**MODULE – III**

**Client-Side Programming: JavaScript Language - History and versions of JavaScript - Introduction to JavaScript - JavaScript in Perspective - Basic Syntax – Variables and Data Types - Statements. - Operators - Literals - Functions - Objects - Arrays - Built-in Objects - Host Objects: Browsers and the DOM - Introduction to the Document Object Model- Intrinsic Event Handling - DOM History and Levels**

**Introduction to JavaScript**

I'd like to begin with a "Hello World!" JavaScript program. There's just one problem: JavaScript itself has no statement for performing output. Instead, the JavaScript language specification leaves it up to browsers to supply output (and input) methods. Happily, certain methods are supported by almost all modern browsers, even though technically they are not part of any standard.

```
window.alert("Hello World!");
```

We can execute this program by referencing a file containing it from a script element within an HTML document.

**Variables and Data Types**

As previously mentioned, variables do not have data types in JavaScript. However, every variable has a value, and every value belongs to one of six JavaScript data types. Every numeric value is of type Number, string values are of type String, the literals true and false represent the two Boolean values, the literal null represents the one value of type Null, and every object is of type Object.

**Data Types**

JavaScript variables can hold different data types: numbers, strings, objects and more:

```
let x;                                  // Now x is undefined
let length = 16;                        // Number
let lastName = "Johnson";               // String
let x = {firstName:"John", lastName:"Doe"};   // Object
```

The Concept of Data Types

In programming, data types are an important concept. To be able to operate on variables, it is important to know something about the type.

**Statements**

There are different kinds of statement is present in java script.

Example

Assignment Statement

Let a=20

Output statement

Document.write("Welcome")

Alert("Hai")

Looping Statements

For loop, while loop

**OPERATORS**

JavaScript Arithmetic Operators

**Arithmetic operators** are used to perform arithmetic on numbers:

| Operator | Description |
|---|---|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| ** | Exponentiation (ES2016) |
| / | Division |
| % | Modulus (Division Remainder) |
| ++ | Increment |
| -- | Decrement |

**Assignment operator =**

```
let x = 10;
x += 5;
```

**Comparison Operators**

| Operator | Description |
|---|---|
| == | equal to |
| === | equal value and equal type |
| != | not equal |
| !== | not equal value or not equal type |
| > | greater than |
| < | less than |
| >= | greater than or equal to |
| <= | less than or equal to |
| ? | ternary operator |

**JavaScript Logical Operators**

| Operator | Description |
|---|---|
| && | logical and |
| \|\| | logical or |
| ! | logical not |

**JavaScript literal or constant**

JavaScript Literals are constant values that can be assigned to the variables that are called literals or constants. JavaScript Literals are syntactic representations for different types of data like numeric, string, Boolean, array, etc data. ... The literal “john” represents, the value john for the variable name.

**Functions**

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:
(parameter1, parameter2, ...)

The code to be executed, by the function, is placed inside curly brackets: {}

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

- Function parameters are listed inside the parentheses () in the function definition.
- Function arguments are the values received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.

Example: Calculate the product of two numbers, and return the result:

```
let x = mul(4, 3);   // Function is called, return value will end up in x
function mul(a, b) {
  return a * b;           // Function returns the product of a and b
}
```

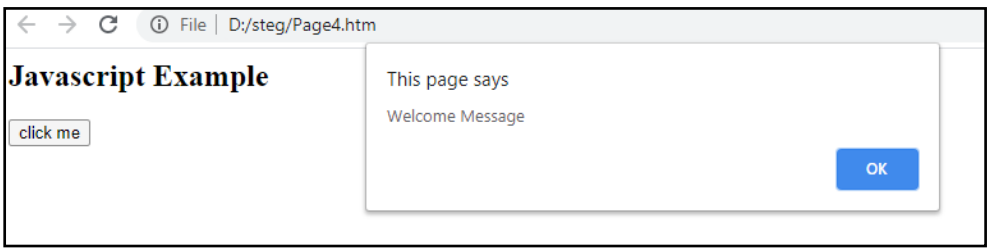
The result in x will be:
12

Importance of function
we can reuse code: Define the code once, and use it many times.

we can use the same code many times with different arguments, to produce different results.

**Example 2**

```
<html>
<script>
function show()
{
  alert("Welcome Message");
}
</script>
<body>
<h2>Javascript Example </h2>
<input type=button value="click me" onclick="show()">
</body>
</html>
```



```
<!-- Page5.htm  Adding two numbers using Javascript-->
<html>
<script>
function add()
{
  var x,y,z;
  x=document.getElementById("t1");
  y=document.getElementById("t2");
  z=parseInt(x.value)+parseInt(y.value);
  document.getElementById("t3").value=z;
}
</script>
<body>
<h2>Javascript Example </h2>
A<input type=text id=t1><br>
B<input type=text id=t2><br>
<input type=button value="ADD" onclick="add()"><br>
C<input type=text id=t3>
</body>
</html>
```

**Javascript Example**

A 45
B 78
ADD
C 123

```
<!-- Page6.htm  Multiplication Table-->
<html>
<script>
function multable()
{
  var n,i,z;
  n=parseInt(document.getElementById("t1").value);
  for(i=1;i<=20;i++)
  {
    z=i*n;
    document.write(i+" x "+n+" = "+z+"<br>");

  }
}
</script>
<body>
<h2>Multiplicaton Table</h2>
A<input type=text id=t1><br>
<input type=button value="OK" onclick="multable()"><br>
</body>
</html>
```

# Multiplicaton Table

A 7

OK

1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
5 x 7 = 35
6 x 7 = 42
7 x 7 = 49
8 x 7 = 56
9 x 7 = 63
10 x 7 = 70
11 x 7 = 77
12 x 7 = 84
13 x 7 = 91
14 x 7 = 98
15 x 7 = 105
16 x 7 = 112
17 x 7 = 119
18 x 7 = 126
19 x 7 = 133
20 x 7 = 140

## OBJECTS

In JavaScript, almost "everything" is an object.

- Booleans can be objects
- Numbers can be objects
- Strings can be objects
- Dates are always objects
- Maths are always objects
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects
- Objects are always objects

**Arrays**

An array is a special variable, which can hold more than one value. The following example shows single dimensional array and built in function present in an array.

**Example**

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");                    // Add new Element at end
fruits.pop();                           // Remove the last element
fruits.shift();                         // Remove the first element
fruits.unshift("Lemon");                // Add new Element at beginning
```

**fruits.sort();** **// Sort the array elements**

---

**Combine two arrays : In the following example variable Children holds the name of girls and boys.**

```
var Girls     = ["Kalai", "Selvi"];
var Boys      = ["Manikandan", "Senthil", "Sivaram"];
var Children  = Girls.concat(Boys);          // Concatenates (joins) Girls and Boys
```

---

**Built in Objects and Functions**

The following example shows date functions present in java script

```
<html>
<body>
<h2>JavaScript Display Date</h2>
<p id="demo"></p>
<script>
var d = new Date();

document.getElementById("demo").innerHTML = d.getDate()+ "/"+ (d.getMonth()+1) + "/" + d.getFullYear();

</script>

</body>
</html>
```

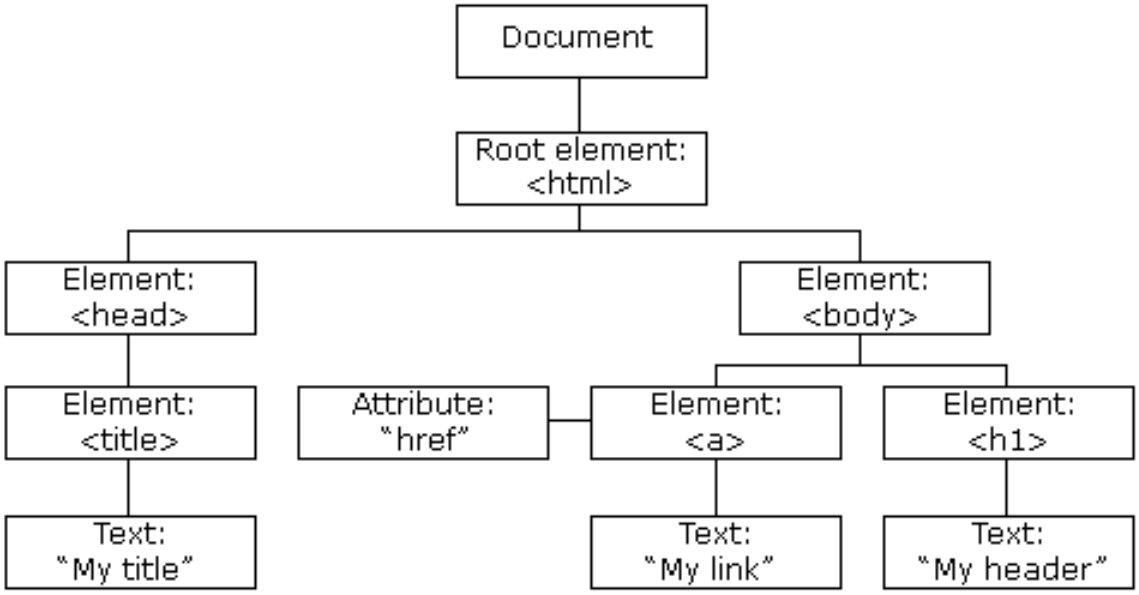
**Output**

# The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:

### The HTML DOM Tree of Objects




# Finding HTML Elements

When you want to access HTML elements with JavaScript, you have to find the elements first.

There are a couple of ways to do this:

- Finding HTML elements by id
- Finding HTML elements by tag name
- Finding HTML elements by class name
- Finding HTML elements by CSS selectors
- Finding HTML elements by HTML object collections

```
var myElement = document.getElementById("intro");
var x = document.getElementsByTagName("p");
var y = document.getElementsByName("sam");
```

## Event Handling

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:
An HTML web page has finished loading
An HTML input field was changed
An HTML button was clicked
Often, when events happen, you may want to do something.
JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.

### Example of onclick event

```
<html>
<body>
<p>Please input a number between 1 and 10:</p>
<input id="numb">
<button type="button" onclick="myFunction()">Submit</button>
<p id="demo"></p>
<script>
function myFunction() {
  var x, msg;
  x = document.getElementById("numb").value;
  if (isNaN(x) || x < 1 || x > 10) {
    msg= "Input not valid";
  } else {
    msg = "Input OK";
  }
  document.getElementById("demo").innerHTML = msg;
}
</script>
</body>
</html>
```

# DOM History and Levels

## History

The DOM Window object provides access to the browser's session history through the history object. It exposes useful methods and properties that let you navigate back and forth through the user's history, and manipulate the contents of the history stack.

The History object is automatically created by the JavaScript runtime engine and consists of an array of URLs. These URLs are the URLs the user has visited within a browser window. The History object is part of the Window object and is accessed through the window.history property.

**Levels of DOM:**

Level 0: Provides a low-level set of interfaces.

Level 1: DOM level 1 can be described in two parts: CORE and HTML.

CORE provides low-level interfaces that can be used to represent any structured document.

HTML provides high-level interfaces that can be used to represent HTML documents.

Level 2 : consists of six specifications: CORE2, VIEWS, EVENTS, STYLE, TRAVERSAL, and RANGE.

CORE2: extends the functionality of CORE specified by DOM level 1.

VIEWS: views allows programs to dynamically access and manipulate the content of the document.

EVENTS: Events are scripts that are either executed by the browser when the user reacts to the web page.

STYLE: allows programs to dynamically access and manipulate the content of style sheets.

TRAVERSAL: allows programs to dynamically traverse the document.

RANGE: allows programs to dynamically identify a range of content in the document.

Level 3: consists of five different specifications: CORE3, LOAD and SAVE, VALIDATION, EVENTS, and XPATH.

CORE3: extends the functionality of CORE specified by DOM level 2.

LOAD and SAVE: allows the program to dynamically load the content of the XML document into the DOM document and save the DOM Document into an XML document by serialization.

VALIDATION: allows the program to dynamically update the content and structure of the document while ensuring the document remains valid.

EVENTS: extends the functionality of Events specified by DOM Level 2.

XPATH: XPATH is a path language that can be used to access the DOM tree.