PHP MODULE II

JAVASCRIPT: INTRODUCTION

JavaScript is a programming language that can be included on web pages to make them more interactive. We can use it to check or modify the contents of forms, change images, open new windows and write dynamic page content. We can even use it with CSS to make DHTML (Dynamic HyperText Markup Language). This allows we to make parts of our web pages appear or disappear or move around on the page. JavaScripts only execute on the page(s) that are on our browser window at any set time. When the user stops viewing that page, any scripts that were running on it are immediately stopped. The only exceptions are cookies or various client side storage APIs, which can be used by many pages to store and pass information between them, even after the pages have been closed.

Before we go any further, let me say; JavaScript has nothing to do with Java. If we are honest, JavaScript, originally nicknamed LiveWire and then LiveScript when it was created by Netscape, should in fact be called ECMAscript as it was renamed when Netscape passed it to the ECMA for standardization.

JavaScript is a client side, interpreted, object oriented, high level scripting language, while Java is a client side, compiled, object oriented high level language. Now after that mouthful, here's what it means.

- Client-side programming runs on the user's computer
- Programs are passed to the computer that the browser is on, and that computer runs the script
- The problem:- the limit of control and problems with operating systems and web browsers.
- JavaScript code is typically embedded in the HTML
- Used to make web pages more interactive
- It executes on our web browser window
- Interpreted and run by the client‘s browser
- The file stored using the extension .js
- JavaScript code is case sensitive
- White space between words and tabs are ignored
- Line breaks are ignored except within a statement
- JavaScript statements end with a semi- colon ;

CLIENT SIDE PROGRAMMING

Programs are passed to the computer that the browser is on, and that computer runs them. The alternative is server side, where the program is run on the server and only the results are passed to the computer that the browser is on. Examples of this would be PHP, Perl, ASP, JSP etc

Client-side programming runs on the user's computer. The programs are passed to the computer that the browser is on, and that computer runs the script. But the problem is the limit of control and problems with operating systems and web browsers.

SCRIPT TAG

The <SCRIPT> tag alerts a browser that JavaScript code follows. It is typically embedded in the HTML. The <script> tag is used to define a client-side script, such as a JavaScript. The <script> element either contains scripting statements, or it points to an external script file through the src attribute. Common uses for JavaScript are image manipulation, form validation, and dynamic changes of content. The script tag has two purposes:

1. It identifies a block of script in the page.
2. It loads a script file.

Which it does depends on the presence of the src attribute. A </script> close tag is required in either case.
A script tag can contain these attributes:

- src="url"

The src attribute is optional. If it is present, then its value is a url which identifies a .js file. The loading and processing of the page pauses while the browser fetches, compiles, and executes the file. The content between the <script src="url"> and the </script>should be blank.

- language="javascript"

This attribute has been deprecated. It was used to select other programming languages and specific versions of JavaScript. You don't need it. Don't use it.

- type="text/javascript"

This attribute is optional. Since Netscape 2, the default programming language in all browsers has been JavaScript. In XHTML, this attribute is required and unnecessary. In HTML, it is better to leave it out. The browser knows what to do.

```
<SCRIPT language = "JavaScript">
    statements
</SCRIPT>
```

Example

```
<html><head>
<SCRIPT type="text/javascript">
alert("Welcome to the script tag test page.");
</SCRIPT>
</head> <body>
<h2>Good Morning</h2>
</body> </html>
```

- Save the changes by choosing Save from the File menu.
- Then Refresh the browser by clicking the Refresh or Reload button.

JAVASCRIPT COMMENTS

Comments can be added to explain the JavaScript, or to make the code more readable. Single line comments start with //. The following example uses single line comments to explain the code:

Example

```
<script type="text/javascript">
// Write a heading
document.write("<h1>This is a heading</h1>");
// Write two paragraphs:
document.write("<p>This is a paragraph.</p>");
document.write("<p>This is another paragraph.</p>");
</script>
```

JavaScript Multi-Line Comments

Multi line comments start with /* and end with */. The following example uses a multi line comment to explain the code:

Example

```
<script type="text/javascript">
/*
The code below will write
one heading and two paragraphs
*/
document.write("<h1>This is a heading</h1>");
document.write("<p>This is a paragraph.</p>");
document.write("<p>This is another paragraph.</p>");
</script>
```

Using Comments at the End of a Line

In the following example the comment is placed at the end of a code line:

Example

```
<script type="text/javascript">
document.write("Hello"); // Write "Hello"
document.write(" Dolly!"); // Write " Dolly!"
</script>
```

VARIABLES

Variables are used to store data. A variable's value can change during the execution of a script. We can refer to a variable by its name to display or change its value. The rules for JavaScript variable names:

- Variable names are case sensitive (y and Y are two different variables)
- Variable names must begin with a letter, the $ character, or the underscore character
- Because JavaScript is case-sensitive, variable names are case-sensitive.

Declaring JavaScript Variables

Creating variables in JavaScript is most often referred to as "declaring" variables. We declare JavaScript variables with the var keyword:

var x;

var carname;

After the declaration shown above, the variables are empty (they have no values yet). However, we can also assign values to the variables when we declare them:

var x=5;

var carname="Volvo";

After the execution of the statements above, the variable x will hold the value 5, and carname will hold the value Volvo. When we assign a text value to a variable, use quotes around the value. If we re-declare a JavaScript variable, it will not lose its value.

Local JavaScript Variables

A variable declared within a JavaScript function becomes LOCAL and can only be accessed within that function (the variable has local scope). We can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared. Local variables are destroyed when we exit the function.

Global JavaScript Variables

Variables declared outside a function become GLOBAL, and all scripts and functions on the web page can access it. Global variables are destroyed when we close the page. If we declare a variable, without using "var", the variable always becomes GLOBAL.

Assigning Values to Undeclared JavaScript Variables

If we assign values to variables that have not yet been declared, the variables will automatically be declared as global variables. These statements:

x=5;

carname="Volvo";

It will declare the variables x and carname as global variables (if they don't already exist).

INCLUDING JAVASCRIPT IN HTML

We have to include JavaScript code in HTML using three deferent ways

1. **Within <head> tag**

```
<html>  <head>
        <script type="text/javascript">
                alert("Good Morning");
        </script>
</head>   <body>
        <H2> Hello JavaScript </H2>
        <p>Welcome to JavaScript</p>
</body>   </html>
```

2. Within <body> tag

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Embedding JavaScript</title>
</head> <body>
   <div id="greet"></div>
   <script>
      document.getElementById("greet").innerHTML = "Hello World!";
   </script>
</body> </html>
```

3. Using external .js file

We can use the SRC attribute of the <SCRIPT> tag to call JavaScript code from an external text file. This is useful if we have a lot of code or we want to run it from several pages, because any number of pages can call the same external JavaScript file. The text file itself contains no HTML tags.

Example:

```
<html>  <head>
        <script type="text/javascript" src="mysript.js">

        </script>
</head> <body>
        <H2> Hello JavaScript </H2>
        <p>Welcome to GEMS College</p>
</body>  </html>
```

The myscript.js file containing the following code only:

```
alert("Good Afternoon");
alert("Helloo");
```

DATA TYPES

JavaScript is a dynamic type language, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. You need to use var here to specify the data type. It can hold any type of values such as numbers, strings etc. For example:

```
var a = 40;          //holding number
var b = "Rahul";     //holding string
```

JavaScript provides different data types to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type
2. Non-primitive (Complex) data type

Primitive data type

- **String**:- Strings are used for storing text. Strings must be inside of either double or single quotes.

```
var str1 = "It is alright, ";
var str2 = `He is Johnny`;
var str3 = 'Good Morning "Tomy"';
```

- **Number**:- There is only one type of Number is used to represent positive or negative numbers with or without a decimal point.

```
var x = 125;
var y = x + 3.7;
var z= x + y;
```

- **Boolean**:- A boolean represents only one of two values: true, or false.

```
var bval = false;
var k = if(10>5)   // k will store true
```

- **Null**:- Null has one value: null. It is explicitly nothing. A null value means that there is no value. It is not equivalent to an empty string ("") or 0, it is simply nothing.

```
var a = null;
```

- **Undefined**:- The meaning of undefined is "value is not assigned". If a variable is declared, but not assigned, then its value is undefined:

```
var car;
```

Non-primitive (Complex) data type

- **Object**:- The object is a complex data type that allows you to store collections of data. An object contains properties, defined as a key-value pair. A property key (name) is always a string, but the value can be any data type, like strings, numbers, booleans, or complex data types like arrays, function and other objects.

```
<script type = "text/javascript">
var person = {
  firstName : "Adnan",
  lastName  : "Sami",
  age       : 50,
  eyeColor  : "blue"
};
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>
```

- **Array**:- An array is a type of object used for storing multiple values in single variable. Each value (also called an element) in an array has a numeric position, known as its index, and it may contain data of any data type-numbers, strings, booleans, functions, objects, and even other arrays. The array index starts from 0, so that the first array element is arr[0] not arr[1]. The simplest way to create an array is by specifying the array elements as a comma-separated list enclosed by square brackets, as shown in the example below:

```
var colors = ["Red", "Yellow", "Green", "Orange"];
var cities = ["London", "Paris", "New York"];
alert(colors[0]); // Output: Red
alert(cities[2]); // Output: New York
```

- **Function:-** JavaScript doesn't have a function data type but when we find the data type of a function using the typeof operator, we find that it returns a function. This is because a function is an object in JavaScript. Ideally the data type of a function should return an object but instead, it returns a function.

```
var greeting = function(){
        return ("Hello World!");
}
typeof(greetings); // This will return data type function
```

OPERATORS:

Arithmetic Operators

Arithmetic operators are used to perform arithmetic between variables and/or values. Given that y=5, the table below explains the arithmetic operators:

Operator	Description	Example	Result	
+	Addition	x=y+2	x=7	y=5
-	Subtraction	x=y-2	x=3	y=5
*	Multiplication	x=y*2	x=10	y=5
/	Division	x=y/2	x=2.5	y=5
%	Modulus (remainder)	x=y%2	x=1	y=5
++	Increment	x=++y	x=6	y=6
		x=y++	x=5	y=6
--	Decrement	x=--y	x=4	y=4
		x=y--	x=5	y=4

Assignment Operators

Assignment operators are used to assign values to JavaScript variables.
Given that x=10 and y=5, the table below explains the assignment operators:

Operator	Example	Same As	Result
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

The String (+) Operator

The + operator can also be used to add string variables or text values together.
To add two or more string variables together, use the + operator.

```
txt1="What a very";
txt2="nice day";
txt3=txt1+txt2;
```

After the execution of the statements above, the variable txt3 contains "What a very nice day". To add a space between the two strings, insert a space into one of the strings:

```
txt1="What a very ";
txt2="nice day";
txt3=txt1+txt2;
or insert a space into the expression:
txt1="What a very";
txt2="nice day";
txt3=txt1+" "+txt2;
```

After the execution of the statements above, the variable txt3 contains:
"What a very nice day"

Relational (Comparison) Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values. Given that x=5, the table below explains the comparison operators:

Operator	Description	Example
==	is equal to	x==8 is false x==5 is true
===	is exactly equal to (value and type)	x===5 is true x==="5" is false
!=	is not equal	x!=8 is true
>	is greater than	x>8 is false
<	is less than	x<8 is true

>=	is greater than or equal to	x>=8 is false
<=	is less than or equal to	x<=8 is true

Comparison operators can be used in conditional statements to compare values and take action depending on the result:
if (age<18) document.write("Too young");

Logical Operators
Logical operators are used to determine the logic between variables or values. Given that x=6 and y=3, the table below explains the logical operators:

Operator	Description	Example
&&	And	(x < 10 && y > 1) is true
||	Or	(x==5 || y==5) is false
!	Not	!(x==y) is true

CONDITIONAL STATEMENTS
In JavaScript we have the following conditional statements:

- **if statement** - use this statement to execute some code only if a specified condition is true
- **if...else statement** - use this statement to execute some code if the condition is true and another code if the condition is false
- **if...else if....else statement** - use this statement to select one of many blocks of code to be executed
- **switch statement** - use this statement to select one of many blocks of code to be executed

If Statement
Use the if statement to execute some code only if a specified condition is true.
Syntax

```
if (condition)
  {
  code to be executed if condition is true
  }
```

Note that if is written in lowercase letters. Using uppercase letters (IF) will generate a JavaScript error!
Example

```
<script>
        var myAge = 20;
        var yourAge = 21;

        if(myAge < yourAge)
        {
                document.write("My Age is LESS than your Age");
```

```
        }
    </script>
```

Notice that there is no ..Else.. in this syntax. We tell the browser to execute some code only if the specified condition is true.

If...else Statement

Use the if....else statement to execute some code if a condition is true and another code if the condition is not true.

Syntax

```
if (condition)
  {
  code to be executed if condition is true
  }
else
  {
  code to be executed if condition is not true
  }
```

Example

```
<script>
        var myAge = 22;
        var yourAge = 20;

        if(myAge < yourAge)
        {
            document.write("My Age is Less than your Age");
        }
        else
        {
        document.write("My Age is Greater than your Age");
        }
  </script>
```

If...else if...else Statement

Use the if....else if...else statement to select one of several blocks of code to be executed.

Syntax

```
if (condition1)
  {
  code to be executed if condition1 is true
  }
else if (condition2)
  {
  code to be executed if condition2 is true
  }
```

```
else
  {
  code to be executed if neither condition1 nor condition2 is true
  }
```

Example

```
<script type="text/javascript">
var a = 10;
var b = 5;
if (a>b)
  {
  document.write("<b>a is larger than b</b>");
  }
else if (a<b)
  {
  document.write("<b>b is larger than a</b>");
  }
else
  {
  document.write("<b>a nd b are equal</b>");
  }
</script>
```

Switch Statement

Use the switch statement to select one of many blocks of code to be executed.

Syntax

```
switch(n)
{
case 1:
  execute code block 1
  break;
case 2:
  execute code block 2
  break;
default:
  code to be executed if n is different from case 1 and 2
}
```

This is how it works: First we have a single expression n (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use break to prevent the code from running into the next case automatically.

Example

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

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

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

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

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

          default:
            document.write("Unknown Grade<br />")
        }
    </script>
```


LOOPS

Loops execute a block of code a specified number of times, or while a specified condition is true. Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this. In JavaScript, there are two different kind of loops:

- for - loop through a block of code a specified number of times
- while - loop through a block of code while a specified condition is true
- do……while loop through a block of statement will execute at lease once

The for Loop

The for loop is used when you know in advance how many times the script should run. For loop repeats until a specified condition evaluates to false

Syntax

```
for (initialisation; condition_checkd; incre/decre)
{
code to be executed
}
```

For statement 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 we can increase or decrease our counter.
- We can put all the three parts in a single line separated by semicolons.

Example

```
<script>
for (i = 15; i <= 65; i++)
{
        if(i%2==0)
        {
                document.write("     "+i);
        }
}
</script>
```

The while loop

The while loop loops through a block of code while a specified condition is true. The condition is evaluated before executing the statement

Syntax

```
while (condition)
  {
  code to be executed
  }
```

Example

```
<script>
var i=15;
while (i <= 65)
{
        if(i%2==0)
        {
                document.write("      "+i);
        }
        i=i+1;
}
</script>
```

The do...while Loop

The do...while loop is a variant of the while loop. This loop will execute the block of code ONCE, and then it will repeat the loop as long as the specified condition is true. Don't miss the semicolon after the while statement

Syntax

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

Example

```
<script>
var i=15;

do{
        if(i%2==0)
        {
                document.write("     "+i);
        }
        i=i+1;
}
while (i <= 65) ;
```

The break Statement

The break statement will break the loop and continue executing the code that follows after the loop

Example

```
<script>
var i;
for (i = 0; i <= 15; i++) {
    if (i === 7)
                break;

        document.write("<br>The number is: "+i);
}
</script>
```

The continue Statement

The continue statement terminates execution of the statements in the current iteration of the current loop, and continues execution of the loop with the next iteration.

Example

```
<script>
var i;
for (i = 0; i <= 15; i++) {
    if (i === 7)
```

```
            continue;
    if (i === 11)
            continue;
        document.write("<br>The number is: "+i);
}
</script>
```

OUTPUT FUNCTIONS

Two output functions are used in JavaScript

- **Write**() method outputs one or more values to the screen without a newline character.
- **Writeln**() method outputs one or more values to the screen with a newline character.

Example

```
<html><body>
<p>Note that write() does NOT add a new line after each statement:</p>
<script type="text/javascript">
document.write("Hello World!");
document.write("Have a nice day!");
</script>
<p>Note that writeln() add a new line after each statement:</p>
<script type="text/javascript">
document.writeln("Hello World!");
document.writeln("Have a nice day!");
</script>
</body> </html>
```

POPUP BOXES

In JavaScript three types of popups used

1. alert() Box
2. confirm() Box
3. prompt() Box

Alert Box

An alert box is often used if we 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:

```
alert("sometext");
```

Example:

```
<html><head>
</head><body>
        <h2>Alert Box Displayed</h2>
        <script type="text/javascript">
```

```
            alert("Good Morning");
        </script>
    </body></html>
```

Confirm Box

A confirm box is often used if we 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

```
confirm("sometext");
```

Example

```
<html><head>
</head>
<body>
    <h2>Confirm Box Displayed</h2>
    <script type="text/javascript">
        confirm("Do You Wnats to Open This Document?");
    </script>
</body>
</html>
```

Prompt Box

A prompt box is often used if we 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

```
prompt("sometext","defaultvalue");
```

Example

```
<html><head>
</head>
<body>
    <h2>Prompt Box Displayed</h2>
    <script type="text/javascript">
        var name;
        name=prompt("Please enter your name","Harry Potter");
        document.write(name);
    </script>
</body>
</html>
```

FUNCTIONS

There are two variant of functions

1. Built-in functions
2. User defined functions

Built-in Global functions

- **Alert(), Confirm(), Prompt()** are already discussed

- **isNan():-** It determines whether a value is NaN or not. The NaN means Not-a-Number. It checks the value is an illegal number. This function will returns Boolean value. That is, it returns true if the value equates to NaN. Otherwise it returns false.

Examle

isNan(123); //returns false
isNan("hello"); //returns true

- **Number():-** It is used to convert a specified object into a number. It converts variable to number

Example

Number(true); //return 1
Number(123); //return 123
Number("hello"); //return NaN

- **parseInt():-** The parseInt() function parses a string argument and returns an integer of the specified radix (the base in mathematical numeral systems). The radix parameter is used to specify which numeral system to be used. If the string cannot be converted to an integer value, NaN is returned

Syntax

parseInt(value, radix);

User Defined Function

A function will be executed by an event or by a call to the function. To keep the browser from executing a script when the page loads, you can put your script into a function. A function contains code that will be executed by an event or by a call to the function. We may call a function from anywhere within a page (or even from other pages if the function is embedded in an external .js file). Functions can be defined both in the <head> and in the <body> section of a document. However, to assure that a function is read/loaded by the browser before it is called, it could be wise to put functions in the <head> section.

Every function should begin with the keyword *function* followed by function name. The function name should be unique. A list of parameters

enclosed within parenthesis and separated by comma. A list of statement composing the body of the function enclosed within curly braces { }. Function can be called (invoked) by typing its name followed by a set of parenthesis. Function can return a value (result) back to the script using return statement.

Syntax:-

function function_name(var1,var2,...,varX)
{
some codes;
}

The parameters var1, var2, etc. are variables or values passed into the function. The { and the } defines the start and end of the function. A function with no parameters must include the parentheses () after the function name. Do not forget about the importance of capitals in JavaScript! The word function must be written in lowercase letters, otherwise a JavaScript error occurs! Also note that we must call a function with the exact same capitals as in the function name.

Example

```
<html>
<head>
	<script type="text/javascript">
	function fact()
	{
		var n,f;
		f=1;
		n=prompt("Enter The Value");
		for(i=1;i<=n;i++)
		{
			f = f * i;
		}
		document.write("the factorial of "+n+" is: "+f);
	}
	</script>
</head>
<body>
	<script type="text/javascript">
		fact();
	</script>
</body>
</html>
```

CALLING FUNCTIONS WITH TIMER

- To execute a function after a certain period of time, we use **setTimeout()**. Its basic syntax is

  setTimeout(function, milliseconds)

This function accepts two parameters: *A function*, which is the function to execute. An optional *delay* parameter, in milliseconds

- To execute a function repeatedly, starting after the interval of time, then repeating continuously at that interval, we can use **setInterval()**. Its basic syntax is

  setInterval(function, milliseconds);

Example 1

```
<html>
<head>
    <script type="text/javascript">
    function fact()
    {
        var n,f;
        f=1;
        n=prompt("Enter The Value");
        for(i=1;i<=n;i++)
        {
            f = f * i;
        }
        document.write("the factorial of "+n+" is: "+f);
    }
    </script>
</head>
<body>
    <script type="text/javascript">
    setTimeout(fact,5000);
    </script>
</body>
</html>
```

Example 2

```
<html>
<head>
    <script type="text/javascript">
    function display()
    {
        alert("Good Morning");
    }
```

```
        </script>
    </head>
    <body>
        <script type="text/javascript">
        setInterval(display,3000);
        </script>
    </body>
    </html>
```

JAVASCRIPT EVENTS

By using JavaScript, we have the ability to create dynamic web pages. Events are actions that can be detected by JavaScript. Every element on a web page has certain events which can trigger a JavaScript.

- **onClick**

We can use the onClick event of a button element to indicate that a function will run when a user clicks on the button. We define the events in the HTML tags. Execute a JavaScript when the user clicks on an element. Events are normally used in combination with functions, and the function will not be executed before the event occurs.

Example

```
<html>  <head>
    <script>
        function display() {
          alert("GEMS COLLEGE");
        }
    </script>  </head>
  <body>
    <p>Click the following button and see result</p>
    <form >
      <input type="button" onclick="display()" value="Click Me" />
            <h1 onclick="this.innerHTML='Ooops!'">Click on this text!</h1>
    </form>  </body><html>
```

- **onLoad**

The onload event occurs immediately after a page is loaded. The onLoad event is triggered when the user enters the web page. The onLoad event is often used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information. The onLoad event is also often used to deal with cookies that should be set when a user enters a web page. For example, we could have a popup asking for the user's name upon his first arrival to our page. The name is then stored in a cookie. Next time the visitor arrives at our page; we could have another popup saying something like: "Welcome John Doe!".

Example

Alert "Page is loaded" immediately after a page is loaded:

```
<!DOCTYPE html>
<html><head>
<script>
function display() {
  alert("Hello, Good Morning");
}
</script></head>
<body onLoad="display()">
<h1>Hello World!</h1>
</body></html>
```

- **onBlur()**

It occurs when an object loses focus. The onblur event is most often used with form validation code. When the user leaves a form field, it will occur. The onblur event is the opposite of the onfocus event.

Example

```
<!DOCTYPE html>
<html><body>
Enter your name: <input type="text" id="fname" onblur="display()">
<h3>When the focus is lost, the input text converts into upper case.</h3>
<script>
function display() {
  var x = document.getElementById("fname");
  x.value = x.value.toUpperCase();
}
</script></body><html>
```

- **onChange**

The onchange event occurs when the content of an element, the selection, or the checked state have changed. The onchange event occurs when the value of an element has been changed. For radio buttons and checkboxes, the onchange event occurs when the checked state has been changed. Only works on <input>, <textarea> and <select> elements.

Example

```
<!DOCTYPE html>
<html><body>
Enter your name: <input type="text" id="fname" onChange="display()">
<h3>When the focus is lost, the input text converts into upper case.</h3>
<script>
function display() {
  var x = document.getElementById("fname");
  x.value = x.value.toUpperCase();
```

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

- **onSubmit**

The onsubmit event occurs when the submit button in a form is clicked. The onSubmit event is used to validate ALL form fields before submitting it. Below is an example of how to use the onSubmit event. The checkForm() function will be called when the user clicks the submit button in the form. If the field values are not accepted, the submit should be cancelled. The function checkForm() returns either true or false. If it returns true the form will be submitted, otherwise the submit will be cancelled:

Example

```
<!DOCTYPE html>
<html><body>
<h3>When you submit the form, a function is triggered which alerts some
text.</h3>
<form action="server.php" onsubmit="display()">
  Enter name: <input type="text" name="fname">
  <input type="submit" value="Submit">
</form><script>
function display() {
  alert("The form was submitted Sucessfully....");
}
</script></body><html>
```

DOCUMENT OBJECT MODEL

The DOM is a W3C standard. The DOM defines a standard for accessing documents; DOM is a platform and language-neutral interface. DOM allows programs and scripts to dynamically access and update the content, structure, and style of a document. DOM standard is separated into 3 different parts:

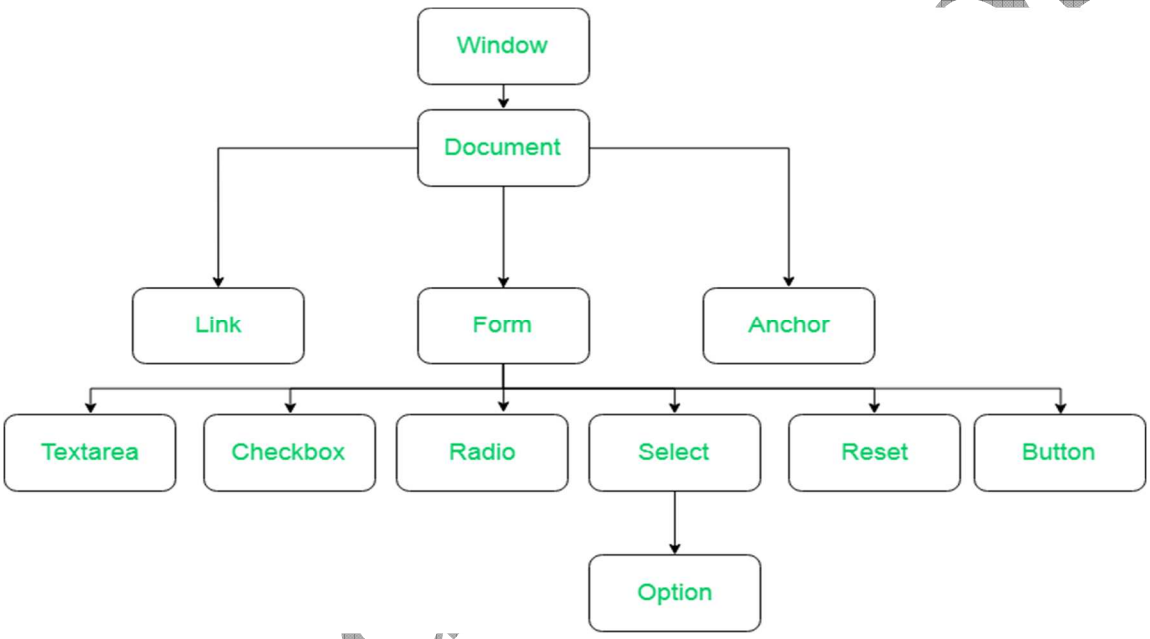
- Core DOM - standard model for all document types
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents

The Document Object Model (DOM) is a programming interface for HTML and XML documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects. That way, programming languages can connect to the page.

A Web page is a document. This document can be either displayed in the browser window or as the HTML source. But it is the same document in both cases. The Document Object Model (DOM) represents that same document so it can be manipulated. The DOM is an object-oriented

representation of the web page, which can be modified with a scripting language such as JavaScript.

The Document Object Model (DOM) is an application programming interface (API) for manipulating HTML and XML documents. The DOM represents a document as a tree of nodes. It provides API that allows you to add, remove, and modify parts of the document effectively. Note that the DOM is cross-platform and language-independent ways of manipulating HTML and XML documents. JavaScript can access all the elements in a webpage making use of Document Object Model (DOM). In fact, the web browser creates a DOM of the webpage when the page is loaded. The DOM model is created as a tree of objects like this:



String Object

A string is a sequence of letters, numbers, special characters or combination of all. It is a global object is used to store strings. A string can be any text inside double or single quotes:

String Property

Length:- Returns the length of a string

The following table lists the standard methods of the String object.

Method	Description
big()	Display text as if in a <big> element
bold()	Display text as if in a <bold> element
charAt()	Returns the character at the specified index.
concat()	Joins two or more strings, and returns a new string.
indexOf()	Returns the index of the first occurrence of the specified value in a string.
italics()	Display text as if in a <i> element

search()	Searches a string against a regular expression, and returns the index of the first match.
slice()	Extracts a portion of a string and returns it as a new string.
split()	Splits a string into an array of substrings.
substr()	Extracts the part of a string between the start index and a number of characters after it.
substring()	Extracts the part of a string between the start and end indexes.
toLowerCase()	Converts a string to lowercase letters.
toString()	Returns a string representing the specified object.
toUpperCase()	Converts a string to uppercase letters.
trim()	Removes whitespace from both ends of a string.

Example

```
<html><body>
<h2>The length property returns the length of a string:</h2>
<p></p>
<p id="demo"></p>
<script type="text/javascript">
var str="Hello, Good Morning to All, Have a Nice Day!<br>";
document.write(str);
document.write("<BR>The length is: <BR> " +str.length);
var pos = str.indexOf("Morning");
document.write("<br>The position of Morning is:");
document.write("<br>"+pos);
var up=str.toUpperCase();
document.write(up);

var lw=str.toLowerCase();
document.write(lw);
var sub=str.substring(15,30);
document.write(sub);
</script></body><html>
```

Date Object

The Date object is a datatype built into the JavaScript. The date objects are created with the new Date() constructor. By default, JavaScript will use the browser's time zone and display a date as a full text string:

Mon Jul 20 2020 17:33:21 GMT+0530 (India Standard Time)

Most methods simply allow us to get and set the year, month, day, hour, minute, second, and millisecond fields of the object.
There are 4 ways to create a new date object:

new Date()
new Date(milliseconds)
new Date(date string)
new Date(year, month, day, hours, minutes, seconds, milliseconds)

Method	Description
getDate()	It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of local time.
getDay()	It returns the integer value between 0 and 6 that represents the day of the week on the basis of local time.
getFullYears()	It returns the integer value that represents the year on the basis of local time.
getHours()	It returns the integer value between 0 and 23 that represents the hours on the basis of local time.
getMilliseconds()	It returns the integer value between 0 and 999 that represents the milliseconds on the basis of local time.
getMinutes()	It returns the integer value between 0 and 59 that represents the minutes on the basis of local time.
getMonth()	It returns the integer value between 0 and 11 that represents the month on the basis of local time.
getSeconds()	It returns the integer value between 0 and 60 that represents the seconds on the basis of local time.

Method	Description
setDate()	It sets the day value for the specified date on the basis of local time.
setDay()	It sets the particular day of the week on the basis of local time.
setFullYears()	It sets the year value for the specified date on the basis of local time.
setHours()	It sets the hour value for the specified date on the basis of local time.
setMilliseconds()	It sets the millisecond value for the specified date on the basis of local time.
setMinutes()	It sets the minute value for the specified date on the basis of local time.
setMonth()	It sets the month value for the specified date on the basis of local time.
setSeconds()	It sets the second value for the specified date on the basis of local time.
toString()	It returns the date in the form of string.

Example

```
<!DOCTYPE html>
<html> <body>
<h2>Date() Object</h2>
<p id="demo"></p>
<script>
var d = new Date();
document.write(d);
var d1 = new Date(2018, 11, 24, 10, 33, 30, 38);
document.getElementById("demo").innerHTML = d1;
var d2 = new Date(2018, 11, 24);
document.write("<br><br>"+d2);
var d = new Date();
document.write("The current Year is: "+d.getFullYear());
document.write("<br>The current Month is: "+d.getMonth());
document.write("<br>The current Date is: "+d.getDate());
document.write("<br>The current Hours is: "+d.getHours());
document.write("<br>The current Minute is: "+d.getMinutes());
document.write("<br>The current Second is: "+d.getSeconds());
document.write("<br>The current Milliseconds is: "+d.getMilliseconds());
document.write("<br>The current Day is: "+d.getDay());
</script></body><html>
```

Array Object

An array is a type of object used for storing multiple values in a single variable. A fixed-size sequential collection of elements of the same type. Each value in an array has a numeric position, called index. The array index starts from zero (0)

var cars = [“Hyundai", "Volvo", "BMW"];

Method	Description
concat()	Merge two or more arrays, and returns a new array.
findIndex()	Returns the index of the first element in an array that pass the test in a testing function.
join()	Joins all elements of an array into a string.
pop()	Removes the last element from an array, and returns that element.
push()	Adds one or more elements to the end of an array, and returns the array's new length.
reverse()	Reverses the order of the elements in an array.
slice()	Selects a part of an array, and returns the new array.
sort()	Sorts the elements of an array.
toString()	Converts an array to a string, and returns the result.

Example

```
<!DOCTYPE html>
<html><body>
<h3>Array Object </h3>
<p id="demo"></p>
<script>
	var fruits = ["Banana", "Orange", "Apple", "Mango"];
	var fname=["Raju", "Kumar", "Babu", "Arjun","Lakshmi", "Divya"];
	document.write("<br>"+fruits.toString());
	document.write("<br>The No. of elements in array is: "+fruits.length);
	document.write("<br>The deleted element is: "+fruits.pop());
	document.write("<br>"+fruits.toString());
	document.write("<br>Now No. of elements in array is: "+fruits.length);
	document.write("<br>Adding two elements");
	fruits.push("Grape","Chicku");
	document.write("<br>"+fruits.toString());
	document.write("<br>Now No. of elements in array is: "+fruits.length);
	document.write("<br>Reverseof the array is: "+fruits.reverse());
	document.write("<br>"+fruits.toString());
	document.write("<br>Sorted array is: "+fruits.sort());
	document.write("<br>"+fruits.toString());

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