JavaScript

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

JavaScript Syntax

JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page.

```
<script ...>
JavaScript code
</script>
```

JavaScript with type attribute:

```
<script type="text/javascript">
 JavaScript code
 </script>
```

Where to place JavaScript?

- JavaScript can be placed in the <head> section of an HTML page.
- JavaScript can be placed in the <body> section of an HTML page.
- JavaScript can also be placed in external files and then linked to HTML Page.

JavaScript in the Head Section

```
<!DOCTYPE html>
<html>
<head>
<script>
function abc()
{
   document.getElementById("ab").innerHTML = "LPU expects some better placements out of you";
}
</script>
</head>
<body>
<p id="ab">Welcome to LPU</p>
<input type="submit" onclick="abc()">
</body>
</html>
```

JavaScript in Body Section

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<p id="abc">Welcome to LPU</p>
<input type="submit" onclick="abc()">
<script>
function abc()
{
   document.getElementById("abc").innerHTML = "LPU expects some better placements
out of you";
}
</script>
</body>
</html>
```

NB: It is a good idea to place scripts at the bottom of the <body> element.
This can improve page load, because script compilation can slow down the display.

External JavaScript (b.js)

```
function abc()
{
   document.getElementById("abc").innerHTML = "LPU expects some better placements out of you";
}
```

a.html

```
<!DOCTYPE html>
<html>
<head>
<script src="b.js"></script>
</head>
<body>
<p id="abc">Welcome to LPU</p>
<input type="submit" onclick="abc()">
</body>
</html>
```

JavaScript Variables

To declare a variable, you use the var keyword followed by the variable name as follows:

```
var message;
let message = "Hello";
```

Constants

A constant holds a value that doesn't change. To declare a constant, you use the const keyword. When defining a constant, you need to initialize it with a value.

```
const workday = 5;
```

JavaScript Data Types

Primitive

- null
- undefined
- boolean
- number
- String
- Bigint

Complex

Object

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.

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.

Example

```
<!DOCTYPE html>
<html>
<head>
<script>
function abc()
{
        window.alert("Welcome to LPU");
        document.write("HTML Alert Dialog Box Example");
}
</script>
</head>
<body>
<input type="submit" onclick="abc()">
</body>
</html>
```

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.

Example

```
<!DOCTYPE html>
<html>
<head>
<script>
function abc()
{
   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>
</head>
<body>
<input type="submit" onclick="abc()">
</body>
</html>
```

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:

(i) a label which you want to display in the text box and

(ii) 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**.

Example

```
<!DOCTYPE html>
<html>
<head>
<script>
function abc()
{
   var retVal = prompt("Enter your name : ", "your name here");
            document.write("You have entered : " + retVal);

}
</script>
</head>
<body>
<input type="submit" onclick="abc()">
</body>
</html>
```

JavaScript Keywords

JavaScript statements often start with a **keyword** to identify the JavaScript action to be performed.

Most Commonly used keywords in Javascript:

Keyword	Description
Break	Terminates a switch or a loop
Continue	Jumps out of a loop and starts at the top
debugger	Stops the execution of JavaScript, and calls (if available) the debugging function
do ... while	Executes a block of statements, and repeats the block, while a condition is true
for	Marks a block of statements to be executed, as long as a condition is true
function	Declares a function
if ... else	Marks a block of statements to be executed, depending on a condition
return	Exits a function
switch	Marks a block of statements to be executed, depending on different cases
try ... catch	Implements error handling to a block of statements
var	Declares a variable

JavaScript keywords are reserved words. Reserved words cannot be used as names for variables.

JavaScript Comments

JavaScript comments can be used to explain JavaScript code, and to make it more readable.
JavaScript comments can also be used to prevent execution, when testing alternative code.

Single Line Comments
Single line comments start with //.
Any text between // and the end of the line will be ignored by JavaScript (will not be executed).

Multi Line Comments

Multi-line comments start with /* and end with */.
Any text between /* and */ will be ignored by JavaScript.

JavaScript Variables

JavaScript variables are containers for storing data values.

Example:

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Variables</h1>
<p id=“abc"></p>
<script>
var x = 5;
var y = 6;
var z = x + y;
document.getElementById(“abc").innerHTML = z;
</script>
</body>
</html>
```

Example:

```
<!DOCTYPE html>
<html>
<body>
<p id=“abc"></p>
<script>
var a = 5;
var b = 6;
var c = a+ b;
document.getElementById(“abc").innerHTML = "The total is: " + c;
</script>
</body>
</html>
```

JavaScript Identifiers

All JavaScript **variables** must be **identified** with **unique names**.
These unique names are called **identifiers**.

The general rules for constructing names for variables (unique identifiers) are:
Names can contain letters, digits, underscores, and dollar signs.
Names must begin with a letter
Names can also begin with $ and _
Names are case sensitive (y and Y are different variables)
Reserved words (like JavaScript keywords) cannot be used as names

NB: JavaScript identifiers are case-sensitive.

Declaring (Creating) JavaScript Variables

Creating a variable in JavaScript is called "declaring" a variable.
You declare a JavaScript variable with the **var** keyword:
var number;
After the declaration, the variable has no value. (Technically it has the value of **undefined**)

Re-Declaring JavaScript Variables

If you re-declare a JavaScript variable, it will not lose its value.

The variable Name will still have the value “Rahul" after the execution of these statements:

Example

```
<!DOCTYPE html>
<html>
<body>
<p>If you re-declare a JavaScript variable, it will not lose its value.</p>
<p id=“abc"></p>
<script>
var Name = “Rahul";
var  Name;
document.getElementById(“abc").innerHTML = Name;
</script>
</body>
</html>
```

JavaScript Arithmetic

Example

```
<!DOCTYPE html>
<html>
<body>
<p>The result of adding 5 + 10 + 20:</p>
<p id=“abc"></p>
<script>
var x = 5 + 10 + 20;
document.getElementById(“abc").innerHTML = x;
</script>
</body>
</html>
```

Write JS code to concatenate two strings.

You can also add strings, but strings will be concatenated:

Example

```
<!DOCTYPE html>
<html>
<body>
<p id=“abc"></p>
<script>
var x = “Rohit" + " " + “Sharma";
document.getElementById("demo").innerHTML = x;
</script>
</body>
</html>
```

Example:

```
<!DOCTYPE html>
<html>
<body>
  <p id=“abc"></p>
<script>
var x = "5" + 8 + 4;
document.getElementById(“abc").innerHTML = x;
</script>
</body>
</html>
```

Example:

```
<!DOCTYPE html>
<html>
<body>
  <p id=“abc"></p>
<script>
var x =  8 + 4+ "5";
document.getElementById(“abc").innerHTML = x;
</script>
</body>
</html>
```

JavaScript Operators

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic on numbers

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Example

```
<!DOCTYPE html>
<html>
<body>
<p id=“abc"></p>
<script>
name1 = “Welcome to  ";
name1 += “Lovely Professional University";
document.getElementById(“abc").innerHTML = name1;
</script>
</body>
</html>
```

Adding Strings and Numbers

Adding two numbers, will return the sum, but adding a number and a string will return a string:

Example

```
<!DOCTYPE html>
<html>
<body>
 <p id=“abc"></p>
<script>
var x = 5 + 10;
var y = "5" + 10;
var z = "Hello" + 5;
document.getElementById(“abc").innerHTML = x + "<br>" + y + "<br>" + z;
</script>
</body>
</html>
```

JavaScript Comparison and Logical 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

Operator Precedence

Operator precedence describes the order in which operations are performed in an arithmetic expression.

Example

```
<!DOCTYPE html>
<html>
<body>
<p id="abc"></p>
<script>
document.getElementById("abc").innerHTML = 100+50*3;
</script>
</body>
</html>

<!DOCTYPE html>
<html>
<body>
<p id="abc"></p>
<script>
document.getElementById("abc").innerHTML = (100+50)*3;
</script>
</body>
</html>
```

JavaScript Booleans

Booleans can only have two values: true or false.

Example

```
var x = true;
var y = false;
```

Booleans are often used in conditional testing.

JavaScript Arrays

JavaScript arrays are written with square brackets.

Array items are separated by commas.

Example

```
<!DOCTYPE html>
<html>
<body>
<p id=“abc"></p>
<script>
var names = [“Rohit",“Rahul",“Virat”];
document.getElementById(“abc").innerHTML = names[0];
</script>
</body>
</html>
```

JavaScript Objects

JavaScript objects are written with curly braces.

Object properties are written as name:value pairs, separated by commas.

Example:

```
<!DOCTYPE html>
<html>
<body>
<p id=“abc"></p>
<script>
var person = {
   firstName : "Rahul",
   lastName  : “Sharma",
   age       : 50
};

document.getElementById(“abc").innerHTML = person.firstName + " is " + person.age + "
years old.";
</script>
</body>
</html>
```

The typeof Operator

You can use the JavaScript **typeof** operator to find the type of a JavaScript variable:

Example:

```
<!DOCTYPE html>
<html>
<body>
<p id=“abc"></p>
<script>
document.getElementById(“abc").innerHTML =
typeof "Rahul" + "<br>" +
typeof 3.14 + "<br>" +
typeof false + "<br>" +
typeof [1,2,3,4] + "<br>" +
typeof {name:‘Rahul', age:34};
</script>
</body>
</html>
```

Date Object Methods:
Method/Description

getDate()
Returns the day of the month (from 1-31)
getDay()
Returns the day of the week (from 0-6)
getFullYear()
Returns the year
getHours()
Returns the hour (from 0-23)
getMilliseconds()
Returns the milliseconds (from 0-999)
getMinutes()
Returns the minutes (from 0-59)
getMonth()
Returns the month (from 0-11)
getSeconds()
Returns the seconds (from 0-59)
getTime()
Returns the number of milliseconds since midnight Jan 1 1970, and a specified date

getTimezoneOffset()
Returns the time difference between UTC time and local time, in minutes
getUTCDate()
Returns the day of the month, according to universal time (from 1-31)
getUTCDay()
Returns the day of the week, according to universal time (from 0-6)
getUTCFullYear()
Returns the year, according to universal time
getUTCHours()
Returns the hour, according to universal time (from 0-23)
getUTCMilliseconds()
Returns the milliseconds, according to universal time (from 0-999)
getUTCMinutes()
Returns the minutes, according to universal time (from 0-59)
getUTCMonth()
Returns the month, according to universal time (from 0-11)
getUTCSeconds()
Returns the seconds, according to universal time (from 0-59)
now()
Returns the number of milliseconds since midnight Jan 1, 1970

parse()
Parses a date string and returns the number of milliseconds since January 1, 1970
setDate()
Sets the day of the month of a date object
setFullYear()
Sets the year of a date object
setHours()
Sets the hour of a date object
setMilliseconds()
Sets the milliseconds of a date object
setMinutes()
Set the minutes of a date object
setMonth()
Sets the month of a date object
setSeconds()
Sets the seconds of a date object
setTime()
Sets a date to a specified number of milliseconds after/before January 1, 1970
setUTCDate()
Sets the day of the month of a date object, according to universal time
setUTCFullYear()
Sets the year of a date object, according to universal time

setUTCHours()
Sets the hour of a date object, according to universal time
setUTCMilliseconds()
Sets the milliseconds of a date object, according to universal time
setUTCMinutes()
Set the minutes of a date object, according to universal time
setUTCMonth()
Sets the month of a date object, according to universal time
setUTCSeconds()
Set the seconds of a date object, according to universal time
toDateString()
Converts the date portion of a Date object into a readable string
toISOString()
Returns the date as a string, using the ISO standard
toJSON()
Returns the date as a string, formatted as a JSON date

toLocaleDateString()
Returns the date portion of a Date object as a string, using locale conventions
toLocaleTimeString()
Returns the time portion of a Date object as a string, using locale conventions
toLocaleString()
Converts a Date object to a string, using locale conventions
toString()
Converts a Date object to a string
toTimeString()
Converts the time portion of a Date object to a string
toUTCString()
Converts a Date object to a string, according to universal time
UTC()
Returns the number of milliseconds in a date since midnight of January 1, 1970, according to UTC time
valueOf()
Returns the primitive value of a Date object

Conditional Statements

In JavaScript we have the following conditional statements:
Use **if** to specify a block of code to be executed, if a specified condition is true.
Use **else** to specify a block of code to be executed, if the same condition is false.
Use **else if** to specify a new condition to test, if the first condition is false.
Use **switch** to specify many alternative blocks of code to be executed.

The if Statement

Use the **if** statement to specify a block of JavaScript code to be executed if a condition is true.
Syntax

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

The else Statement:

Use the **else** statement to specify a block of code to be executed if the condition is false.

```
if (condition) {
   block of code to be executed if the condition is true
} else {
   block of code to be executed if the condition is false
}
```

The else if Statement

Use the **else if** statement to specify a new condition if the first condition is false.

Syntax

```
if (condition1) {
   block of code to be executed if condition1 is true
} else if (condition2) {
   block of code to be executed if the condition1 is false and condition2 is true
} else {
   block of code to be executed if the condition1 is false and condition2 is false
}
```

JavaScript Switch Statement:

The switch statement is used to perform different actions based on different conditions.
The JavaScript Switch Statement
Use the switch statement to select one of many blocks of code to be executed.
Syntax

```
switch(expression) {
   case n:
      code block
      break;
   case n:
      code block
      break;
   default:
      default code block
}
```

JavaScript Loops

JavaScript supports different kinds of loops:
for - loops through a block of code a number of times
for/in - loops through the properties of an object
while - loops through a block of code while a specified condition is true
do/while - also loops through a block of code while a specified condition is true

The For Loop

The for loop is often the tool you will use when you want to create a loop.
The for loop has the following syntax:

```
for (statement 1; statement 2; statement 3) {
   code block to be executed
}
```

The While Loop

The while loop loops through a block of code as long as a specified condition is true.
Syntax

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

The Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

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

The For/In Loop

The JavaScript for/in statement loops through the properties of an object:

```
<!DOCTYPE html>
<html>
<body>
<p id="abc"></p>
<script>
var txt=“”;
var person = {fname:"Rahul", lname:"Sharma", age:25};
var x;
for (x in person) {
   txt += person[x] + " ";
}
document.getElementById("abc").innerHTML = txt;
</script>

</body>
</html>
```

Events in JavaScript

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.

onclick Event

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

Example

```
<!DOCTYPE html>
<html>
  <head>

    <script type="text/javascript">

        function abc() {
          alert("Welcome to the School of CSE")
        }

    </script>
  </head>
  <body>
    <form>
      <input type="button" onclick="abc()" value="Test" />
    </form>

  </body>
</html>
```

onsubmit Event

onsubmit is an event that occurs when you try to submit a form. You can put your form validation against this event type.

Example

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript">

        function validate() {
          alert("Validated");
        }
    </script>
  </head>
  <body>
    <form method="POST" onsubmit="return validate()">
      <input type="submit" value="Submit" >
    </form>
  </body>
</html>
```

onmouseover and onmouseout Events

These two event types will help you create nice effects with images or even with text as well. The **onmouseover** event triggers when you bring your mouse over any element and the **onmouseout** triggers when you move your mouse out from that element.

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript">
        function over() {
          document.write ("Mouse Over");
        }
    </script>
  </head>
  <body>
    <div onmouseover="over()">
            Hello LPU
    </div>
  </body>
</html>
```

Example

```
<!DOCTYPE html>
<html>
  <head>

    <script type="text/javascript">

        function out() {
          document.write ("Mouse Out");
        }
    </script>
  </head>
  <body>
    <div onmouseout="out()">
            Hello LPU
    </div>
  </body>
</html>
```

onkeypress Event

The onkeypress event occurs when the user presses a key (on the keyboard).

Example

```
<!DOCTYPE html>
<html>
  <head>

    <script type="text/javascript">

        function abc() {
          document.write ("Pressed");
        }
    </script>
  </head>
  <body>
    <input type="text" onkeypress="abc()">
  </body>
</html>
```

onload Event

The onload event occurs when an object has been loaded.
onload is most often used within the <body> element to execute a script once a web page has completely loaded all content (including images, script files, CSS files, etc.).
Example

```
<!DOCTYPE html>
<html>
  <head>

    <script type="text/javascript">

        function abc() {
          document.write ("Example of Text on Page Loading");
        }
    </script>
  </head>
  <body onload="abc()">
  </body>
</html>
```

onreset Event

When you reset the form, a function is triggered which alerts some text

```
<!DOCTYPE html>
<html>
<body>
<form onreset=“abc()">
  Enter name: <input type="text">
  <input type="reset">
</form>

<script>
function abc()
 {
    alert("The form was reset");
}
</script>
</body>
</html>
```

JavaScript Form Validation

Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button.

If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process which used to put a lot of burden on the server.

JavaScript provides a way to validate form's data on the client's computer before sending it to the web server.

Example1: Matching Password and Confirm Password

```
<script type="text/javascript">
      function CanSubmit() {
         //alert("ok");
         var pwd = document.forms[0].txtPassword.value
         var cpwd = document.forms[0].txtConfirmPassword.value
         if (pwd == cpwd)
            return true;
         else {
            alert("Please make sure that Password and Confirm Password are Same");
            return false;
         }
      }
   </script>

<form action="" method="post" onsubmit ="return CanSubmit()">
Password: <input type="password" name="txtPassword" value="" /> <br />
ConfirmPassword: <input type="password" name="txtConfirmPassword" value="" />
<br />
<input type="submit" name="btnSubmit" value="Submit" />
</form>
```

Example 2: Providing alert before data will be deleted

```
function CanDelete()
    {
        return confirm("Are you Sure to delete your Data");
     }

<input type="submit" name="btnDelete" value="Delete" onclick ="return CanDelete()" />
```

Example 3: Validate Textboxes for anydata in arithmetic operations

```
function ValidateMathFunction()
    {
        var FN = document.forms[1].txtFN.value;
        var SN = document.forms[1].txtSN.value;
        if (FN == "" || SN == "") {
            alert("Please ensure that data is inserted in both textboxes");
            return false;
        }
        else
            return true;
      }
<form action="/" method="post" onsubmit="return ValidateMathFunction()">
Enter First Number <input type="text" name="txtFN" value="" /> <br />
Enter Second Number <input type="text" name="txtSN" value="" /><br />
 <input type="submit" name="btnAdd" value="+" />
<input type="submit" name="btnSub" value="-" />
<input type="submit" name="btnMul" value="*" />
<input type="submit" name="btnDel" value="/" />
</form>
```

Example 4 Denominator cant be zero

```
function CheckDenominator() {
        var SN = document.forms[1].txtSN.value;
        if (SN == 0) {

           alert("Denominator cant be zero");
           return false;
        }
        else
           return true;
     }

<input type="submit" name="btnDel" value="/" onclick="return CheckDenominator()"/>
```

Example 5 Whether the values entered in texboxes are numbers are not?

```
function ValidateMathFunction()
   {
        var FN = document.forms[1].txtFN.value;
        var SN = document.forms[1].txtSN.value;
        if (FN == "" || SN == ""|| isNaN(FN)||isNaN(SN)) {
           alert("Please ensure that valid data is inserted in both textboxes");
           return false;
        }
        else
           return true;
     }
```

Example 6 Validating Email using Regular Expressions

```
<!DOCTYPE html>
<html>
  <head>
  <script type="text/javascript">
      function validateEmail()
   {
      var emailTextBox = document.getElementById("txtEmail");
      var email = emailTextBox.value;
      var emailRegEx =
/^(([^<>()[\]\\.,;:\s@\"]+(\.[^<>()[\]\\.,;:\s@\"]+)*)|(\".+\"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;

      emailTextBox.style.color = "white";
```

```
 if (emailRegEx.test(email))
       {
          emailTextBox.style.backgroundColor = "green";
       }
       else
       {
          emailTextBox.style.backgroundColor = "red";
       }
    }
    </script>


   </head>
   <body>
   Email : <input type="text" id="txtEmail" onkeyup="validateEmail()" />

   </body>
</html>
```

JavaScript Timing Events

In JavaScript a piece of code can be executed at specified time interval. For example, you can call a specific JavaScript function every 1 second. This concept in JavaScript is called timing events.

The global window object has the following 2 methods that allow us to execute a piece of JavaScript code at specified time intervals.

setInterval(func, delay) - Executes a specified function, repeatedly at specified time interval. This method has 2 parameters. The **func** parameter specifies the name of the function to execute. The **delay** parameter specifies the time in milliseconds to wait before calling the specified function.

setTimeout(func, delay) - Executes a specified function, after waiting a specified number of milliseconds. This method has 2 parameters. The func parameter specifies the name of the function to execute. The delay parameter specifies the time in milliseconds to wait before calling the specified function. The actual wait time (delay) may be longer.

The following code displays current date and time in the div tag.

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
   function getCurrentDateTime()
   {
      document.getElementById("timeDiv").innerHTML = new Date();
   }

</script>
</head>
<body>
<div id="timeDiv" ></div>
<input type="submit" onclick="getCurrentDateTime()">
</body>
</html>
```

At the moment the time is static.

To make the time on the page dynamic, modify the script as shown below. Notice that the time is now updated every second. In this example, we are using setInterval() method and calling getCurrentDateTime() function every 1000 milli-seconds.

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
   setInterval(getCurrentDateTime, 1000);

   function getCurrentDateTime()
   {
      document.getElementById("timeDiv").innerHTML = new Date();
   }
</script>
</head>
<body>
<div id="timeDiv" ></div>
</body>
</html>
```

Starting and stopping the clock with button click : In this example, **setInterval()**method returns the **intervalId** which is then passed to **clearInterval()** method. When you click the **"Start Clock"** button the clock is updated with new time every second, and when you click **"Stop Clock"** button it stops the clock.

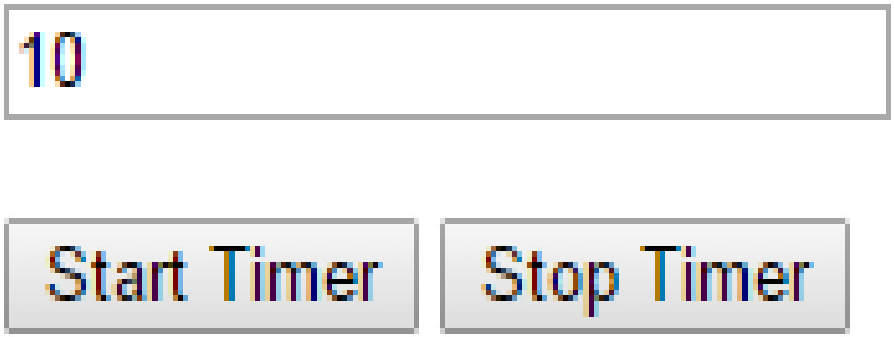
```
 <!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
   var intervalId;

   function startClock()
   {
      intervalId = setInterval(getCurrentDateTime, 1000);
   }

   function stopClock()
   {
      clearInterval(intervalId);
   }
```

```
 function getCurrentDateTime()
   {
      document.getElementById("timeDiv").innerHTML= new Date();
   }
</script>
</head>
<body>
<div id="timeDiv" ></div> <br />
<input type="button" value="Start Clock" onclick="startClock()" />
<input type="button" value="Stop Clock" onclick="stopClock()" />
</body>
</html>
```

Now let's look at example of using **setTimeout**() and **clearTimeout**() functions. The syntax and usage of these 2 functions is very similar to **setInterval**() and **clearInterval**().

Countdown timer example :

When we click **"Start Timer"** button, the value 10 displayed in the textbox must start counting down. When click **"Stop Timer"** the countdown should stop. When you click **"Start Timer"** again, it should start counting down from where it stopped and when it reaches ZERO, it should display **Done** in the textbox and function should return.



```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
   var intervalId;

   function startTimer(controlId)
   {
      var control = document.getElementById(controlId);
      var seconds = control.value;
      seconds = seconds - 1;
      if (seconds == 0)
      {
         control.value = "Done";
         return;
      }
      else
      {
         control.value = seconds;
      }
```

```
intervalId = setTimeout(function () { startTimer('txtBox'); }, 1000);
    }

    function stopTimer()
    {
        clearTimeout(intervalId);
    }
</script>
</head>
<body>
<input type="text" value="10" id="txtBox" />
<br /><br />
<input type="button" value="Start Timer" onclick="startTimer('txtBox')" />
<input type="button" value="Stop Timer" onclick="stopTimer()" />
</body>
</html>
```

JavaScript Image Slideshow

The slideshow should be as shown in the image below.

Start Slide Show Stop Slide Show

When you click **"Start Slide Show"** button the image slideshow should start and when you click the **"Stop Slide Show"** button the image slideshow should stop.

```
<html>
<head>
<script type="text/javascript">
   var intervalId;

   function startImageSlideShow()
   {
      intervalId = setInterval(setImage, 500);
   }

   function stopImageSlideShow()
   {
      clearInterval(intervalId);
   }

   function setImage()
   {
      var imageSrc = document.getElementById("image").getAttribute("src");
      var currentImageNumber = imageSrc.substring(imageSrc.lastIndexOf("/") +
1,imageSrc.lastIndexOf("/") + 2);
```

```
 if (currentImageNumber == 6)
      {
         currentImageNumber = 0;
      }
      document.getElementById("image").setAttribute("src", "Images/"  +
(Number(currentImageNumber) + 1) + ".jpg");
   }
</script>
</head>

<body>
<img id="image" src="Images/1.jpg" style="width: 500px; height: 150px" />
<br /> <br /> <br />
<input type="button" value="Start Slide Show" onclick="startImageSlideShow()" />
<input type="button" value="Stop Slide Show" onclick="stopImageSlideShow()" />
</body>
</html>
```

Recursive Function in JavaScript

Recursion is a programming concept that is applicable to all programming languages including JavaScript.

Recursive function is function that calls itself.

When writing recursive functions **there must be a definite break condition**, otherwise we risk creating infinite loops.

Example : Computing the factorial of a number without recursion

```
<html>
<head>
<script>
function factorial(n)
{
   if (n == 0 || n == 1)
   {
      return 1;
   }
```

```
var result = n;
   while (n > 1)
   {
      result = result * (n - 1)
      n = n - 1;
   }

   return result;

}
function abc()
{
document.write(factorial(5));
}
</script>
<head>
<body>
<input type="submit" onclick="abc()">
</body>
</html>
```

Example : Computing the factorial of a number using a recursive function

```
<html>
<head>
<script>
function factorial(n)
{
   if (n == 0 || n == 1)
   {
      return 1;
   }
   return n * factorial(n - 1);
}
function abc()
{
document.write(factorial(5));
}
</script>
<head>
<body>
<input type="submit" onclick="abc()">
</body>
</html>
```

Error handling in JavaScript

Use try/catch/finally to handle runtime errors in JavaScript. These runtime errors are called exceptions. An exception can occur for a variety of reasons. For example, referencing a variable or a method that is not defined can cause an exception.

The JavaScript statements that can possibly cause exceptions should be wrapped inside a try block. When a specific line in the try block causes an exception, the control is immediately transferred to the catch block skipping the rest of the code in the try block.

JavaScript try catch example :

```
<html>
<head>
<script>
function abc()
{
  try
{
   // Referencing a function that does not exist cause an exception
   document.write(sayHello());
   // Since the above line causes an exception, the following line will not be executed
   document.write("This line will not be executed");
}
```

```
// When an exception occurs, the control is transferred to the catch block
catch (e)
{
   document.write("Description = " + e.description + "<br/>");
   document.write("Message = " + e.message + "<br/>");
   document.write("Stack = " + e.stack + "<br/><br/>");
}
 }
</script>
<head>
<body>
<input type="submit" onclick="abc()">
</body>
</html>
```

Please note : A try block should be followed by a catch block or finally block or both.

finally block is guaranteed to execute irrespective of whether there is an exception or not. It is generally used to clean and free resources that the script was holding onto during the program execution. For example, if your script in the try block has opened a file for processing, and if there is an exception, the finally block can be used to close the file.

Example with finally block

```
<html>
<head>
<script>
function abc()
{
  try
{
   // Referencing a function that does not exist cause an exception
   document.write(sayHello());
   // Since the above line causes an exception, the following line will not be executed
   document.write("This line will not be executed");
}
// When an exception occurs, the control is transferred to the catch block
catch (e)
{
   document.write("Description = " + e.description + "<br/>");
   document.write("Message = " + e.message + "<br/>");
   document.write("Stack = " + e.stack + "<br/><br/>");
}
```

```
finally
{
   document.write("This line is guaranteed to execute");
}

 }
</script>
<head>
<body>
<input type="submit" onclick="abc()">
</body>
</html>
```

Syntax errors and exceptions in JavaScript

try/catch/finally block can catch exceptions but not syntax errors.

Example : In the example, below we have a syntax error - missing the closing parentheses. The associated catch block will not catch the syntax errors.

```
<html>
<head>
<script>
function abc()
{
  try
{
   alert("Hello";
}
// When an exception occurs, the control is transferred to the catch block
catch (e)
{
   document.write("JavaScript syntax errors cannot be caught in the catch block");
}
 }
</script>
<head>
<body>
<input type="submit" onclick="abc()">
</body>
</html>
```

JavaScript throw statement : Use the throw statement to raise a customized exceptions.

JavaScript throw exception example :

```
<html>
<head>
<script>
function divide()
{
   var numerator = Number(prompt("Enter numerator"));
   var denominator = Number(prompt("Enter denominator"));

   try
   {
      if (denominator == 0)
      {
         throw {
            error: "Divide by zero error",
            message: "Denominator cannot be zero"
         };
      }
```

```
else
      {
         alert("Result = " + (numerator / denominator));
      }
   }
   catch (e)
   {
      document.write(e.error + "<br/>");
      document.write(e.message + "<br/>");
   }
}
</script>
<head>
<body>
<input type="submit" onclick="divide()">
</body>
</html>
```