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false. **Ex:** ! (A && B) is false.

Bitwise Operators

JavaScript supports the following bitwise operators

Assume variable A holds 2 and variable B holds 3, then −

Operator & Description
& (Bitwise AND) It performs a Boolean AND operation on each bit of its integer arguments. **Ex:** (A & B) is 2.
| (BitWise OR) It performs a Boolean OR operation on each bit of its integer arguments. **Ex:** (A | B) is 3.
^ (Bitwise XOR) It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both. **Ex:** (A ^ B) is 1.
~ (Bitwise Not) It is a unary operator and operates by reversing all the bits in the operand. **Ex:** (~B) is -4.
<< (Left Shift) It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on. **Ex:** (A << 1) is 4.
>> (Right Shift) Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand. **Ex:** (A >> 1) is 1.
>>> (Right shift with Zero) This operator is just like the >> operator, except that the bits shifted in on the left are always zero. **Ex:** (A >>> 1) is 1.

Assignment Operators

JavaScript supports the following assignment operators

Operator & Description
= (Simple Assignment) Assigns values from the right side operand to the left side operand **Ex:** C = A + B will assign the value of A + B into C
+= (Add and Assignment) It adds the right operand to the left operand and assigns the result to the left operand. **Ex:** C += A is equivalent to C = C + A
−= (Subtract and Assignment) It subtracts the right operand from the left operand and assigns the result to the left operand. **Ex:** C -= A is equivalent to C = C - A
***= (Multiply and Assignment)** It multiplies the right operand with the left operand and assigns the result to the left operand. **Ex:** C *= A is equivalent to C = C * A
/= (Divide and Assignment) It divides the left operand with the right operand and assigns the result to the left operand. **Ex:** C /= A is equivalent to C = C / A
%= (Modules and Assignment) It takes modulus using two operands and assigns the result to the left operand. **Ex:** C %= A is equivalent to C = C % A

Miscellaneous Operator

- We will discuss two operators here that are quite useful in JavaScript: the **conditional operator** (? :) and the **typeof operator**.
- **Conditional Operator (? :)**
- The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

Operator and Description
? : (Conditional) If Condition is true? Then value X : Otherwise value Y

JAVASCRIPT - IF...ELSE STATEMENT

- While writing a program, there may be a situation when you need to adopt one out of a given set of paths. In such cases, you need to use conditional statements that allow your program to make correct decisions and perform right actions.
- JavaScript supports conditional statements which are used to perform different actions based on different conditions. Here we will explain the if..else statement.
- JavaScript supports the following forms of if..else statement
  - if statement
  - if...else statement
  - if...else if... statement.

if statement

- The if statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax

- The syntax for a basic if statement is as follows

```
if (expression) {
   Statement(s) to be executed if expression is true
}
```

- Here a JavaScript expression is evaluated. If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement would be not executed. Most of the times, you will use comparison operators while making decisions.

Example

```
<script type = "text/javascript">
        var age = 20;
        if( age > 18 ) {
           document.write("<b>Qualifies for driving</b>");
        }
</script>
```

if...else statement

- The 'if...else' statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.

Syntax

- The syntax for a basic if statement is as follows

```
if (expression) {
   Statement(s) to be executed if expression is true
} else {
   Statement(s) to be executed if expression is false
}
```

- Here JavaScript expression is evaluated. If the resulting value is true, the given statement(s) in the 'if' block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

Example

```
<script type = "text/javascript">
        var age = 15;
        if( age > 18 ) {
           document.write("<b>Qualifies for driving</b>");
        } else {
           document.write("<b>Does not qualify for driving</b>");
        }
</script>
```

if...else if... statement

- The if...else if... statement is an advanced form of if…else that allows JavaScript to make a correct decision out of several conditions.

Syntax

- The syntax of an if-else-if statement is as follows −

```
if (expression 1) {
   Statement(s) to be executed if expression 1 is true
} else if (expression 2) {
   Statement(s) to be executed if expression 2 is true
} else if (expression 3) {
   Statement(s) to be executed if expression 3 is true
} else {
   Statement(s) to be executed if no expression is true
}
```

- There is nothing special about this code. It is just a series of if statements, where each if is a part of the else clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the else block is executed.

Example

```
<script type = "text/javascript">
        var book = "maths";
        if( book == "history" ) {
           document.write("<b>History Book</b>");
        } else if( book == "maths" ) {
           document.write("<b>Maths Book</b>");
        } else if( book == "economics" ) {
           document.write("<b>Economics Book</b>");
        } else {
           document.write("<b>Unknown Book</b>");
        }
    </script>
```

JAVASCRIPT - SWITCH CASE

- You can use multiple if...else…if statements, as in the previous chapter, to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.
- Starting with JavaScript 1.2, you can use a switch statement which handles exactly this situation, and it does so more efficiently than repeated if...else if statements.

Syntax

- The objective of a switch statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

```
switch (expression) {
  case condition 1: statement(s)
  break;
  case condition 2: statement(s)
  break;
  ...
  case condition n: statement(s)
  break;
  default: statement(s)
}
```

- The break statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

Example

```
<script type = "text/javascript">
        var grade = 'A';
        document.write("Entering switch block<br />");
        switch (grade) {
           case 'A': document.write("Good job<br />");
           break;

           case 'B': document.write("Pretty good<br />");
           break;

           case 'C': document.write("Passed<br />");
```

```
        break;

        case 'D': document.write("Not so good<br />");
        break;

        case 'F': document.write("Failed<br />");
        break;

        default:  document.write("Unknown grade<br />")
      }
      document.write("Exiting switch block");
</script>
```

THE WHILE LOOP

- The most basic loop in JavaScript is the while loop which would be discussed in this chapter. The purpose of a while loop is to execute a statement or code block repeatedly as long as an expression is true. Once the expression becomes false, the loop terminates.

Syntax

- The syntax of while loop in JavaScript is as follows −

while (expression) {
Statement(s) to be executed if expression is true
}

Example:

```
<script type = "text/javascript">
        var count = 0;
        while (count < 10) {
          document.write("Current Count : " + count + "<br />");
          count++;
        }
</script>
```

THE DO...WHILE LOOP

- The do...while loop is similar to the while loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is false.

Syntax

- The syntax for do-while loop in JavaScript is as follows

do {
Statement(s) to be executed;
} while (expression);

- ***Note −*** *Don't miss the semicolon used at the end of the do...while loop.*

Example

```
<script type = "text/javascript">
        var count = 0;
        do {
          document.write("Current Count : " + count + "<br />");
          count++;
        }
        while (count < 5);
```

```
</script>
```

JAVASCRIPT - FOR LOOP

- The 'for' loop is the most compact form of looping. It includes the following three important parts.
- The **loop initialization** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The **test statement** which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
- The **iteration statement** where you can increase or decrease your counter.
- You can put all the three parts in a single line separated by semicolons.

Syntax

- The syntax of for loop is JavaScript is as follows

```
for (initialization; test condition; iteration statement) {
   Statement(s) to be executed if test condition is true
}
```

Example

```
<script type = "text/javascript">
        var count;
        for(count = 0; count < 10; count++) {
          document.write("Current Count : " + count );
          document.write("<br />");
        }
</script>
```

JAVASCRIPT - LOOP CONTROL

- JavaScript provides full control to handle loops and switch statements. There may be a situation when you need to come out of a loop without reaching its bottom. There may also be a situation when you want to skip a part of your code block and start the next iteration of the loop.
- To handle all such situations, JavaScript provides break and continue statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

The break Statement

- The break statement, which was briefly introduced with the switch statement, is used to exit a loop early, breaking out of the enclosing curly braces.

The continue Statement

- The continue statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block. When a continue statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop.

JAVASCRIPT – FUNCTIONS

- A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.
- Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions. You must have seen functions like alert() and write() in the earlier chapters. We were using these functions again and again, but they had been written in core JavaScript only once.

- JavaScript allows us to write our own functions as well. This section explains how to write your own functions in JavaScript.

Function Definition

- Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the function keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

Syntax

- The basic syntax is shown here.

```
<script type = "text/javascript">
   function functionname(parameter-list) {
     statements;
   }
</script>
```

Example

```
<script type = "text/javascript">
    function sayHello() {
        document.write("Hello World" );
    }
</script>
```

Function Parameters

- Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

Syntax/Example

```
<script type = "text/javascript">
      function sayHello(name, age) {
         document.write (name + " is " + age + " years old.");
      }
    </script>
```

The return Statement

- A JavaScript function can have an optional return statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.
- For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

Syntax/Example

```
<script type = "text/javascript">
      function concatenate(first, last) {
         var full;
         full = first + last;
         return full;
      }
</script>
```

JAVASCRIPT - DIALOG BOXES

- JavaScript supports three important types of dialog boxes. These dialog boxes can be used to raise and alert, or to get confirmation on any input or to have a kind of input from the users. Here we will discuss each dialog box one by one.

Alert Dialog Box

- An alert dialog box is mostly used to give a warning message to the users. For example, if one input field requires to enter some text but the user does not provide any input, then as a part of validation, you can use an alert box to give a warning message.
- Nonetheless, an alert box can still be used for friendlier messages. Alert box gives only one button "OK" to select and proceed.

Syntax/Example

```
<script type = "text/javascript">
        alert ("This is a warning message!");
</script>
```

Confirmation Dialog Box

- A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: OK and Cancel.
- If the user clicks on the OK button, the window method confirm() will return true. If the user clicks on the Cancel button, then confirm() returns false. You can use a confirmation dialog box as follows.

Syntax/Example

```
<script type = "text/javascript">
        function getConfirmation() {
          var retVal = confirm("Do you want to continue ?");
          if( retVal == true ) {
            document.write ("User wants to continue!");
            return true;
          } else {
            document.write ("User does not want to continue!");
            return false;
          }
        }
</script>
```

Prompt Dialog Box

- The prompt dialog box is very useful when you want to pop-up a text box to get user input. Thus, it enables you to interact with the user. The user needs to fill in the field and then click OK.
- This dialog box is displayed using a method called prompt() which takes two parameters:
  - As label which you want to display in the text box
  - A default string to display in the text box.
- This dialog box has two buttons: OK and Cancel. If the user clicks the OK button, the window method prompt() will return the entered value from the text box. If the user clicks the Cancel button, the window method prompt() returns null.

Syntax/Example

```
<script type = "text/javascript">
        var retVal = prompt("Enter your name : ", "your name here");
        document.write("You have entered : " + retVal);
</script>
```

JAVASCRIPT - THE ARRAYS OBJECT

- The Array object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Syntax

- Use the following syntax to create an Array object −
  var fruits = new Array("apple", "orange", "mango");
- The Array parameter is a list of strings or integers. When you specify a single numeric parameter with the Array constructor, you specify the initial length of the array. The maximum length allowed for an array is 4,294,967,295.

Example

```
<script type = "text/javascript">
      var arr = new Array( 10, 20, 30 );
</script>
```

IN-BUILT STRING FUNCTIONS

1. charAt():

- charAt() is a method that returns the character from the specified index.
- Characters in a string are indexed from left to right. The index of the first character is 0, and the index of the last character in a string, called stringName, is stringName.length- 1
- Syntax: string.charAt(index);
- Argument Details
  - **index** − An integer between 0 and 1 less than the length of the string.
- Return Value
  - Returns the character from the specified index.

2. concat():

- This method adds two or more strings and returns a new single string.
- Syntax: string.concat(string2, string3[, ..., stringN]);
- Argument Details
  - **string2...stringN** − These are the strings to be concatenated.
- Return Value
  - Returns a single concatenated string.

3. indexOf():

- This method returns the index within the calling String object of the first occurrence of the specified value, starting the search at fromIndex or -1 if the value is not found.
- Syntax: string.indexOf(searchValue[, fromIndex]);
- Argument Details
  - **searchValue** − A string representing the value to search for.
  - **fromIndex** − The location within the calling string to start the search from. It can be any integer between 0 and the length of the string. The default value is 0.

4. lastIndexOf():

- This method returns the index within the calling String object of the last occurrence of the specified value, starting the search at fromIndex or -1 if the value is not found.
- Syntax: string.lastIndexOf(searchValue[, fromIndex]);
- Argument Details
  - **searchValue** − A string representing the value to search for.
  - **fromIndex** − The location within the calling string to start the search from. It can be any integer between 0 and the length of the string. The default value is 0.
- Return Value

- Returns the index of the last found occurrence, otherwise -1 if not found.

5. replace():

- This method finds a match between a regular expression and a string, and replaces the matched substring with a new substring.
- Syntax: string.replace(substr, newSubStr);
- Argument Details
  - **substr** − A String that is to be replaced by newSubStr.
  - **newSubStr** − The String that replaces the substring received from parameter

6. search():

- This method executes the search for a match between a regular expression and this String object.
- Syntax: string.search(regexp);
- Argument Details
  - **regexp** − A regular expression object. If a non-RegExp object obj is passed, it is implicitly converted to a RegExp by using new RegExp(obj).
- Return Value
  - If successful, the search returns the index of the regular expression inside the string. Otherwise, it returns -1.

7. substr():

- This method returns the characters in a string beginning at the specified location through the specified number of characters.
- Syntax: string.substr(start[, length]);
- Argument Details
  - **start** − Location at which to start extracting characters (an integer between 0 and one less than the length of the string).
  - **length** − The number of characters to extract.
- **Note** − If start is negative, substr uses it as a character index from the end of the string.
- Return Value
  - The substr() method returns the new sub-string based on given parameters.

8. substring():

- This method returns a subset of a String object.
- Syntax: string.substring(indexA, [indexB])
- Argument Details
  - **indexA** − An integer between 0 and one less than the length of the string.
  - **indexB** − (optional) An integer between 0 and the length of the string.
- Return Value
  - The substring method returns the new sub-string based on given parameters.

9. toLowerCase():

- This method returns the calling string value converted to lowercase.
- Syntax: string.toLowerCase( )
- Return Value
  - Returns the calling string value converted to lowercase.

10. toUpperCase():

- This method returns the calling string value converted to uppercase.
- Syntax: string.toUpperCase( )
- Return Value
  - Returns a string representing the specified object.

IN-BUILD MATH FUNCTION

1. abs():
- This method returns the absolute value of a number.
- Syntax
  - Math.abs(x) ;
- Return Value
  - Returns the absolute value of a number.

2. ceil():
- This method returns the smallest integer greater than or equal to a number.
- Syntax
  - Math.ceil(x) ;
- Return Value
  - Returns the smallest integer greater than or equal to a number.

3. floor():
- This method returns the largest integer less than or equal to a number.
- Syntax
  - Math.floor(x) ;
- Return Value
  - Returns the largest integer less than or equal to a number x.

4. max():
- This method returns the largest of zero or more numbers. If no arguments are given, the results is –Infinity.
- Syntax
  - Math.max(value1, value2, ... valueN) ;
- Return Value
  - Returns the largest of zero or more numbers.

5. min():
- This method returns the smallest of zero or more numbers. If no arguments are given, the results is +Infinity.
- Syntax
  - Math.min(value1, value2, ... valueN) ;
- Return Value
  - Returns the smallest of zero or more numbers.

6. pow():
- This method returns the base to the exponent power, that is, base ^{exponent}.
- Syntax
  - Math.pow(base, exponent) ;
- Return Value
  - Returns the base to the exponent power, that is, base ^{exponent}.

7. random():
- This method returns a random number between 0 (inclusive) and 1 (exclusive).
- Syntax
  - Math.random() ;
- Return Value
  - Returns a random number between 0 (inclusive) and 1 (exclusive).

8. round():
- This method returns the value of a number rounded to the nearest integer.
- Syntax
  - Math.round(x) ;

- Return Value
  - Returns the value of a number rounded to the nearest integer.

IN-BUILT DATE FUNCTION

1. getDate():

- Javascript date getDate() method returns the day of the month for the specified date according to local time. The value returned by getDate() is an integer between 1 and 31.
- Syntax
  - Date.getDate();
- Return Value
  - Returns today's date and time.

2. getDay():

- Javascript date getDay() method returns the day of the week for the specified date according to local time. The value returned by getDay() is an integer corresponding to the day of the week: 0 for Sunday, 1 for Monday, 2 for Tuesday, and so on.
- Syntax
  - Date.getDay();
- Return Value
  - Returns the day of the week for the specified date according to local time.

3. getHours():

- Javascript Date getHours() method returns the hour in the specified date according to local time. The value returned by getHours() is an integer between 0 and 23.
- Syntax
  - Date.getHours();
- Return Value
  - Returns the hour in the specified date according to local time.

4. getMinutes():

- Javascript date getMinutes() method returns the minutes in the specified date according to local time. The value returned by getMinutes() is an integer between 0 and 59.
- Syntax
  - Date.getMinutes();
- Return Value
  - Returns the minutes in the specified date according to local time.

5. getMonth():

- Javascript date getMonth() method returns the month in the specified date according to local time. The value returned by getMonth() is an integer between 0 and 11. 0 corresponds to January, 1 to February, and so on.
- Syntax
  - Date.getMonth();
- Return Value
  - Returns the Month in the specified date according to local time.

6. getSeconds():

- Javascript date getSeconds() method returns the seconds in the specified date according to local time. The value returned by getSeconds() is an integer between 0 and 59.
- Syntax
  - Date.getSeconds();
- Return Value

- Returns the seconds in the specified date according to local time.

7. setTime():

- Javascript date setTime()method sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC.
- Syntax
  - Date.setTime(timeValue);
- **Note** − Parameters in the bracket are always optional.

8. setYear():

- Javascript date setYear() method sets the year for a specified date according to universal time.
- Syntax
  - Date.setYear(yearValue);

IN-BUILT ARRAY FUNCTION

1. join():

- Javascript array join() method joins all the elements of an array into a string.
- Syntax
  - array.join(separator);
- Parameter Details
  - **separator** − Specifies a string to separate each element of the array. If omitted, the array elements are separated with a comma.
- Return Value
  - Returns a string after joining all the array elements.

2. pop():

- Javascript array pop() method removes the last element from an array and returns that element.
- Syntax
  - array.pop();
- Return Value
  - Returns the removed element from the array.

3. push():

- Javascript array push() method appends the given element(s) in the last of the array and returns the length of the new array.
- Syntax
  - array.push(element1, ..., elementN);
- Return Value
  - Returns the length of the new array.

4. reverse():

- Javascript array reverse() method reverses the element of an array. The first array element becomes the last and the last becomes the first.
- Syntax
  - array.reverse();
- Return Value
  - Returns the reversed single value of the array.

5. sort():

- Javascript array sort() method sorts the elements of an array.
- Syntax
  - array.sort(element1, ..., elementN);
- Parameter Details
- Return Value
  - Returns a sorted array.

6. toString():

- Javascript array toString() method returns a string representing the source code of the specified array and its elements.
- Syntax
  - array.toString();
- Return Value
  - Returns a string representing the array.

JavaScript – Events

What is an Event ?

- JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.
- When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.
- Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.
- Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

Mouse Events

onclick — The event occurs when the user clicks on an element
oncontextmenu — User right-clicks on an element to open a context menu
ondblclick — The user double-clicks on an element
onmousedown — User presses a mouse button over an element
onmouseenter — The pointer moves onto an element
onmouseleave — Pointer moves out of an element
onmousemove — The pointer is moving while it is over an element
onmouseover — When the pointer is moved onto an element or one of its children
onmouseout — User moves the mouse pointer out of an element or one of its children
onmouseup — The user releases a mouse button while over an element

Keyboard Events

onkeydown — When the user is pressing a key down
onkeypress — The moment the user starts pressing a key
onkeyup — The user releases a key

Frame Events

onabort — The loading of a media is aborted
onbeforeunload — Event occurs before the document is about to be unloaded
onerror — An error occurs while loading an external file
onhashchange — There have been changes to the anchor part of a URL
onload — When an object has loaded
onpagehide — The user navigates away from a webpage
onpageshow — When the user navigates to a webpage
onresize — The document view is resized
onscroll — An element's scrollbar is being scrolled
onunload — Event occurs when a page has unloaded

Form Events

onblur — When an element loses focus

onchange — The content of a form element changes (for <input>, <select> and <textarea>)
onfocus — An element gets focus
onfocusin — When an element is about to get focus
onfocusout — The element is about to lose focus
oninput — User input on an element
oninvalid — An element is invalid

onreset — A form is reset

onsearch — The user writes something in a search field (for <input="search">)
onselect — The user selects some text (for <input> and <textarea>)

onsubmit — A form is submitted

Drag Events

ondrag — An element is dragged
ondragend — The user has finished dragging the element
ondragenter — The dragged element enters a drop target
ondragleave — A dragged element leaves the drop target
ondragover — The dragged element is on top of the drop target
ondragstart — User starts to drag an element
ondrop — Dragged element is dropped on the drop target

Clipboard Events

oncopy — User copies the content of an element
oncut — The user cuts an element's content
onpaste — A user pastes the content in an element

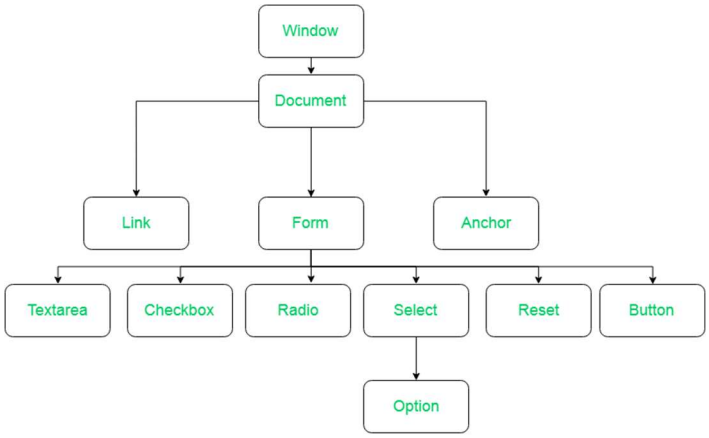
Media Events

onabort — Media loading is aborted
oncanplay — The browser can start playing media (e.g. a file has buffered enough)
oncanplaythrough — The browser can play through media without stopping
ondurationchange — The duration of the media changes
onended — The media has reached its end
onerror — Happens when an error occurs while loading an external file
onloadeddata — Media data is loaded
onloadedmetadata — Metadata (like dimensions and duration) are loaded
onloadstart — The browser starts looking for specified media
onpause — Media is paused either by the user or automatically
onplay — The media has been started or is no longer paused
onplaying — Media is playing after having been paused or stopped for buffering
onprogress — The browser is in the process of downloading the media
onratechange — The playing speed of the media changes
onseeked — User is finished moving/skipping to a new position in the media
onseeking — The user starts moving/skipping
onstalled — The browser is trying to load the media but it is not available
onsuspend — The browser is intentionally not loading media
ontimeupdate — The playing position has changed (e.g. because of fast forward)
onvolumechange — Media volume has changed (including mute)
onwaiting — Media paused but expected to resume (for example, buffering)

Document Object Model (DOM)

Introduction:

- The Document Object Model (DOM) is a programming interface for HTML and XML(Extensible markup language) documents. It defines the logical structure of documents and the way a document is accessed and manipulated.
- **Note**: It is called a Logical structure because DOM doesn't specify any relationship between objects.
- DOM is a way to represent the webpage in a structured hierarchical way so that it will become easier for programmers and users to glide through the document. With DOM, we can easily access and manipulate tags, IDs, classes, Attributes, or Elements using commands or methods provided by the Document object.
- **Structure of DOM:**
- DOM can be thought of as a Tree or Forest(more than one tree). The term structure model is sometimes used to describe the tree-like representation of a document. One important property of DOM structure models is structural isomorphism: if any two DOM implementations are used to create a representation of the same document, they will create the same structure model, with precisely the same objects and relationships.
- **Why called an Object Model?**
- Documents are modeled using objects, and the model includes not only the structure of a document but also the behavior of a document and the objects of which it is composed of like tag elements with attributes in HTML.
- Properties of DOM:
- Let's see the properties of the document object that can be accessed and modified by the document object.



- **Window Object**: Window Object is always at top of the hierarchy.
- **Document object**: When an HTML document is loaded into a window, it becomes a document object.
- **Form Object**: It is represented by form tags.
- **Link Objects**: It is represented by link tags.
- **Anchor Objects**: It is represented by a href tags.
- **Form Control Elements**: Form can have many control elements such as text fields, buttons, radio buttons, and checkboxes, etc.

Methods of Document Object:

- **document.write("string"):** writes the given string on the document.
- **getElementById():** returns the element having the given id value.

- **getElementsByName():** returns all the elements having the given name value.
- **getElementsByTagName():** returns all the elements having the given tag name.
- **getElementsByClassName():** returns all the elements having the given class name.

What DOM is not?

- The Document Object Model is not a binary description where it does not define any binary source code in its interfaces.
- The Document Object Model is not used to describe objects in XML or HTML whereas the DOM describes XML and HTML documents as objects.
- The Document Object Model is not represented by a set of data structures; it is an interface that specifies object representation.
- The Document Object Model does not show the criticality of objects in documents i.e it doesn't have information about which object in the document is appropriate to the context and which is not.

HISTORY OBJECT

- The history property has the return value as history object, which is an array of history items having details of the URL's visited from within that window. Also, note that the History object is a JavaScript object and not an HTML DOM object.
- General syntax of history property of window object:
- *window.history*
- The JavaScript runtime engine automatically creates this object.
- An introduction on the history object and the properties and methods associated with it was covered in an earlier section.

Property of History Object:

1. length:

- The length property of the history object returns the number of elements in the history list.
- General syntax of length property of history Object:
- *history.length*
- An example to understand the length property of history object.

2. current:

- This property contains the complete URL of the current History entry. General syntax of current property of history Object:
- *history.current*

3. next:

- The next property of history object contains the complete URL of the next element in the History list. This functionality or the URL visited is the same as pressing the forward button or menu.
- General syntax of next property of history Object.
- *history.next*

4. previous:

- The previous property of history object contains the complete URL of the previous element in the History list. This functionality or the URL visited is the same as pressing the back button or menu.
- General syntax of previous property of history Object.
- *history.previous*

Methods of History Object:

1. back():

- There may be scenarios where the programmer wishes to load the previous URL present in the history list. In this case, the programmer can make use of the back() method of the history object.
- The back() method of the history object takes the user to the previous page. The functionality results in the same as pressing the back button of the browser.
- General syntax of back method of history Object:
- history.back()

2. forward():

- The forward() method of the history object loads the next URL in the History list. The functionality results are the same as pressing the forward button of the browser.
- General syntax of forward method of history Object:
- *history.forward()*

3. go():

- If the programmer wishes to load a specified URL from the History list, then the go method of history object can be used.
- General syntax of go method of history Object:
- *history.go(number) or history.go(URL)*
- The back method seen above is the same as history.go(-1) in terms of go method of history object. The forward method seen above is the same as history.go(1)

EMAIL VALIDATION & FORM VALIDATION

EMAIL VALIDATION

- Validating email is a very important point while validating an HTML form. In this page we have discussed how to validate an email using JavaScript :
- An email is a string (a subset of ASCII characters) separated into two parts by @ symbol. a "personal_info" and a domain, that is personal_info@domain. The length of the personal_info part may be up to 64 characters long and domain name may be up to 253 characters.
- The personal_info part contains the following ASCII characters.
- Uppercase (A-Z) and lowercase (a-z) English letters.
- Digits (0-9).
- Characters ! # $ % & ' * + - / = ? ^ _ ` { | } ~
- Character . (period, dot or fullstop) provided that it is not the first or last character and it will not come one after the other.

- JavaScript code to validate an email id
- function ValidateEmail(mail)

```
{
if (/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/.test(myForm.emailAddr.value))
{
        return (true);
}
alert("You have entered an invalid email address!");
        return (false);
}
```

FORM VALIDATION

- It is important to validate the form submitted by the user because it can have inappropriate values. So validation is must.
- The JavaScript provides you the facility the validate the form on the client side so processing will be fast than server-side validation. So, most of the web developers prefer JavaScript form validation.
- Through JavaScript, we can validate name, password, email, date, mobile number etc fields.
- In this example, we are going to validate the name and password. The name can't be empty and password can't be less than 6 characters long.
- Here, we are validating the form on form submit. The user will not be forwarded to the next page until given values are correct.

```
<script>
function validateform()
{
	var name=document.myform.name.value;
	var password=document.myform.password.value;
	if (name==null || name=="")
	{
		alert("Name can't be blank"); return false;
	}
	else if(password.length<6)
	{
		alert("Password must be at least 6 characters long."); return false;
	}
}
</script>
```

JavaScript Number Validation

```
<script>
function validate()
{
	var num=document.myform.num.value;
	if (isNaN(num))
	{
	  document.getElementById("numloc").innerHTML="Enter Numeric value only";
	  return false;
	}
	else
	{
	  return true;
	}
}
</script>
```

JavaScript Retype Password Validation

```
<script type="text/javascript">
function matchpass()
{
	var firstpassword=document.f1.password.value;
	var secondpassword=document.f1.password2.value;
	if(firstpassword==secondpassword){
		return true;
```

```
        }
        else{
                alert("password must be same!");
                return false;
        }
}  </script>
```