

What is the use of JavaScript in ASP.NET Web Application

[Project Name: 01_UseOfJavascript]

Example : The following web form captures user First Name, Last Name & Email.

First Name	<input type="text"/>
Last Name	<input type="text"/>
Email	<input type="text"/>
<input type="submit" value="Submit"/>	

When we click the **Submit** button, we want to save the data to the following SQL Server **Users** table

ID	FirstName	LastName	Email
1	Test	Test	Test@Test.com

Here are the steps

Step 1 : Create Users Table

Create table Users

```
(  
    ID int primary key identity,  
    FirstName nvarchar(50),  
    LastName nvarchar(50),  
    Email nvarchar(50)  
)
```

Step 2 : Create a stored procedure to insert user record into Users table

Create procedure spInsertUser

```
@FirstName nvarchar(50),  
@LastName nvarchar(50),  
@Email nvarchar(50)
```

as

Begin

Insert into Users values (@FirstName, @LastName, @Email)

End

Step 3 : Create a new empty asp.net web application project. Name it Demo.

Step 4 : In the web.config file include connection string to the database

```
<add name="CS"
      connectionString="server=.;database=Sample;integrated security=SSPI"/>
```

Step 5 : Add a WebForm to the project. Copy and paste the following HTML in the <form> tag of the webform.

```
<table style="border:1px solid black; font-family:Arial">
  <tr>
    <td>
      <b>First Name</b>
    </td>
    <td>
      <asp:TextBox ID="txtFirstName" runat="server"></asp:TextBox>
      <asp:Label ID="lblFirstName" runat="server" ForeColor="Red"></asp:Label>
    </td>
  </tr>
  <tr>
    <td>
      <b>Last Name</b>
    </td>
    <td>
      <asp:TextBox ID="txtLastName" runat="server"></asp:TextBox>
      <asp:Label ID="lblLastName" runat="server" ForeColor="Red"></asp:Label>
    </td>
  </tr>
  <tr>
    <td>
      <b>Email</b>
    </td>
    <td>
      <asp:TextBox ID="txtEmail" runat="server"></asp:TextBox>
      <asp:Label ID="lblEmail" runat="server" ForeColor="Red"></asp:Label>
    </td>
  </tr>
  <tr>
    <td colspan="2">
      <asp:Button ID="btnSubmit" runat="server" Text="Submit" onclick="btnSubmit_Click" />
    </td>
  </tr>
</table>
```

```
</td>
</tr>
</table>
```

Step 6 : Copy and paste the following code in the code-behind file.

```
using System;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
```

```
namespace Demo
```

```
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {}
    }
}
```

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    if (ValidateForm())
    {
        SaveData();
    }
}
```

```
private void SaveData()
{
    string cs = ConfigurationManager.ConnectionStrings["CS"].ConnectionString;
    using (SqlConnection con = new SqlConnection(cs))
    {
        SqlCommand cmd = new SqlCommand("spInsertUser", con);
        cmd.CommandType = CommandType.StoredProcedure;
```

```
        SqlParameter paramFirstName = new
            SqlParameter("@FirstName", txtFirstName.Text);
        SqlParameter paramLastName = new
            SqlParameter("@LastName", txtLastName.Text);
        SqlParameter paramEmail = new
            SqlParameter("@Email", txtEmail.Text);
```

```
cmd.Parameters.Add(paramFirstName);
cmd.Parameters.Add(paramLastName);
cmd.Parameters.Add(paramEmail);

con.Open();
cmd.ExecuteNonQuery();
}
}
```

```
private bool ValidateForm()
{
    bool ret = true;
    //System.Threading.Thread.Sleep(3000);

    if (string.IsNullOrEmpty(txtFirstName.Text))
    {
        ret = false;
        lblFirstName.Text = "First Name is required";
    }
    else
    {
        lblFirstName.Text = "";
    }

    if (string.IsNullOrEmpty(txtLastName.Text))
    {
        ret = false;
        lblLastName.Text = "Last Name is required";
    }
    else
    {
        lblLastName.Text = "";
    }

    if (string.IsNullOrEmpty(txtEmail.Text))
    {
        ret = false;
        lblEmail.Text = "Email required";
    }
    else
    {

```

```

        lblEmail.Text = "";
    }

    return ret;
}
}
}

```

Run the application. Leave all the fields blank and click the Submit button. The form is posted to the web server, the server validates the form and since all the fields are required fields the user is presented with the following error messages.

First Name	<input type="text"/>	First Name is required
Last Name	<input type="text"/>	Last Name is required
Email	<input type="text"/>	Email required
<input type="button" value="Submit"/>		

In this case the form validation is done on the server. Just to validate the form, there is a round trip between the client browser and the web server. The request has to be sent over the network to the web server for processing. This means if the network is slow or if the server is busy processing other requests, the end user may have to wait a few seconds.

Isn't there a way to validate the form on the client side?

Yes, we can and that's when we use JavaScript. JavaScript runs on the client browser.

Copy and paste the following JavaScript function in the head section of WebForm1.aspx

```

<script type="text/JavaScript" language="JavaScript">
function ValidateForm()
{
    var ret = true;
    if (document.getElementById("txtFirstName").value == "")
    {
        document.getElementById("lblFirstName").innerText = "First Name is required";
        ret = false;
    }
    else
    {
        document.getElementById("lblFirstName").innerText = "";
    }

    if (document.getElementById("txtLastName").value == "")
    {

```

```

        document.getElementById("lblLastName").innerText = "Last Name is required";
        ret = false;
    }
    else
    {
        document.getElementById("lblLastName").innerText = "";
    }

    if (document.getElementById("txtEmail").value == "")
    {
        document.getElementById("lblEmail").innerText = "Email is required";
        ret = false;
    }
    else
    {
        document.getElementById("lblEmail").innerText = "";
    }

    return ret;
}
</script>

```

For the Submit button set **OnClick="return ValidateForm()"**. This calls the JavaScript ValidateForm() function. The JavaScript code runs on the client browser and validates the form fields.

If the fields are left blank the user is presented with an error message right away. This is much faster because there is no round trip between the client and the web server. This also means the load on the server is reduced, and we are using the client machine processing power. If all the form fields are populated the form will be submitted to the server and the server saves the data to the database table.

```

<asp:Button ID="btnSubmit" runat="server" Text="Submit"
    OnClick="return ValidateForm()" onclick="btnSubmit_Click" />

```

What are the advantages of using JavaScript

1. Form validation can be done on the client side, which reduces the unnecessary round trips between the client and the server. This also means the load on the server is reduced and the application is more responsive.
2. JavaScript uses the client machine processing power.

3. With JavaScript partial page updates are possible i.e only portions of the page can be updated, without reloading the entire web form. This is commonly called as AJAX.

4. JavaScript can also be used to animate elements on a page. For example, show or hide elements and sections of the page.

Why do we need both Client and Server side validation

Client side validation can be very easily bypassed by disabling JavaScript on a client browser. For example the following are the steps to disable JavaScript in Google chrome.

1. Open Google Chrome browser
2. Click on the Customize button on the top right hand corner of the browser
3. Select Settings from the context menu
4. Type JavaScript, in the Search Settings textbox
5. Click on "Content Settings" button
6. Under "JavaScript" section select "Do not allow any site to run JavaScript" radio button.
7. Close "Content Settings" window

In the application comment the call to **ValidatForm()** method. This is the server side method that validates form input. At this point the code in **btnSubmit_Click()** method should be as shown below.

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    //if (ValidateForm())
    //{
        SaveData();
    //}
}
```

So, at the moment

1. We have disabled JavaScript on the client browser
2. We don't have any server side validation

Run the application and click the Submit button, without filling any data. Notice that an empty row is inserted into Users table.

ID	FirstName	LastName	Email
1	Test	Test	Test@Test.com
2			

This is because client side validation is bypassed as we have disabled JavaScript and we also don't have any server side method validating the form. This is one of the reasons why we always want to have both client side and server side validation.

If JavaScript is disabled and if we don't have any server side validation, there could be different threats ranging from storing invalid data to security vulnerabilities.

Client-side validation provides better user experience as it reduces the unnecessary round trips between the client and the server. So client side validation is nice to have.

However, if JavaScript is disabled or if the user is making a request using tools like fiddler we still want to validate the form before saving data. So, server side validation should always be there irrespective of whether we have client side validation or not.

Disadvantages of JavaScript

[Project Name: 02_BrowserCompatibilityIssues]

There are two main **disadvantages of JavaScript**.

Security : JavaScript runs on the client machine. So a malicious user may use JavaScript to do a variety of things like tracking your browsing history, stealing passwords etc. This is one of the main reasons why people disable JavaScript.

Browser Compatibility : Not all browsers treat the same piece of JavaScript in the same manner. This means the functionality and the user interface may vary from browser to browser. That is why cross-browser testing is very important. However, with JavaScript libraries like jQuery Browser Compatibility is no longer a major issue.

JavaScript Browser Compatibility Examples

Example 1 : innerText property is supported in IE & Chrome, but not in Firefox. This means the **ValidatForm()** JavaScript function, will only work in IE & Chrome but not in Firefox.

```
function ValidatForm()
{
    var ret = true;
    if (document.getElementById("txtFirstName").value == "")
    {
        document.getElementById("lblFirstName").innerText = "First Name is required";
        ret = false;
    }
    else {
        document.getElementById("lblFirstName").innerText = "";
    }
}
```



```

if (document.getElementById("txtLastName").value == "")
{
    document.getElementById("lblLastName").innerText = "Last Name is required";
    ret = false;
}
else
{
    document.getElementById("lblLastName").innerText = "";
}

if (document.getElementById("txtEmail").value == "")
{
    document.getElementById("lblEmail").innerText = "Email is required";
    ret = false;
}
else
{
    document.getElementById("lblEmail").innerText = "";
}

return ret;
}

```

For the above JavaScript function to work in all the browsers that is in IE, Chrome & Firefox, replace **innerText** property with **textContent** as shown below.

```

function ValidatForm() {
    var ret = true;
    if (document.getElementById("txtFirstName").value == "")
    {
        document.getElementById("lblFirstName").textContent = "First Name is required";
        ret = false;
    }
    else
    {
        document.getElementById("lblFirstName").textContent = "";
    }

    if (document.getElementById("txtLastName").value == "")
    {
        document.getElementById("lblLastName").textContent = "Last Name is required";
        ret = false;
    }
}

```

```

    }
    else
    {
        document.getElementById("lblLastName").textContent = "";
    }

    if (document.getElementById("txtEmail").value == "")
    {
        document.getElementById("lblEmail").textContent = "Email is required";
        ret = false;
    }
    else
    {
        document.getElementById("lblEmail").textContent = "";
    }

    return ret;
}

```

Example 2 : The following JavaScript function **ddlGenderSelectionChanged()** works in Chrome and Firefox but not in Internet Explorer.

```

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title></title>
    <script type="text/javascript" language="javascript">
        function ddlGenderSelectionChanged() {
            alert('You selected ' + ddlGender.value);
        }
    </script>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <select id="ddlGender" onchange="ddlGenderSelectionChanged()">
                <option>Male</option>
                <option>Female</option>
            </select>
        </div>
    </form>
</body>
</html>

```

For the **JavaScript** function to work in all browsers (Chrome, Firefox & Internet Explorer) it

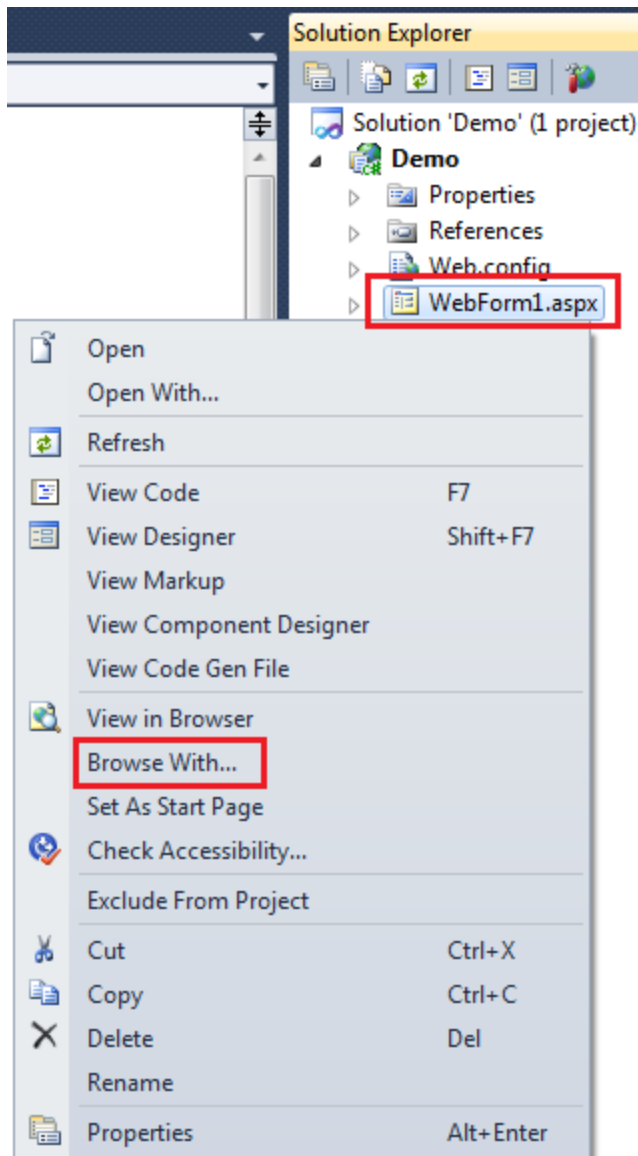
needs to be modified as shown below.

```
<script type="text/javascript" language="javascript">
  function ddlGenderSelectionChanged()
  {
    alert('You selected ' + document.getElementById('ddlGender').value);
  }
</script>
```

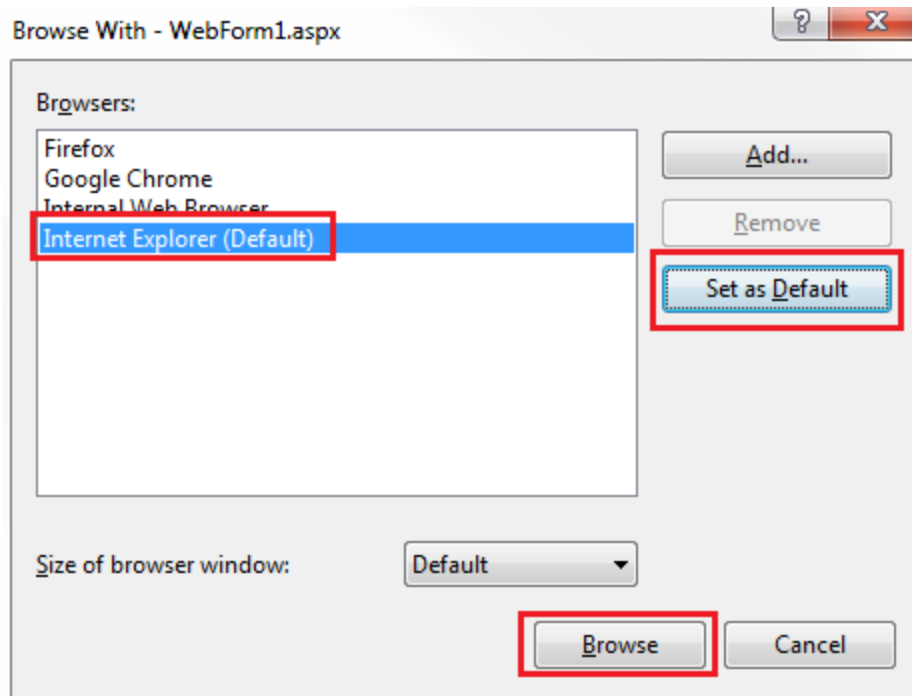
Debugging JavaScript in Visual Studio

Here are the steps

1. In Visual Studio, right click on a web form and select "**Browse With...**" option from the context menu



2. In the "**Browse With**" window,
 - a) select "**Internet Explorer**"
 - b) Click "**Set as Default**" button
 - c) Finally click "**Browse**" button



3. In Visual Studio, throw a break point on the JavaScript function that you want to debug

4. Run the application in Debug mode

At this point, in the internet explorer perform the action (i.e clicking on button or changing a selection in the DropDownList) that calls the JavaScript function. The JavaScript execution should stop at the breakpoint and we should be able to step over and step into code using F10 and F11 keys respectively.

At this point we should also be able to use **watch**, **quick watch** and **immediate windows** in visual studio.

This approach will not work in visual studio 2008, if you are using any other browser apart from Internet Explorer.

Tools for learning JavaScript

To learn JavaScript all you need is a text editor like NOTEPAD and a browser. You can use any browser Chrome, Internet Explorer or Firefox.

Here are stepd to use notepad to write and test your first JavaScript program

1. Open a notepad.

2. Copy and paste the following HTML and JavaScript in the notepad.

```
<html>

<head>

</head>

<body>

  <h1>JavaScript Training</h1>

  <script type="text/javascript">

    alert("Welcome to JavaScript Tutorial");

  </script>

</body>

</html>
```

Please Note : JavaScript should be inside the <script> element, The type attribute specifies that the script is JavaScript. If the type attribute is absent, then the script is treated as JavaScript. With in the script tag we have a single line of JavaScript, that displays an alert box with the message "Welcome to JavaScript Tutorial". alert() is a JavaScript function. The JavaScript statement ends with a semicolon(;). It is not mandatory for a JavaScript statement to end with a semicolon, but it's a good practice to avoid ambiguity. The above JavaScript code will work even without the semicolon.

3. Save the notepad with **.htm** or **.html** extension.

4. Open your favorite browser. From the File menu, select Open and navigate to the location where you have saved the notepad and open it. An alert box should pop up displaying the message "Welcome to JavaScript Tutorial".

Websites that help write and test JavaScript : One of the websites I have found very useful to write and test JavaScript is <http://jsbin.com>. This website also provides intelligence and autocomplete features. Type the partial name of the variable or the function and press TAB key.

Advantage of using Visual Studio to learn JavaScript

Advantage of using Visual Studio over notepad to learn JavaScript is that it provides intelligence and autocomplete features which can reduce typos and increase developer productivity. It also provides rich JavaScript debugging support.

Inline Vs External JavaScript

[Project Name: 03_InlineVsExternalJavaScript]

JavaScript can be stored either inline on the page or in an external .js file. Let's look at an example of both the approaches.

Inline JavaScript example : In this example, **IsEven()** JavaScript function is present inline on the page.

```
<html>
<head>
  <script type="text/javascript">
    function IsEven() {
      var number = document.getElementById("TextBox1").value;
      if (number % 2 == 0) {
        alert(number + " is even number");
      }
      else {
        alert(number + " is odd number");
      }
    }
  </script>
</head>
<body>
  <form id="form1" runat="server">
    Number :
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    <input type="button" value="Check Number" onclick="IsEven()" />
  </form>
</body>
</html>
```

External JavaScript example : Steps to store JavaScript in an external .js file

1. In Visual Studio, right click on the project name in Solution Explorer and select add select Add => New Item

2. From the "Add New Item" dialog box select "JScript File". Name the file "ExternalJavaScript.js" and click **Add**.

3. Copy and paste the following JavaScript function in the "ExternalJavaScript.js" file

```
function IsEven()
{
```

```

var number = document.getElementById("TextBox1").value;
if (number % 2 == 0)
{
    alert(number + " is even number");
}
else
{
    alert(number + " is odd number");
}
}

```

4. On your webform in the head section include a reference to the external JavaScript file using script element as shown below

```

<script type="text/javascript" src="ExternalJavaScript.js"></script>

```

5. At this point the HTML on your webform should be as shown below.

```

<html>
<head>
    <script type="text/javascript" src="ExternalJavaScript.js"></script>
</head>
<body>
    <form id="form1" runat="server">
        Number :
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <input type="button" value="Check Number" onclick="IsEven()" />
    </form>
</body>
</html>

```

Advantages of external JavaScript over inline JavaScript : Here are some of the general advantages of external JavaScript over inline JavaScript

Maintainability : JavaScript in external files can be referenced on multiple pages without having to duplicate the code inline on every page. If something has to change, you only have one place to change. So external JavaScript code can be reused and maintenance will be much easier.

Separation of Concerns : Storing JavaScript in a separate external .js file adheres to Separation of concerns design principle. In general it is a good practice to separate HTML, CSS and JavaScript as it makes it easier working with them. Also allows multiple developers to work simultaneously.

Performance : An external JavaScript file can be cached by the browser, where as an inline JavaScript on the page is loaded every time the page loads.

Place for Script Tag

[Project Name: 04_PlacesOfJavascript]

where should the script tag be placed in the html. Should it be placed in the body or head section of the page. In general the script tag can be placed either in the head or body section.

Let's look at a few examples :

Example 1 : Script tag in the head section

```
<html>
<head>
  <script type="text/javascript">
    alert("Welcome to JavaScript Tutorial");
  </script>
</head>
<body>
  <form id="form1" runat="server">
  </form>
</body>
</html>
```

Example 2 : Script tag in the body section

```
<html>
<head>
</head>
<body>
  <form id="form1" runat="server">
  </form>
  <script type="text/javascript">
    alert("Welcome to JavaScript Tutorial");
  </script>
</body>
</html>
```

In **Example 1** we placed the script tag in the head section and in **Example 2**, we placed it in body section. In both the cases JavaScript works as expected.

Example 3 : Sets the background color of the TextBox to red.

```

<html>
<head>
</head>
<body>
  <form id="form1" runat="server">
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
  </form>
  <script type="text/javascript">
    document.getElementById("TextBox1").style.backgroundColor = "red";
  </script>
</body>
</html>

```

Example 4 : This is same as **Example 3**, except we moved the script tag to the head section. In this case the script will not work as expected. Depending on the browser you are using you get one of the following JavaScript error.

Firefox - TypeError: document.getElementById(...) is null

Chrome - Uncaught TypeError: Cannot read property 'style' of null

IE - Unable to get property 'style' of undefined or null reference

To see these JavaScript errors press F12 which launches developer tools. F12 works for all the 3 browsers.

```

<html>
<head>
  <script type="text/javascript">
    document.getElementById("TextBox1").style.backgroundColor = "red";
  </script>
</head>
<body>
  <form id="form1" runat="server">
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
  </form>
</body>
</html>

```

Why did the JavaScript did not work in this case

JavaScript code is present before the textbox control. By the time the JavaScript code is executed, the textbox is still not loaded into browser DOM (Document Object Model). This is the reason JavaScript can't find the textbox and throws a NULL reference error.

In general it is a good practice to store JavaScript in an external .js file. So, if the JavaScript is present in an external file and if you are referencing it on a page using <script> element, where should such <script> element be placed.

To answer this question, first let's understand what happens when a browser starts loading a

web page.

1. The browser starts parsing the HTML

2. When the parser encounters a `<script>` tag that references an external JavaScript file. The parser stops parsing the HTML and the browser makes a request to download the script file. Until the download is complete, the parser is blocked from parsing the rest of the HTML on the page.

3. When the download is complete, that's when the parser will resume to parse the rest of the HTML.

This means the page loading is stopped until all the scripts are loaded which affects user experience.

In general, the suggested good practice is to place the `<script>` tag just before the closing `<body>` tag, so it doesn't block the HTML parser. However, modern browsers support `async` and `defer` attributes on scripts. These attributes tell the browser it's safe to continue parsing while the scripts are being downloaded. But even with these attributes, from a performance standpoint it is still better to place `<script>` tag just before closing `<body>` tag.

According to HTTP/1.1 specification browsers download no more than two components in parallel. So, if the page has several images to download and if you place `<script>` tags at the top of the page, the script files start to download first which blocks the images download which adds to the total page load time.

JavaScript Basics

[Project Name: 05_DataTypesInJavaScript]

Is JavaScript case sensitive

Yes, JavaScript is case sensitive programming language. Variable names, keywords, methods, object properties and event handlers all are case sensitive.

Example 1 : `alert()` function name should be all small letters

```
<script>  
  alert("JavaScripts Basics Tutorial");  
</script>
```

Example 2 : `Alert()` is not same as `alert()`. Throws **Alert is not defined** error. To see the error press F12 key.

```
<script>
  Alert("JavaScripts Basics Tutorial");
</script>
```

Comments in JavaScript :

There are 2 types of comments in JavaScript.

1) Single Line Comment

Example :

```
<script>
  // This is a sinle line comment
</script>
```

2) Multi Line Comment

Example:

```
<script>
  /* This is a
  multi line
  comment */
</script>
```

Data types in JavaScript

The following are the different data types in JavaScript

Numbers - 5, 5.234

Boolean - true / false

String - "MyString", 'MyString'

To create a variable in JavaScript use var keyword. Variable names are case sensitive.

In c# to create an integer variable we use int keyword

int X = 10;

to create a string variable we use string keyword

string str = "Hello"

With JavaScript we always use var keyword to create any type of variable. Based on the value assigned the type of the variable is inferred.

```
var a = 10;
var b = "MyString";
```

In C#, you cannot assign a string value to an integer variable

```
int X = 10;
```

```
X = "Hello"; // Compiler error
```

JavaScript is a dynamically typed language. This means JavaScript data types are converted automatically as needed during script execution. Notice that, in myVariable we are first storing a number and then a string later.

```
<script>
  var myVariable = 100;
  alert(myVariable);
  myVariable = "Assigning a string value";
  alert(myVariable);
</script>
```

When a + operator is used with 2 numbers, JavaScripts adds those numbers.

```
<script>
  var a = 10;
  var b = 20;
  var c = a + b;
  alert(c);
</script>
```

Output : 30

When a + operator is used with 2 strings, JavaScript concatenates those 2 strings

```
<script>
  var a = "Hello "
  var b = "JavaScript";
  var c = a + b;
  alert(c);
</script>
```

Output : Hello JavaScript

When a + operator is used with a string and a number, JavaScript converts the numeric value to a string and performs concatenation.

```
<script>
  var a = "Number is : "
  var b = 10;
  var c = a + b;
```

```
    alert(c);  
</script>
```

Output : Number is 10

```
<script>  
    var a = "50"  
    var b = 10;  
    var c = a + b;  
    alert(c);  
</script>
```

Output : 5010

But if you use a minus operator, numeric value is not converted to string

```
<script>  
    var a = "50"  
    var b = 10;  
    var c = a - b;  
    alert(c);  
</script>
```

Output : 40

Converting String to a Number

[Project Name: 06_ConversionsFromString]

There are various methods that are available in JavaScript to convert strings to numbers.

Let's design the web form as shown below.

First Number	<input type="text"/>
Second Number	<input type="text"/>
Result	<input type="text"/>
	<input type="button" value="Add"/>

This form allows the user to enter 2 numbers and add them. Along the way we will learn the use of the following functions

1. **parseInt()**
2. **parseFloat()**

3. isNaN()

Here is the HTML for the web form

```
<table style="border:1px solid black; font-family:Arial">
  <tr>
    <td>First Number</td>
    <td><asp:TextBox ID="txtFirstNumber" runat="server"></asp:TextBox></td>
  </tr>
  <tr>
    <td>Second Number</td>
    <td><asp:TextBox ID="txtSecondNumber" runat="server"></asp:TextBox></td>
  </tr>
  <tr>
    <td>Result</td>
    <td><asp:TextBox ID="txtResult" runat="server"></asp:TextBox></td>
  </tr>
  <tr>
    <td>
      <td>
      </td>
      <td>
        <input type="button" value="Add" id="btnAdd"/>
      </td>
    </td>
  </tr>
</table>
```

Include the following JavaScript in the head section of the web form.

```
<script type="text/javascript">
  function addNumbers()
  {
    var firstNumber = document.getElementById("txtFirstNumber").value;
    var secondNumber = document.getElementById("txtSecondNumber").value;
    document.getElementById("txtResult").value = firstNumber + secondNumber;
  }
</script>
```

Set the **onclick** attribute of the Add button to call the **addNumbers()** function. The HTML for the button should be as shown below.

```
<input type="button" value="Add" id="btnAdd" onclick="addNumbers()" />
```

Run the application and enter 20 as the first number and 10 as the second number. Click Add button. Notice that JavaScript concatenates the numbers instead of adding them. This is because the value property of the textbox is returning the number in a string format.

First Number	<input type="text" value="20"/>
Second Number	<input type="text" value="10"/>
Result	<input type="text" value="2010"/>
	<input type="button" value="Add"/>

So we need to **explicitly do the conversion**. This is when `parseInt()` function is useful. Modify the `addNumbers()` JavaScript function as shown below.

```
function addNumbers()
{
    var firstNumber = parseInt(document.getElementById("txtFirstNumber").value);
    var secondNumber = parseInt(document.getElementById("txtSecondNumber").value);
    document.getElementById("txtResult").value = firstNumber + secondNumber;
}
```

Run the application and test. Notice that we now get 30 as expected.

First Number	<input type="text" value="20"/>
Second Number	<input type="text" value="10"/>
Result	<input type="text" value="30"/>
	<input type="button" value="Add"/>

Let's do another test. Enter 20.5 as the first number and 10.3 as the second number. Click the Add button. Notice that the decimal part is ignored.

First Number	<input type="text" value="20.5"/>
Second Number	<input type="text" value="10.3"/>
Result	<input type="text" value="30"/>
	<input type="button" value="Add"/>

To retain the decimal places, use **`parseFloat()` function**.

```
function addNumbers()
{
    var firstNumber = parseFloat(document.getElementById("txtFirstNumber").value);
    var secondNumber = parseFloat(document.getElementById("txtSecondNumber").value);
    document.getElementById("txtResult").value = firstNumber + secondNumber;
}
```


First Number	<input type="text" value="20.5"/>
Second Number	<input type="text" value="10.3"/>
Result	<input type="text" value="30.8"/>
<input type="button" value="Add"/>	

If you leave first number and second number textboxes blank or if you enter text instead of a number, and when you click the Add button, **NaN** is displayed in result textbox.

NaN in JavaScript stands for Not-a-Number. In JavaScript we have isNaN() function which determines whether a value is an illegal number. This function returns true if the value is not a number, and false if not.

Modify the addNumbers() JavaScript function as shown below.

```
function addNumbers()
{
    var firstNumber = parseFloat(document.getElementById("txtFirstNumber").value);
    if (isNaN(firstNumber))
    {
        alert("Please enter a valid number in the first number textbox");
        return;
    }
    var secondNumber = parseFloat(document.getElementById("txtSecondNumber").value);
    if (isNaN(secondNumber))
    {
        alert("Please enter a valid number in the second number textbox");
        return;
    }
    document.getElementById("txtResult").value = firstNumber + secondNumber;
}
```

Now, when you leave first number and second number textboxes blank or if you enter text instead of a number, and when you click the Add button, you get relevant validation error messages as expected.

Let's make the validation error message a little more relevant:

If the first number and second number textboxes are left blank, then we want to display the following validation messages

- a) First Number is required
- b) Second Number is required

If you enter text instead of number

- a) Please enter a valid number in the first number textbox

b) Please enter a valid number in the second number textbox

To achieve this modify addNumbers() function as shown below.

```
function addNumbers()
{
    var firstNumber = document.getElementById("txtFirstNumber").value;
    var secondNumber = document.getElementById("txtSecondNumber").value;

    if (firstNumber == "")
    {
        alert("First Number is required");
        return;
    }

    firstNumber = parseFloat(firstNumber);
    if (isNaN(firstNumber))
    {
        alert("Please enter a valid number in the first number textbox");
        return;
    }

    if (secondNumber == "")
    {
        alert("Second Number is required");
        return;
    }

    secondNumber = parseFloat(secondNumber)
    if (isNaN(secondNumber))
    {
        alert("Please enter a valid number in the second number textbox");
        return;
    }

    document.getElementById("txtResult").value = firstNumber + secondNumber;
}
```

Strings in JavaScript

[Project Name: 07_Strings]

There are different functions that are available to manipulate strings in JavaScript.

A string is any text inside quotes. You can use either single or double quotes.

```
var string1 = "string in double quotes"
```

```
var string2 = 'string in single quotes'
```

Concatenating strings : There are 2 options to concatenate strings in JavaScript. You could either use + operator or concat() method

Example : Concatenating strings using + operator

```
var string1 = "Hello"  
var string2 = "JavaScript"  
var result = string1 + " " + string2;  
alert(result);
```

Output : Hello JavaScript

Example : Concatenating strings using concat() method

```
var string1 = "Hello"  
var string2 = "JavaScript"  
var result = string1.concat(" ", string2);  
alert(result);
```

Output : Hello JavaScript

If you want single quotes inside a string, there are 2 options

Option 1 : Place your entire string in double quotes, and use single quotes inside the string wherever you need them.

Example :

```
var myString = "Welcome to 'JavaScript' Training";  
alert(myString);
```

Output : Welcome to 'JavaScript' Training

Option 2 : If you prefer to place your entire string in single quotes, then use escape sequence character \ with a single quote inside the string.

Example :

```
var myString = 'Welcome to \'JavaScript\' Training';  
alert(myString);
```

Output : Welcome to 'JavaScript' Training

Please Note : You can use the above 2 ways if you need double quotes inside a string

Converting a string to uppercase : Use toUpperCase() method

Example :

```
var upperCaseString = "JavaScript";  
alert(upperCaseString.toUpperCase());
```

Output : JAVASCRIPT

Converting a string to lowercase : Use toLowerCase() method

Example :

```
var lowerCaseString = "JavaScript";  
alert(lowerCaseString.toLowerCase());
```

Output : javascript

Length of string javascript : Use length property

Example : alert("JavaScript".length);

Output : 10

Example :

```
var myString = "Hello JavaScript";  
alert(myString.length);
```

Output : 16

Remove whitespace from both ends of a string : Use trim() method

Example :

```
var string1 = " AB ";  
var string2 = " CD ";  
var result = string1.trim() + string2.trim();  
alert(result);
```

Output : ABCD

Replacing strings in JavaScript : Use replace() method. This method searches a given string for a specified value or a regular expression, replaces the specified values with the replacement values and returns a new string. This method does not change the original string.

Example : Replaces JavaScript with World

```
var myString = "Hello JavaScript";  
var result = myString.replace("JavaScript", "World");  
alert(result);
```

Output : Hello World

Example : **Perform a case-sensitive global replacement.** In this example, we are using a regular expression between the 2 forward slashes(/). The letter g after the forward slash specifies a global replacement. The match here is case sensitive. This means Blue(with capital B) is not replaced with green.

```
var myString = "A Blue bottle with a blue liquid is on a blue table";  
var result = myString.replace(/blue/g, "green");  
alert(result);
```

Output : A Blue bottle with a green liquid is on a green table

Example : **Perform a case-insensitive global replacement.** The letters gi after the forward slash indicates to do global case-insensitive replacement. Notice that the word Blue(with capital B) is also replaced with green.

```
var myString = "A Blue bottle with a blue liquid is on a blue table";  
var result = myString.replace(/blue/gi, "green");  
alert(result);
```

Output : A green bottle with a green liquid is on a green table

Substrings in JavaScript

[Project Name: 08_SubstringsInJavaScript]

The following are the 3 methods in JavaScript that can be used to **retrieve a substring from a given string**.

substring()
substr()
slice()

substring() method : This method has 2 parameters start and end. start parameter is required and specifies the position where to start the extraction. end parameter is optional and specifies the position where the extraction should end. The character at the end position is not included in the substring. If the end parameter is not specified, all the characters from the start position till the end of the string are extracted. If the value of start parameter is greater than the value of the end parameter, this method will swap the two arguments. This means start will be used as end and end will be used as start.

Extract the first 10 characters

```
var str = "JavaScript Tutorial";  
var result = str.substring(0, 10);  
alert(result);
```

Output : JavaScript

If the value of start parameter is greater than the value of the end parameter, then start will be used as end and end will be used as start

```
var str = "JavaScript Tutorial";  
var result = str.substring(10, 0);  
alert(result);
```

Output : JavaScript

substr() method : This method has 2 parameters start and count. start parameter is required and specifies the position where to start the extraction. count parameter is optional and specifies the number of characters to extract. If the count parameter is not specified, all the characters from the start position till the end of the string are extracted. If count is 0 or negative, an empty string is returned.

Extract the first 10 characters

```
var str = "JavaScript Tutorial";  
var result = str.substr(0, 10);  
alert(result);
```

Output : JavaScript

If the count parameter is not specified, all the characters from the start position till the end of the string are extracted

```
var str = "JavaScript Tutorial";  
var result = str.substr(11);  
alert(result);
```

Output : Tutorial

slice() method : This method has 2 parameters start and end. start parameter is required and specifies the position where to start the extraction. end parameter is optional and specifies the position where the extraction should end. The character at the end position is not included in the substring. If the end parameter is not specified, all the characters from the start position till the end of the string are extracted.

Extract the first 10 characters

```
var str = "JavaScript Tutorial";  
var result = str.slice(0, 10);  
alert(result);
```

Output : JavaScript

If the end parameter is not specified, all the characters from the start position till the end of the string are extracted

```
var str = "JavaScript Tutorial";  
var result = str.slice(11);  
alert(result);
```

Output : Tutorial

What is the difference between substr and substring methods?

The difference is in the second parameter. The second parameter of substring() method specifies the index position where the extraction should stop. The character at the end position is not included in the substring. The second parameter of substr() method specifies the number

of characters to return.

Another difference is substr() method does not work in IE8 and earlier versions.

What is the difference between slice and substring?

If start parameter is greater than stop parameter, then substring will swap those 2 parameters, where as slice will not swap.

Another method that is very useful when extracting a substring is indexOf() method. This method returns the position of the first occurrence of a specified value in a string. If the specified value is not present then -1 is returned.

Example : Retrieve the index position of @ character in the email

```
var str = "test@test.com";  
var result = str.indexOf("@");  
alert(result);
```

Output : 4

Substring Example

[Project Name: 08_SubstringsInJavaScript]

A simple real time example of where we can use indexOf(), lastIndexOf() and substring() methods.

Design the web form as shown below : This form allows the use to enter an email address and when they click "Get email & domain parts" button, we should retrieve email part and domain part from the email address and display in respective textboxes.

Email Address	<input type="text" value="venkat.k@pragimtech.com"/>
Email Part	<input type="text" value="venkat.k"/>
Domian Part	<input type="text" value="pragimtech.com"/>
<input type="button" value="Get email & domain parts"/>	

Here is the HTML

```
<table style="border:1px solid black; font-family:Arial">
```



```

<tr>
  <td>
    Email Address
  </td>
  <td>
    <asp:TextBox ID="txtEmailAddress" runat="server" Width="250px">
    </asp:TextBox>
  </td>
</tr>
<tr>
  <td>
    Email Part
  </td>
  <td>
    <asp:TextBox ID="txtEmailPart" runat="server" Width="250px"></asp:TextBox>
  </td>
</tr>
<tr>
  <td>
    Domian Part
  </td>
  <td>
    <asp:TextBox ID="txtDomainPart" runat="server" Width="250px"></asp:TextBox>
  </td>
</tr>
<tr>
  <td></td>
  <td>
    <input type="button" value="Get email & domain parts" style="width:250px"/>
  </td>
</tr>
</table>

```

In the head section of the webform, include the following script section

```

function getEmailandDomainParts()
{
  var emailAddress = document.getElementById("txtEmailAddress").value;

  var emailPart = emailAddress.substring(0, emailAddress.indexOf("@"));
  var domainPart = emailAddress.substring(emailAddress.indexOf("@") + 1);

  document.getElementById("txtEmailPart").value = emailPart;
  document.getElementById("txtDomainPart").value = domainPart;
}

```

```
}
```

Finally set the onclick attribute of the button to call the JavaScript function

```
<input type="button" value="Get email & domain parts" style="width:250px"
      onclick="getEmailandDomainParts()"/>
```

lastIndexOf() method returns the position of the last occurrence of a specified value in a string. Since its job is to return the last index of the specified value, this method searches the given string from the end to the beginning and returns the index of the first match it finds. This method returns -1 if the specified value is not present in the given string.

Example : Retrieve the last index position of dot (.) in the given string

```
var url = "http://www.csharp-video-tutorials.blogspot.com";
alert(url.lastIndexOf("."));
```

Output : 42

Simple real time **example where lastIndexOf and substring methods can be used**

Design the web form as shown below : This form allows the use to enter a web site URL and when they click "**Get top level domain**" button, we should retrieve domain name and display in "Top level domain" textbox.

Website URL	<input type="text" value="http://www.csharp-video-tutorials.blogspot.com"/>
Top level domain	<input type="text" value=".com"/>
<input type="button" value="Get top level domain"/>	

Here is the HTML

```
<table style="border: 1px solid black; font-family: Arial">
  <tr>
    <td>
      Website URL
    </td>
    <td>
      <asp:TextBox ID="txtURL" runat="server" Width="300px"></asp:TextBox>
    </td>
  </tr>
  <tr>
    <td>
      Top level domain
    </td>
    <td>
      <asp:TextBox ID="txtDomain" runat="server" Width="100px"></asp:TextBox>
    </td>
  </tr>
  <tr>
    <td colspan="2">
      <input type="button" value="Get top level domain" />
    </td>
  </tr>
</table>
```

```

        Top level domian
    </td>
    <td>
        <asp:TextBox ID="txtDomian" runat="server" Width="300px"></asp:TextBox>
    </td>
</tr>
<tr>
    <td>
    </td>
    <td>
        <input type="button" value="Get top level domain" style="width: 300px" />
    </td>
</tr>
</table>

```

In the head section of the webform, include the following script section

```

<script type="text/javascript">
    function getDomainName()
    {
        var url = document.getElementById("txtURL").value;
        var domainName = url.substr(url.lastIndexOf("."));
        document.getElementById("txtDomian").value = domainName;
    }
</script>

```

Finally set the onclick attribute of the button to call the JavaScript function

```

<input type="button" value="Get top level domain" style="width: 300px"
    onclick="getDomainName()" />

```

Conditional Statements in JavaScript

[Project Name: 09_ConditionalStatements]

JavaScript code is executed in a linear fashion from the first line to the last line. If for some reason you want to interrupt this flow and execute certain statements, only, if certain condition is met, then we use conditional statements.

JavaScript has the following conditional statements

if

if else

if else if else

switch

ternary operator - shortcut for an if...else statement

If example

```
var userInput = Number(prompt("Please enter a number", ""));
if (userInput == 1)
{
    alert("You number is One");
}
if (userInput == 2)
{
    alert("You number is Two");
}
if (userInput == 3)
{
    alert("Your number is Three");
}
if (userInput != 1 && userInput != 2 && userInput != 3)
{
    alert("Your number is not between 1 and 3");
}
```

If else if example

```
var userInput = Number(prompt("Please enter a number", ""));
if (userInput == 1)
{
    alert("You number is One");
}
else if (userInput == 2)
{
    alert("You number is Two");
}
else if (userInput == 3)
{
    alert("Your number is Three");
}
else
{
    alert("Your number is not between 1 and 3");
}
```

Switch Statement

[Project Name: 09_ConditionalStatements]

When should we use switch statement

To improve the readability of a program multiple if else if statements can be replaced with a switch statement.

Notice that in the following code we have several if else if statements

```
var userInput = Number(prompt("Please enter a number", ""));
if (userInput == 1)
{
    alert("You number is One");
}
else if (userInput == 2)
{
    alert("You number is Two");
}
else if (userInput == 3)
{
    alert("Your number is Three");
}
else
{
    alert("Your number is not between 1 and 3");
}
```

The above code can be rewritten using a switch statement and it greatly improves the readability

```
var userInput = Number(prompt("Please enter a number", ""));
switch (userInput)
{
    case 1:
        alert("You number is One");
        break;
    case 2:
        alert("You number is Two");
        break;
    case 3:
        alert("You number is Three");
        break;
    default:
        alert("You number is not between 1 and 3");
        break;
}
```

In general we need to have break statement after each case statement to ensure the program breaks out of the switch statement after executing statements present in a specific case.

What happens if there is no break in a switch statement

If there is no break statement the execution falls automatically to next case until a break statement is encountered or end of the program is reached.

In the example below, since we don't have a break statement for case 1, when we enter 1 as the number we would get 2 alerts. First alert from case 1 and the second alert from case 2.

```
var userInput = Number(prompt("Please enter a number", ""));
switch (userInput)
{
    case 1:
        alert("You number is One");
    case 2:
        alert("You number is Two");
        break;
    case 3:
        alert("You number is Three");
        break;
    default:
        alert("You number is not between 1 and 3");
        break;
}
```

When would you combine multiple case statements together

If you want the same piece of code to be executed for multiple cases you can combine them together as shown below. Case statement with no code in-between creates a single case for multiple values. A case without any code will automatically fall through to the next case.

In this example case 1 and case 2 will fall through and execute code for case 3

```
var userInput = Number(prompt("Please enter a number", ""));
switch (userInput)
{
    case 1:
    case 2:
    case 3:
        alert("You number is " + userInput);
        break;
    default:
        alert("You number is not between 1 and 3");
        break;
}
```

Ternary Operator

Ternary operator can be used as a shortcut for if...else... statement.

Here is the syntax for the ternary operator

Boolean Expression ? Statements to execute if true : Statements to execute if false

The following example checks if the number is even or odd. We are using if...else... statement.

```
var userInput = Number(prompt("Please enter a number"));
var message = "";
if (userInput % 2 == 0)
{
    message = "Your number is even";
}
else
{
    message = "Your number is odd";
}
```

```
alert(message);
```

Ternary operator example : In the above example if...else... statement can be replaced with the ternary operator as shown below.

```
var userInput = Number(prompt("Please enter a number"));
var message = userInput % 2 == 0 ? "Your number is even"
                                : "Your number is odd";

alert(message);
```

Can we have multiple statements per case with a ternary operator

Yes, we can put the multiple statements in a bracket and separate them with a comma.

Example :

```
var userInput = Number(prompt("Please enter a number"));
userInput % 2 == 0
    ? (alert("Your number is even"), alert("Your number is " + userInput))
```

```
: (alert("Your number is odd"), alert("Your number is " + userInput));
```

Multiple if..else if... statements can be replaced with ternary operator.

The following example displays the month name based on the month number that is provided using if...else if... statement

```
var userInput = Number(prompt("Please enter a month number - 1, 2 or 3"));
var monthName = "";
```

```
if (userInput == 1)
{
    monthName = "January";
}
else if (userInput == 2)
{
    monthName = "February";
}
else if (userInput == 3)
{
    monthName = "March";
}
else
{
    monthName = "Unknown month number"
}
```

```
alert(monthName);
```

Ternary operator with multiple conditions example : In the following example, if...else if... statements are replaced with ternary operator

```
var userInput = Number(prompt("Please enter a month number - 1, 2 or 3"));
var monthName = userInput == 1 ? "January" : userInput == 2
    ? "February" : userInput == 3
    ? "March" : "Unknown month number";

alert(monthName);
```


Loops in JavaScript

[Project Name: 10_LoopingStatements]

While Loop

Here are the basic **loops in JavaScript**

while

do...while

for

The following example, prints all even numbers from 0 till the target number that the end user has provided.

```
var targetNumber = Number(prompt("Please enter your target number", ""));
var start = 0;
while (start <= targetNumber)
{
    document.write(start + "<br/>");
    start = start + 2;
}
```

How does the while loop work

1. While loop checks the condition first
2. If the condition is true, statements with in the loop are executed
3. This process is repeated as long as the condition evaluates to true.

Failure to update the variable that participates in the condition will create a never ending or infinite loop. In the example below, notice that we have commented the line that updates start variable. As a result of this start will always be ZERO and will always be less than the target number. This mean the condition will never become false, and we have an infinite while loop.

JavaScript infinite while loop example

```
var targetNumber = Number(prompt("Please enter your target number", ""));
var start = 0;
while (start <= targetNumber)
{
    document.write(start + "<br/>");
    // start = start + 2;
}
```

Let us now understand the **use of break statement in a while loop** with an example. We want a while loop that would to the following

1. If the target number that is provided by the end user is less than 100, then print all even numbers from ZERO until the target number
2. If the target number that is provided by the end user is greater than 100, then print all even

numbers from ZERO until 100.

So, this means the loop should break once it has printed even numbers until 100.

JavaScript while loop break out example

```
var targetNumber = Number(prompt("Please enter your target number", ""));
var start = 0;
while (start <= targetNumber)
{
    document.write(start + "<br/>");
    start = start + 2;

    if (start > 100) {
        break;
    }
}
```

What is the use of break statement

If break statement is used in a switch statement, it causes the execution to break out of that switch statement. In a similar fashion, if used in a loop, the loop ends.

JavaScript while loop continue example

continue statement tells the JavaScript interpreter to skip remaining code that is present after this statement and start the next iteration of the loop. Let us understand this with an example.

Example: The following example prints all odd numbers between 1 and 10.

```
var start = 0;
while (start < 10)
{
    start = start + 1;

    if (start % 2 == 0)
    {
        continue;
    }

    document.write(start + "<br/>");
}
```

Do While Loop

while loop :

1. while loop checks the condition first
2. If the condition is true, statements within the loop are executed
3. This process is repeated as long as the condition evaluates to true.

do while loop :

1. do while loop checks its condition at the end of the loop
2. This means the do...while... loop is guaranteed to execute at least one time.
3. In general, do...while... loops are used to present a menu to the user

What is the difference between while and do while in JavaScript

1. while loop checks the condition at the beginning, whereas do...while... loop checks the condition at the end of the loop
2. do...while... loop is guaranteed to execute at least once, whereas while loop is not.

do while loop example

```
var userChoice = "";
do
{
    var targetNumber = Number(prompt("Please enter your target number", ""));
    var start = 0;
    while (start <= targetNumber)
    {
        document.write(start + "<br/>");
        start = start + 2;
    }
    do
    {
        userChoice = prompt("Do you want to continue - Yes or No").toUpperCase();
        if (userChoice != "YES" && userChoice != "NO") {
            alert("Invalid choice. Please say, Yes or No");
        }
    } while (userChoice != "YES" && userChoice != "NO");
} while (userChoice == "YES");
```

For Loop

To understand for loop, let us print even numbers starting from ZERO till a number provided by the user at runtime.

First, let us do this using a while loop. The following while loop example prints all even numbers starting from ZERO till a number provided by the user at runtime. For example if the end user

provides 10 at runtime, even numbers starting from ZERO till 10 are printed.

```
var targetNumber = Number(prompt("Please enter your target number", ""));
var start = 0;
while (start <= targetNumber)
{
    document.write(start + "<br/>");
    start = start + 2;
}
```

With while loop

1. The initialization of the variable is done at one place
2. Boolean condition is checked at another place
3. The variable participating in the Boolean expression is updated at another place

For loop example : With for loop all three can be done in one place. To make this clear, look at the syntax of the for loop below.

```
for(initialization; Boolean Condition; update variable)
{
    statements;
}
```

Now, let us rewrite the above example using a for loop, instead of while loop

```
var targetNumber = Number(prompt("Please enter your target number", ""));
for (var start = 0; start <= targetNumber; start = start + 2)
{
    document.write(start + "<br/>");
}
```

Notice that all the 3 things, i.e

1. Variable initialization
2. Boolean condition check and
3. Updating the variable participating in the Boolean expression are all done at one place in the for loop. All these 3 things are optional in a for loop.

Example : Variable initialization is optional in a for loop. Notice that we have moved variable initialization (var start = 0;) outside of the for loop head.

```
var targetNumber = Number(prompt("Please enter your target number", ""));
var start = 0;
for (; start <= targetNumber; start = start + 2)
{
    document.write(start + "<br/>");
}
```

Example : Just like variable initialization, condition check is also optional in a for loop. Notice that we have moved condition check (start > targetNumber) from the head of the for loop, into its body and we used a break statement to end the loop, otherwise this could become an infinite for loop.

```
var targetNumber = Number(prompt("Please enter your target number", ""));
var start = 0
for (; ; start = start + 2)
{
    if (start > targetNumber)
    {
        break;
    }
    document.write(start + "<br/>");
}
```

Example : In this example we have moved the expression that updates the variable participating in the Boolean expression (start = start + 2) from the head of the for loop, into its body.

```
var targetNumber = Number(prompt("Please enter your target number", ""));
var start = 0
for (; ; )
{
    if (start > targetNumber)
    {
        break;
    }
    document.write(start + "<br/>");
    start = start + 2;
}
```

Arrays In JavaScript

Arrays are collections and are ZERO indexed. This means the first element is at index ZERO and the last element is at index arrayObject.length - 1. length property of the array object returns the size of the array.

The following JavaScript code creates an empty array. length property returns 0 in this case.

```
var emptyArray = [];
alert(emptyArray.length);
```

Output : 0

Another way to create an array is by using the Array constructor as shown below. In this example we are setting the length of the array to 10.

```
var myArray = new Array(10);  
alert(myArray.length);
```

Output : 10

Retrieving first and last elements from the array using the array index

```
var myArray = [10, 20, 30];  
document.write("First element = " + myArray[0] + "<br/>");  
document.write("Last element = " + myArray[myArray.length - 1] + "<br/>");
```

Output :

First element = 10

Last element = 30

Populating an array in JavaScript : There are several ways to populate an array in JavaScript. Let's look at those different options now.

Declaring an array first and then populating using the array index

```
var myArray = [];
```

```
myArray[0] = 10;  
myArray[1] = 20;  
myArray[2] = 30;
```

```
alert(myArray);
```

Output : 10, 20, 30

Declaring and populating the array at the same time

```
var myArray = [10, 20, 30];  
alert(myArray);
```

Output : 10, 20, 30

Declaring an array first using the Array constructor and then populating using the array index. Though the initial size is set to 3, adding a fourth element to the array will not throw an exception, because arrays in JavaScript can grow in size.

```
var myArray = new Array(3);
```

```
myArray[0] = 10;  
myArray[1] = 20;  
myArray[2] = 30;
```

```
alert(myArray);
```

Output : 10, 20, 30

Declaring and populating the array at the same time using the Array constructor

```
var myArray = new Array(10, 20, 30);  
alert(myArray);
```

Output : 10, 20, 30

Please note : If only one number is passed to the Array constructor, then that number is used to set the size of the array. If more than one number is passed then those will be used as elements to populate the array.

A for loop can be used to populate and retrieve elements from the array object in JavaScript

Populating an array using for loop : The following JavaScript code stores even numbers in the array from 0 to 10. Notice that we are using a for loop to populate the array.

```
var evenNumbersArray = [];  
  
for (var i = 0; i <= 5; i++)  
{  
    evenNumbersArray[i] = i * 2;  
}  
alert(evenNumbersArray);
```

Output : 0,2,4,6,8,10

Retrieving elements from the array using for loop :

```
var evenNumbersArray = [];  
for (var i = 0; i <= 5; i++)  
{  
    evenNumbersArray[i] = i * 2;  
}  
  
for (var i = 0; i < evenNumbersArray.length; i++)  
{  
    document.write(evenNumbersArray[i] + "<br/>");  
}
```

Output :

```
0  
2  
4  
6  
8  
10
```

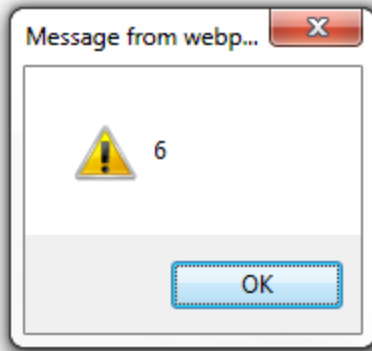
Array Push and Pop methods

In the example below, we are populating **myArray** using a for loop and the array index. Subsequently we are using another for loop to retrieve the elements from the array. Finally we are displaying the length of the array using JavaScript alert.

```
var myArray = [];  
for (var i = 0; i <= 5; i++)  
{  
    myArray[i] = i * 2;  
}  
for (var i = 0; i <= 5; i++)  
{  
    document.write(myArray[i] + "<br/>");  
}  
alert(myArray.length);
```

Output :

0
2
4
6
8
10



Please note : Retrieving array elements using the array index, will not change the length of the array.

JavaScript push() method

This method adds new items to the end of the array. This method also changes the length of the array.

JavaScript pop() method

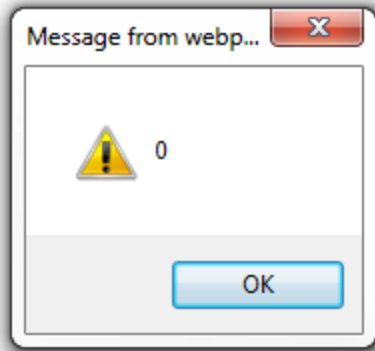
This method removes the last element of an array, and returns that element. This method changes the length of an array.

Example : In the example below, we are using push() method to populate the array and pop() method to retrieve elements from the array. Notice that push() and pop() methods change the length property of the array.

```
var myArray = [];  
for (var i = 0; i <= 5; i++)  
{  
    myArray.push(i * 2);  
}  
alert(myArray.length);  
for (var i = 0; i <= 5; i++)  
{  
    document.write(myArray.pop() + "<br/>");  
}  
alert(myArray.length);
```

Output :

10
8
6
4
2
0



JavaScript unshift() Method

push() method adds new items to the end of the array. To add new items to the beginning of an array, then use unshift() method. Just like push() method, unshift() method also changes the length of an array

Example :

```
var myArray = [2, 3];  
// Adds element 4 after element 3  
myArray.push(4);  
// Adds element 1 before element 2  
myArray.unshift(1);  
document.write("Array elements = " + myArray + "<br/>");  
document.write("Array Length = " + myArray.length);
```

Output :

Array elements = 1,2,3,4
Array Length = 4

JavaScript shift() Method

pop() method removes the last element of an array, and returns that element. shift() method removes the first item of an array, and returns that item. Just like pop() method, shift() method also changes the length of an array.

Example :

```
var myArray = [1, 2, 3, 4, 5];  
// removes the last element i.e 5 from the array  
var lastElement = myArray.pop();  
document.write("Last element = " + lastElement + "<br/>");
```

```
// removes the first element i.e 1 from the array
```

```
var firstElement = myArray.shift();
document.write("First element = " + firstElement + "<br/><br/>");
document.write("Array elements = " + myArray + "<br/>");
document.write("Array Length = " + myArray.length);
```

Output :

Last element = 5

First element = 1

Array elements = 2,3,4

Array Length = 3

Array Mutators

There are several methods that can be used with the array object in JavaScript. Some methods modify the array object while the others do not. **The methods that modify the array object are called as mutator methods.**

The following are the **examples of non-mutator methods**

contains

indexOf

lastIndexOf

The following are the examples of mutator methods

push

pop

shift

unshift

reverse

sort

splice

JavaScript sort method : Sorts the elements of an array. By default, the sort() method sorts the values by converting them to strings and then comparing those strings. This works well for strings but not for numbers. Let us look at an example.

Example : Notice that the strings are sorted correctly as expected.

```
var myArray = ["Sam","Mark","Tom","David"];
myArray.sort();
document.write(myArray);
```

Output : David,Mark,Sam,Tom

Now, let's look at an example of sorting numbers.

```
var myArray = [20, 1, 10, 2, 3];  
myArray.sort();  
document.write(myArray);
```

Output : 1,10,2,20,3

Notice that the numbers are not sorted as expected. We can fix this by providing a "**compare function**" as a parameter to the sort function. The compare function should return a negative, zero, or positive value.

Example :

```
var myArray = [20, 1, 10, 2, 3];  
myArray.sort(function (a, b) { return a - b });  
document.write(myArray);
```

Output : 1,2,3,10,20

Let's now discuss **how the compare function work**. The function has 2 parameters (a,b). This function subtracts b from a and returns the result. If the return value is

Positive	a is a number bigger than b
Negative	a is a number smaller than b
ZERO	a is equal to b

So, based on these return values, the numbers in the array are sorted.

Sorting the numbers in descending order : There are 2 ways to sort an array in descending order

1. Return (b-a) from the compare function instead of (a-b)

Example :

```
var myArray = [20, 1, 10, 2, 3];  
myArray.sort(function (a, b) { return b - a });  
document.write(myArray);
```

Output : 20,10,3,2,1

2. Sort the numbers first in ascending order and then use the reverse function to reverse the order of the elements in the array.

Example :

```
var myArray = [20, 1, 10, 2, 3];  
myArray.sort(function (a, b) { return a - b }).reverse();  
document.write(myArray);
```

Output : 20,10,3,2,1

JavaScript reverse method : reverses the order of the elements in an array.

JavaScript splice method : This method is used to add or remove elements from an array.

Syntax : array.splice(index,deleteCount,item1,.....,itemX)

index	Required. Specifies at what position to add or remove items
deleteCount	Required. The number of items to be removed. If set to 0, no items will be removed.
item1,.....,itemX	Optional. The new item(s) to be added to the array

Example :

```
var myArray = [1,2,5];  
myArray.splice(2, 0, 3, 4);  
document.write(myArray);
```

Output : 1,2,3,4,5

Example :

```
var myArray = [1,2,55,67,3];  
myArray.splice(2, 2);  
document.write(myArray);
```

Output : 1,2,3

Array Filter Method

[Project Name: 11_ArraysInJavaScript]

The filter() method creates a new array and populates that array with all the elements that meet the condition specified in a callback function.

Syntax : array.filter(callbackFunction[, thisArg])

Parameters :

callbackFunction	Required. Function that gets called for each element of the array. If the function returns true, the element is kept otherwise filtered.
thisArg	Optional. An object to which the this keyword can refer in the callbackfn function.

The **filter method calls the callbackfn function one time for each element in the array**. If the callback function returns false for all elements of the array, the length of the new array that will be returned is 0.

Callback Function Syntax

function callbackFunction(value, index, array)

Callback Function Parameters

value	The value of the element in the array
index	The index position of the element in the array
array	The source array object that contains the element

Example 1 : Retrieve only even numbers from myArray

// Callback function

```
function IsEven(value, index, array)
{
    if (value % 2 == 0)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

// Source array

```
var myArray = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
```

// Pass the callback function as argument to the filter method

```
var result = myArray.filter(IsEven);
```

```
document.write(result);
```

Output : 2,4,6,8,10

Example 2 : In **Example 1** we defined a callback function first and then passed it as an argument to the filter() method. In the example below, we created the callback function as an anonymous function directly in the filter method where it is required.

// Source array

```
var myArray = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
```

// callback function created directly in the filter method as anonymous function

```
var result = myArray.filter(function (v, i, a) { return v % 2 == 0 });
```

```
document.write(result);
```

Output : 2,4,6,8,10

Example 3 : Remove duplicates from JavaScript array

```
var myArray = ["Sam", "Mark", "Tim", "Sam"];
```

```
var uniqueItems = myArray.filter(function (v, i, a) { return a.indexOf(v) == i });
```

```
document.write(uniqueItems);
```

Output :
Sam,Mark,Tim

Two Dimensional Array

JavaScript does not have a special syntax for creating multidimensional arrays. Instead we create an array of arrays.

Example : The following JavaScript code creates a 2 dimensional 3x3 array.

```
var myArray1 = new Array(3)
for (i = 0; i < 3; i++)
    myArray1[i] = new Array(3)
```

```
myArray1[0][0] = "1"  
myArray1[0][1] = "2"  
myArray1[0][2] = "3"
```

```
myArray1[1][0] = "4"  
myArray1[1][1] = "5"  
myArray1[1][2] = "6"
```

```
myArray1[2][0] = "7"  
myArray1[2][1] = "8"  
myArray1[2][2] = "9"
```

```
for (var i = 0; i < 3; i++)  
{  
    for (var j = 0; j < 3; j++)  
    {  
        document.write(myArray1[i][j] + "&emsp;");  
    }  
    document.write("<br/>");  
}
```

Output :

1	2	3
4	5	6
7	8	9

In **Example 1**, we have manually populated each storage location in the array. Instead we can use to **2 nested for loops** as shown below.

```
var myArray1 = new Array(3)  
for (i = 0; i < 3; i++)  
    myArray1[i] = new Array(3)
```

```
var start = 1;  
for (var i = 0; i < 3; i++)  
{  
    for (var j = 0; j < 3; j++)  
    {  
        myArray1[i][j] = start;  
        start = start + 1;  
    }  
}
```



```

}
for (var i = 0; i < 3; i++)
{
    for (var j = 0; j < 3; j++)
    {
        document.write(myArray1[i][j] + "&emsp;");
    }
    document.write("<br/>");
}

```

Performing addition between 2 two dimensional arrays in JavaScript

Example :

- The first 3 x 3 array should contain numbers from 1 to 9
- The second 3 x 3 array should contain numbers from 9 to 1
- The numbers present at each index position in the first and second array should be added and the result should be stored in a third 3x3 array.

1	2	3
4	5	6
7	8	9

+

9	8	7
6	5	4
3	2	1

=

10	10	10
10	10	10
10	10	10

// Create the first 2 dimensional 3 X 3 array

```

var myArray1 = new Array(3)
for (i = 0; i < 3; i++)
    myArray1[i] = new Array(3)

```

```

var start = 1;
for (var i = 0; i < 3; i++)

```

```

{
    for (var j = 0; j < 3; j++)
    {
        myArray1[i][j] = start;
        start = start + 1;
    }
}
for (var i = 0; i < 3; i++)
{
    for (var j = 0; j < 3; j++)
    {
        document.write(myArray1[i][j] + "&emsp;");
    }
    document.write("<br/>");
}

```

```

document.write("<br/>");
document.write("&emsp;+");
document.write("<br/>");
document.write("<br/>");

```

// Create the second 2 dimensional 3 X 3 array

```

var myArray2 = new Array(3)
for (i = 0; i < 3; i++)
    myArray2[i] = new Array(3)
for (var i = 0; i < 3; i++)
{
    for (var j = 0; j < 3; j++)
    {
        start = start - 1;
        myArray2[i][j] = start;
    }
}
for (var i = 0; i < 3; i++)
{
    for (var j = 0; j < 3; j++)
    {
        document.write(myArray2[i][j] + "&emsp;");
    }
    document.write("<br/>");
}
document.write("<br/>");
document.write("&emsp;=");
document.write("<br/>");
document.write("<br/>");

```

```

// Create the third 2 dimensional 3 X 3 array
var myArray3 = new Array(3)
for (i = 0; i < 3; i++)
    myArray3[i] = new Array(3)

for (var i = 0; i < 3; i++)
{
    for (var j = 0; j < 3; j++)
    {
        myArray3[i][j] = myArray1[i][j] + myArray2[i][j];
    }
}

for (var i = 0; i < 3; i++)
{
    for (var j = 0; j < 3; j++)
    {
        document.write(myArray3[i][j] + "&emsp;");
    }
    document.write("<br/>");
}

```

Creating a 2 dimensional 3 X 5 array

```

var myArray1 = new Array(3)
for (i = 0; i < 3; i++)
    myArray1[i] = new Array(5)

var start = 101;
for (var i = 0; i < 3; i++)
{
    for (var j = 0; j < 5; j++)
    {
        myArray1[i][j] = start;
        start = start + 1;
    }
}

for (var i = 0; i < 3; i++)
{
    for (var j = 0; j < 5; j++)
    {
        document.write(myArray1[i][j] + "&emsp;");
    }
}

```

```
document.write("<br/>");  
}
```

Output :

```
101  102  103  104  105  
106  107  108  109  110  
111  112  113  114  115
```

Functions in JavaScript

A function is a block of reusable code. A function allows us to write code once and use it as many times as we want just by calling the function.

JavaScript function syntax

```
function functionName(parameter1, parameter2,...parameter_n)  
{  
    //function statements  
}
```

Points to remember

1. Use the function keyword to define a function, followed by the name of the function. The name of the function should be followed by parentheses ().
2. Function parameters are optional. The parameter names must be with in parentheses separated by commas.

Example : JavaScript function to add 2 numbers. The following JavaScript function adds 2 numbers and return the sum

```
function addNumbers(firstNumber, secondNumber)  
{  
    var result = firstNumber + secondNumber;  
    return result;  
}
```

Calling the JavaScript function : Call the JavaScript function by specifying the name of the function and values for the parameters if any.

```
var sum = addNumbers(10, 20);  
alert(sum);
```

Output : 30

What happens when you do not specify values for the function parameters when calling the function

The parameters that are missing values are set to undefined.

Example : In the example below, we are passing 10 for the firstNumber parameter but the secondNumber parameter is missing a value, so this parameter will be set to undefined. When a plus (+) operator is applied between 10 and undefined we get not a number (NaN) and that will be alerted.

```
function addNumbers(firstNumber, secondNumber)
{
    var result = firstNumber + secondNumber;
    return result;
}
```

```
var sum = addNumbers(10);
alert(sum);
```

Output : NaN

What happens when you specify too many parameter values when calling the function. The extra parameter values are ignored.

Example : In the example below, 30 & 40 are ignored.

```
function addNumbers(firstNumber, secondNumber)
{
    var result = firstNumber + secondNumber;
    return result;
}
```

```
var sum = addNumbers(10, 20, 30, 40);
alert(sum);
```

Should a javascript function always return a value

No, they don't have to. It totally depends on what you want the function to do. If an explicit return is omitted, undefined is returned automatically. Let's understand this with an example.

The following function returns the sum of two numbers. We are storing the return value of the function in sum variable and writing it's value to the document.

```
function addNumbers(firstNumber, secondNumber)
{
    var result = firstNumber + secondNumber;
    return result;
}
```

```
var sum = addNumbers(10, 20);  
document.write(sum);
```

The following function does not return any value. It simply writes the sum of two numbers to the page. However, we are assigning the return value of addNumbers() function to sum variable. Since addNumbers() function in this case does not have an explicit return statement, undefined will be returned.

```
function addNumbers(firstNumber, secondNumber)  
{  
    var result = firstNumber + secondNumber;  
    document.write(result);  
}
```

```
var sum = addNumbers(10, 20);  
alert(sum);
```

Different Ways of defining Functions in JavaScript

Defining a function using function declaration

Example 1 : Declaring a function first and then calling it.

```
function addNumbers(firstNumber, secondNumber)  
{  
    var result = firstNumber + secondNumber;  
    return result;  
}
```

```
var sum = addNumbers(10, 20);  
document.write(sum);
```

Output : 30

Example 2 : A function call is present before the function declaration

In Example 1, we are first defining the function and then calling it. The call to a JavaScript function can be present anywhere, even before the function is declared. The following code also works just fine. In the example below, we are calling the function before it is declared.

```
var sum = addNumbers(10, 20);  
document.write(sum);
```

```
function addNumbers(firstNumber, secondNumber)
{
    var result = firstNumber + secondNumber;
    return result;
}
```

Function Hoisting : By default, JavaScript moves all the function declarations to the top of the current scope. This is called function hoisting. This is the reason JavaScript functions can be called before they are declared.

Defining a JavaScript function using a function expression :

A Function Expression allows us to define a function using an expression (typically by assigning it to a variable). There are 3 different ways of defining a function using a function expression.

Anonymous function expression example : In this example, we are creating a function without a name and assigning it to variable add. We use the name of the variable to invoke the function.

```
var add = function (firstNumber, secondNumber)
{
    var result = firstNumber + secondNumber;
    return result;
}
```

```
var sum = add(10, 20);
document.write(sum);
```

Functions defined using a function expression are not hoisted. So, this means a function defined using a function expression can only be called after it has been defined while a function defined using standard function declaration can be called both before and after it is defined.

```
// add() is undefined at this stage
var sum = add(10, 20);
document.write(sum);
```

```
var add = function (firstNumber, secondNumber)
{
    var result = firstNumber + secondNumber;
    return result;
}
```

Named function expression example : This is similar to the example above. The difference is instead of assigning the variable to an anonymous function, we're assigning it to a named function (computeFactorial).

```
var factorial = function computeFactorial(number)
```

```

{
  if (number <= 1)
  {
    return 1;
  }

  return number * computeFactorial(number - 1);
}

```

```

var result = factorial(5);
document.write(result);

```

The name of the function (i.e computeFactorial) is available only within the same function. This syntax is useful for creating recursive functions. If you use computeFactorial() method outside of the function it raises **'computeFactorial' is undefined** error.

```

var factorial = function computeFactorial(number)
{
  if (number <= 1)
  {
    return 1;
  }
  return number * computeFactorial(number - 1);
}

```

```

var result = computeFactorial(5);
document.write(result);

```

Output : Error - **'computeFactorial' is undefined.**

Selfinvoking function expression example :

```

var result = (function computeFactorial(number)
{
  if (number <= 1)
  {
    return 1;
  }

  return number * computeFactorial(number - 1);
})(5);

```

```

document.write(result);

```

Output : 120

These are called with different names
Immediately-Invoked Function Expression (IIFE)
Self-executing anonymous functions
Self-invoked anonymous functions

Local and Global Variables in JavaScript

In JavaScript there are 2 types of variables

1. Local variables
2. Global variables

JavaScript local variables : Local variables are the variables declared with in a function. These variables have local scope meaning these are available only inside the function that contains them. Local variables are created when a function starts, and deleted as soon as the function completes execution.

```
function helloWorld()
{
    var greeting = "Hello";
    // The variable greeting is available in the function
    greeting = greeting + " JavaScript";
    alert(greeting);
}

helloWorld();

// The variable greeting is not available outside the function
// Error : 'greeting' is undefined
alert(greeting);
```

JavaScript global variables : Global variables are the variables declared outside a function. Global variables have global scope meaning all scripts and functions on the page can access them. The lifetime of a global variable starts with its declaration and is deleted when the page is closed.

```
var greeting = "Hello";
function helloWorld()
{
    // The variable greeting is available in the function
    greeting = greeting + " JavaScript";
    alert(greeting);
}
```

```
helloWorld();
```

If you assign a value to a variable that has not been declared, it will automatically become a global variable, even if it is present inside a function.

```
function helloWorld()
{
    // The variable greeting is not declared but a value is assigned.
    // So it will automatically become a global variable
    greeting = "Hello JavaScript";
}
helloWorld();

// Variable greeting is available outside the function
alert(greeting);
```

A local variable can have the same name as a global variable. Changing the value of one variable has no effect on the other. If the variable value is changed inside a function, and if a local version of the variable exists then the local variable gets modified. If the variable value is changed outside a function then the global variable gets modified.

```
var greeting = "This is from global Variable";
function helloWorld()
{
    var greeting = "This is from local variable";
    document.write(greeting + "<br/>");
}
// This line will modify the global greeting variable
greeting += "!!!";
helloWorld();
document.write(greeting);
```

Output :

This is from local variable

This is from global Variable!!!

Sometimes, variable hoisting and local & global variable with the same name can cause unexpected behavior.

```
var greeting = "This is from global Variable";  
helloWorld();  
function helloWorld()  
{  
    document.write(greeting);  
    var greeting = "Hello from local variable"  
}
```

**Output :
undefined**

At runtime due to variable hoisting, the above program would look more like as shown below.

```
var greeting = "This is from global Variable";  
helloWorld();  
function helloWorld(){  
    var greeting;  
    document.write(greeting);  
    greeting = "Hello from local variable"  
}
```

Braces do not create scope in JavaScript : In the following example **otherNumber** is a global variable though it is defined inside braces. In many languages like C# and Java, braces create scope, but not JavaScript.

```
var number = 100;  
if (number > 10)  
{  
    var otherNumber = number;  
}  
document.write(otherNumber);
```

Output : 100

Closures In JavaScript

What is a closure

A closure is an inner function that has access to the outer function's variables in addition to its own variables and global variables. The inner function has access not only to the outer function's variables, but also to the outer function's parameters. You create a closure by adding a function inside another function.

JavaScript Closure Example

```
function addNumbers(firstNumber, secondNumber)
{
    var returnValue = "Result is : ";
    // This inner function has access to the outer function's variables & parameters
    function add()
    {
        return returnValue + (firstNumber + secondNumber);
    }
    return add();
}
```

```
var result = addNumbers(10, 20);
document.write(result);
```

Output : Result is : 30

The following code Returns the inner function expression

```
function addNumbers(firstNumber, secondNumber)
{
    var returnValue = "Result is : ";
    function add()
    {
        return returnValue + (firstNumber + secondNumber);
    }
    // We removed the parentheses. This will return the
    // inner function expression without executing it.
    return add;
}

// addFunc will contain add() function (inner function) expression.
var addFunc = addNumbers(10, 20);
// call the addFunc() function and store the return value in result variable
```

```
var result = addFunc();
```

```
document.write(result);
```

Returning and executing the inner function

```
function addNumbers(firstNumber, secondNumber)
{
    var returnValue = "Result is : ";
    function add()
    {
        return returnValue + (firstNumber + secondNumber);
    }
    // We removed the parentheses. This will return the
    // inner function add() expression without executing it.
    return add;
}
```

```
// This returns add() function (inner function) definition
```

```
// and executes it. Notice the additional parentheses.
```

```
var result = addNumbers(10, 20)();
```

```
document.write(result);
```

Closure Function Example

A **simple JavaScript closure example**. Every time we click a button on a web page, we want to increment the click count by 1. There are several ways we can do this.

Using a global variable and incrementing it every time we click the button : The problem with this approach is that, since clickCount is a global variable any script on the page can accidentally change the variable value.

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

```
    var clickCount = 0;
```

```
</script>
```

```
<input type="button" value="Click Me" onclick="alert(++clickCount);" />
```

Using a local variable with in a function and incrementing it by calling the function : The problem with this approach is that, click count is not incremented beyond 1, no matter how many times you click the button.

```
<script type="text/javascript">
  function incrementClickCount()
  {
    var clickCount = 0;
    return ++clickCount;
  }
</script>
```

```
<input type="button" value="Click Me" onclick="alert(incrementClickCount());" />
```

Using a JavaScript closure : A closure is an inner function that has access to the outer function's variables in addition to its own variables and global variables. In simple terms a closure is function inside a function. These functions, that is the inner and outer functions could be named functions or anonymous functions. In the example below we have an anonymous function inside another anonymous function. The variable incrementClickCount is assigned the return value of the self invoking anonymous function.

```
<script type="text/javascript">
  var incrementClickCount = (function ()
  {
    var clickCount = 0;
    return function ()
    {
      return ++clickCount;
    }
  })();
</script>
```

```
<input type="button" value="Click Me" onclick="alert(incrementClickCount);" />
```

In the example above, in the alert function we are calling the variable incrementClickCount without parentheses. At this point, when you click the button, you get the inner anonymous function expression in the alert. The point we want to prove here is that, the outer self-invoking anonymous function run only once and sets clickCount variable to ZERO, and returns the inner function expression. Inner function has access to clickCount variable. Now every time we click the button, the inner function increments the clickCount variable. The important point to keep in mind is that no other script on the page has access to clickCount variable. The only way to change the clickCount variable is thru incrementClickCount function.

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

```
var incrementClickCount = (function ()
{
    var clickCount = 0;
    return function ()
    {
        return ++clickCount;
    }
})();
</script>
```

```
<input type="button" value="Click Me" onclick="alert(incrementClickCount());" />
```

JavaScript Arguments Object

The **JavaScript arguments object** is a local variable available within all functions. It contains all the function parameters that are passed to the function and can be indexed like an array. The length property of the arguments object returns the number of arguments passed to the function.

JavaScript arguments object example :

```
function printArguments()
{
    document.write("Number of arguments = " + arguments.length + "<br/>")
    for (var i = 0; i < arguments.length; i++)
    {
        document.write("Argument " + i + " = " + arguments[i] + "<br/>");
    }
    document.write("<br/>");
}
```

```
printArguments();
```

```
printArguments("A", "B");
```

```
printArguments(10, 20, 30);
```

Output :

Number of arguments = 0

Number of arguments = 2

Argument 0 = A

Argument 1 = B

Number of arguments = 3

Argument 0 = 10

Argument 1 = 20

Argument 2 = 30

Is it possible to pass variable number of arguments to a JavaScript function

Yes, you can pass as many arguments as you want to any JavaScript function. All the parameters will be stored in the arguments object.

```
function addNumbers()
{
    var sum = 0;
    document.write("Count of numbers = " + arguments.length + "<br/>")
    for (var i = 0; i < arguments.length; i++)
    {
        sum = sum + arguments[i];
    }
    document.write("Sum of numbers = " + sum);
    document.write("<br/><br/>");
}
```

addNumbers();

addNumbers(10, 20, 30);

Output :

Count of numbers = 0

Sum of numbers = 0

Count of numbers = 3

Sum of numbers = 60

The arguments object is available only inside a function body. Attempting to access the arguments object outside a function results in **'arguments' is undefined** error. Though you can index the arguments object like an array, it is not an array. It does not have any Array properties except length. For example it does not have the sort() method, that the array object has.

However, you can convert the arguments object to an array.

Converting JavaScript arguments object to an array

```
function numbers()
{
    var argsArray = Array.prototype.slice.call(arguments);
    argsArray.sort();
    document.write(argsArray);
}
```

numbers(50, 20, 40);

Output : 20, 40, 50

Converting JavaScript arguments object to an array using array literals

```
function numbers()
{
    var argsArray = [].slice.call(arguments);
    argsArray.sort();
    document.write(argsArray);
}
```

numbers(50, 20, 40);

Output : 20, 40, 50

Recursive Function Example

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

What is a recursive function?

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

```
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;  
}  
  
document.write(factorial(5));
```

Output : 120

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

```
function factorial(n)  
{  
    if (n == 0 || n == 1)  
    {  
        return 1;  
    }  
    return n * factorial(n - 1);  
}  
  
document.write(factorial(5));
```

Output : 120

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 :

```

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/>");
}

document.write("This line will be executed");

```

Output :

```

Description = 'sayHello' is undefined
Message = 'sayHello' is undefined
Stack = ReferenceError: 'sayHello' is undefined at Global code
(http://localhost:52212/HTMLPage1.htm:4:9)

```

This line will be executed

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.

JavaScript try catch finally example :

```

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");
}

```

Output :

```

Description = 'sayHello' is undefined
Message = 'sayHello' is undefined
Stack = ReferenceError: 'sayHello' is undefined at Global code
(http://localhost:52212/HTMLPage1.htm:4:9)

```

This line is guaranteed to execute

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.

```

try
{
    alert("Hello";
}
catch (e)
{
    document.write("JavaScript syntax errors cannot be caught in the catch block");
}

```

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

JavaScript throw exception example :

```

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/>");
}
}

divide();

```

window.onerror Event

In general **we use try/catch statement to catch errors in JavaScript**. If an error is raised by a statement that is not inside a try...catch block, the onerror event is fired.

Assign a function to window.onerror event that you want to be executed when an error is raised as shown below. The function that is associated as the event handler for the onerror event has three parameters:

message	Specifies the error message
URL	Specifies the location of the file where the error occurred
line	Specifies the line number where the error occurred

JavaScript window onerror event example

```

window.onerror = function (message, url, line)
{
  alert("Message : " + message + "\nURL : " + url + "\nLine Number : " + line);
}

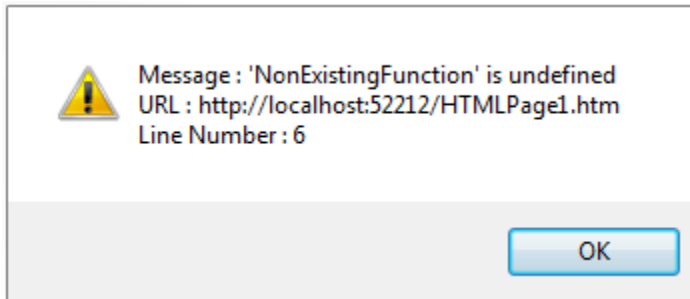
```

```

// Return true to suppress the browser error messages
// (like in older versions of Internet Explorer)
return true;
}
NonExistingFunction();

```

Output :



If the error is handled by a try/catch statement, then the onerror event is not raised. onerror event is raised only when there is an unhandled exception.

```

window.onerror = function (message, url, line)
{
    alert("Message : " + message + "\nURL : " + url + "\nLine Number : " + line);
    return true;
}
try
{
    NonExistingFunction();
}
catch (e)
{
    document.write(e.message);
}

```

Output : 'NonExistingFunction' is undefined

onerror event handler method can also be used with HTML elements : In the example below, since the image is not existing and cannot be found we get "There is a problem loading the image" error.

```

<script type="text/javascript">
    function imageErrorHandler()
    {
        alert("There is a problem loading the image");
    }

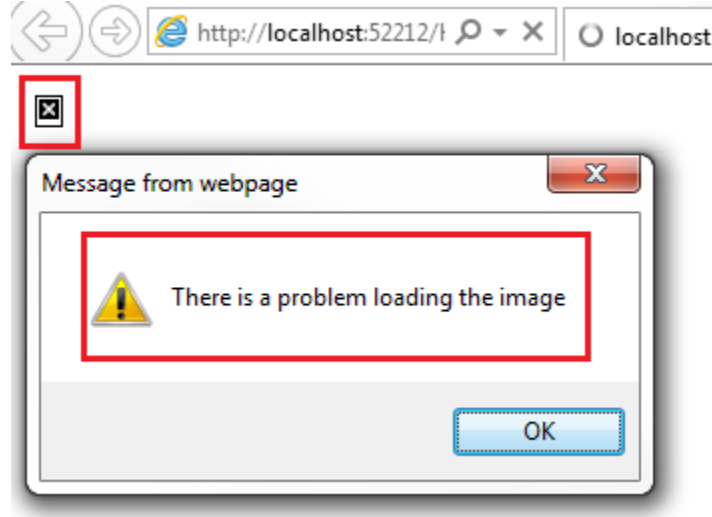
```

```
</script>
```

```

```

Output :



Dates in JavaScript

To create date object in JavaScript use Date() constructor

The following example writes the current Date and Time to the web page
`document.write(new Date());`

If the **Date()** constructor is used without any arguments, it returns the current date and time. To create a date object with specific dates there are 2 ways.

Creating a specific date object in JavaScript using a date string

```
var dateOfBirth = new Date("January 13, 1980 11:20:00");  
document.write(dateOfBirth);
```

You can also create a specific date object using number for year, month, day, hours, minutes, seconds, & milliseconds. The syntax is shown below.

```
var dateOfBirth = new Date(year, month, day, hours, minutes, seconds, milliseconds);
```

Example :

```
var dateOfBirth = new Date(1980, 0, 13, 11, 20, 0, 0);  
document.write(dateOfBirth);
```

Please note : In JavaScript month numbers start from ZERO. So if you want the month of march then use 2 instead of 3.

The above code produces the following output on my machine, because **(UTC) Dublin, Edinburgh, Lisbon, London** time zone selected on machine
Sun Jan 13 1980 11:20:00 GMT+0000 (GMT Standard Time)

If you have a different time zone selected on your computer, you may get a slightly different result. For example if you have **(UTC+05:30) Chennai, Kolkata, Mumbai, New Delhi** time zone selected, the result will be as shown below

Sun Jan 13 1980 11:20:00 GMT+0530 (India Standard Time)

Some useful Date object methods in JavaScript

getFullYear() - Returns the full year (all the four digits)

Example : The following example returns 1980

```
var year = new Date(1980, 0, 13, 11, 20, 0, 0).getFullYear();  
document.write(year);
```

getMonth() - Returns the month number (from 0-11)

Example : The following example returns 0 (for January)

```
var month = new Date(1980, 0, 13, 11, 20, 0, 0).getMonth();  
document.write(month);
```

You can use the following code to **get the month name from the month number** in Javascript. The following example returns January.

```
function getMonthNameFromNumber(monthNumber)  
{  
    var monthNames = ["January", "February", "March", "April",  
                      "May", "June", "July", "August", "September",  
                      "October", "November", "December"];  
    return monthNames[monthNumber];  
}
```

```
var monthName = getMonthNameFromNumber(new Date(1980, 0, 13, 11, 20, 0,  
0).getMonth());  
document.write(monthName);
```

getDate() - Returns the day of the month (from 1-31)

Example : The following example returns 13

```
var dayOfMonth = new Date(1980, 0, 13, 11, 20, 0, 0).getDate();  
document.write(dayOfMonth);
```


getDay() - Returns the day number of the week (from 0-6). 0 is sunday, 1 is monday.

Example : The following example returns 0

```
var dayOfWeek = new Date(1980, 0, 13, 11, 20, 0, 0).getDay();
document.write(dayOfWeek);
```

You can use the following code **to get the day of the week from the day number** in Javascript. The following example returns Sunday.

```
function getWeekDayNameFromNumber(dayNumber)
{
    var weekDays = ["Sunday", "Monday", "Tuesday", "Wednesday",
                    "Thursday", "Friday", "Saturday"];
    return weekDays[dayNumber];
}
```

```
var weekdayName = getWeekDayNameFromNumber(new Date(1980, 0, 13, 11, 20, 0,
0).getDay());
document.write(weekdayName);
```

You also have the following methods that return the time parts of the date object

getHours()	Returns the hour (from 0-23)
getMinutes()	Returns the minutes (from 0-59)
getSeconds()	Returns the seconds (from 0-59)
getMilliseconds()	Returns the milliseconds (from 0-999)

How to convert date in javascript to dd/mm/yyyy format

```
function formatDate(date)
{
    var day = date.getDate();
    if (day < 10)
    {
        day = "0" + day;
    }
    var month = date.getMonth() + 1;
    if (month < 10)
    {
        month = "0" + month;
    }
    var year = date.getFullYear();
    return day + "/" + month + "/" + year;
}
```

```
document.write(formatDate(new Date()));
```

If you don't want ZERO (0) before a single digit month or day number, then modify the formatDate() function as shown below.

```
function formatDate(date)
{
    var day = date.getDate();
    var month = date.getMonth() + 1;
    var year = date.getFullYear();
    return day + "/" + month + "/" + year;
}
```

```
document.write(formatDate(new Date()));
```

JavaScript Timing Events

[Project Name: 13_DateEvents]

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.

Let's understand timing events in JavaScript with an example. The following code displays current date and time in the div tag.

```
<div id="timeDiv" ></div>
```

```
<script type="text/javascript">
    function getCurrentDateTime()
    {
```

```
document.getElementById("timeDiv").innerHTML = new Date();
}
```

```
getCurrentDateTime();
</script>
```

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.

```
<div id="timeDiv" ></div>
```

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

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

clearInterval(intervalID) - Cancels the repeated execution of the method that was set up using **setInterval()** method. **intervalID** is the identifier of the repeated action you want to cancel. This ID is returned from **setInterval()** method. The following example demonstrates the use of **clearInterval()** method.

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.

```
<div id="timeDiv" ></div>
```

```
<br />
```

```
<input type="button" value="Start Clock" onclick="startClock()" />
```

```
<input type="button" value="Stop Clock" onclick="stopClock()" />
```

```
<script type="text/javascript">
    var intervalId;
    function startClock()
    {
        intervalId = setInterval(getCurrentDateTime, 1000);
    }
    function stopClock()
    {
        clearInterval(intervalId);
    }
</script>
```

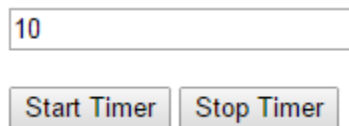
```

    }
    function getCurrentDateTime()
    {
        document.getElementById("timeDiv").innerHTML = new Date();
    }
    getCurrentDateTime();
</script>

```

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.



```

<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()" />
<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);
    }

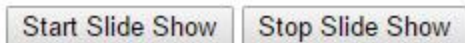
```

```
function stopTimer()
{
    clearTimeout(intervalId);
}
</script>
```

Create Image Slideshow using JavaScript

Creating a simple image slideshow using JavaScript. We will be using **setInterval()** and **clearInterval()** JavaScript methods to achieve this.

The slideshow should be as shown in the image below. 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.



For the purpose of this we will be using the images that can be found on any windows machine at the following path.

C:\Users\Public\Pictures\Sample Pictures

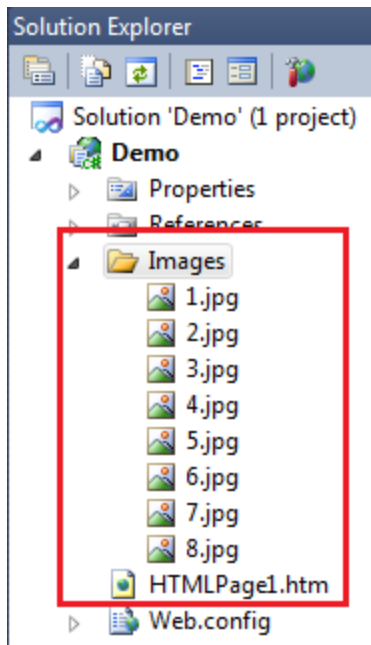
Here are the steps to create the image slideshow using JavaScript.

Step 1 : Open Visual Studio and create a new empty asp.net web application project. Name it Demo.

Step 2 : Right click on the Project Name in Solution Explorer in Visual Studio and create a new folder with name = Images.

Step 3 : Copy the images from C:\Users\Public\Pictures\Sample Pictures to Images folder in your project. Change the names of the images to 1.jpg, 2.jpg etc.

Step 4 : Right click on the Project Name in Solution Explorer in Visual Studio and add a new HTML Page. It should automatically add **HTMLPage1.htm**. At this point your solution explorer should look as shown below.



Step 5 : Copy and paste the following HTML and JavaScript code in HTMLPage1.htm page.

```

```

```
<br />
```

```
<input type="button" value="Start Slide Show" onclick="startImageSlideShow()" />
```

```
<input type="button" value="Stop Slide Show" onclick="stopImageSlideShow()" />
```

```
<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 == 8)
```

```
        {
```

```
            currentImageNumber = 0;
```

```
        }
```

```
        document.getElementById("image").setAttribute("src", "/Images/" +  
                                                    (Number(currentImageNumber) + 1) + ".jpg");
```

```
    }
```

`</script>`

Finally run the application and test it.

Events in JavaScript

[Project Name: 14_Events]

What is an event

An event is a signal from the browser that something has happened. For example,

1. When a user clicks on an HTML element, click event occurs
2. When a user moves the mouse over an HTML element, mouseover event occurs

When events occur, we can execute JavaScript code or functions in response to those events. To do this we need to associate JavaScript code or functions to the events. The function that executes in response to an event is called event handler.

In JavaScript, there are several ways to associate an event handler to the event

1. Using the attributes of an HTML tag
2. Using DOM object property
3. Using special methods

In the following example, the code to execute in response to **onmouseover** & **onmouseout** events is set directly in the HTML markup. The keyword "this" references the current element. In this example "**this**" references the button control.

```
<input type="button" value="Click me" id="btn"
onmouseover="this.style.background= 'red'; this.style.color = 'yellow'"
onmouseout="this.style.background= 'black'; this.style.color = 'white'" />
```

The above example, can be rewritten as shown below. In this case the code to execute in response to the event is placed inside a function and then the function is associated with the event.

```
<input type="button" value="Click me" id="btn"
onmouseover="changeColorOnMouseOver()"
onmouseout="changeColorOnMouseOut()" />
```

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

```

function changeColorOnMouseOver()
{
    var control = document.getElementById("btn");
    control.style.background = 'red';
    control.style.color = 'yellow';
}
function changeColorOnMouseOut()
{
    var control = document.getElementById("btn");
    control.style.background = 'black';
    control.style.color = 'white';
}
</script>

```

Events are very useful in real-world applications. For example they can be used to

1. Display confirmation dialog box on submitting a form
2. Form data validation and many more

How to show confirmation dialog in JavaScript

```

<input type="submit" value="Submit" id="btn" onclick="return confirmSubmit()" />
<script type="text/javascript">
function confirmSubmit()
{
    if (confirm("Are you sure you want to submit"))
    {
        alert("You selected OK");
        return true;
    }
    else
    {
        return false;
        confirm("You selected cancel");
    }
}
</script>

```

JavaScript form validation example : In this example, both First Name and Last Name fields are required fields. When you type the first character in any of the textbox, the background color is automatically changed to green. If you delete all the characters you typed or if you leave the textbox without entering any characters the background color changes to red indicating the field is required. We made this possible using onkeyup and onblur events.

onkeyup occurs when the user releases a key.

onblur occurs when an element loses focus.


```

<table>
  <tr>
    <td>
      First Name
    </td>
    <td>
      <input type="text" id="txtFirstName"
        onkeyup="validateRequiredField('txtFirstName')"
        onblur="validateRequiredField('txtFirstName')"/>
    </td>
  </tr>
  <tr>
    <td>
      Last Name
    </td>
    <td>
      <input type="text" id="txtLastName"
        onkeyup="validateRequiredField('txtLastName')"
        onblur="validateRequiredField('txtLastName')"/>
    </td>
  </tr>
</table>
<script type="text/javascript">
  function validateRequiredField(controlId)
  {
    var control = document.getElementById(controlId);
    control.style.color = 'white';
    if (control.value == "")
    {
      control.style.background = 'red';
    }
    else
    {
      control.style.background = 'green';
    }
  }
</script>

```

Assigning Event Handlers in JavaScript using DOM object Property

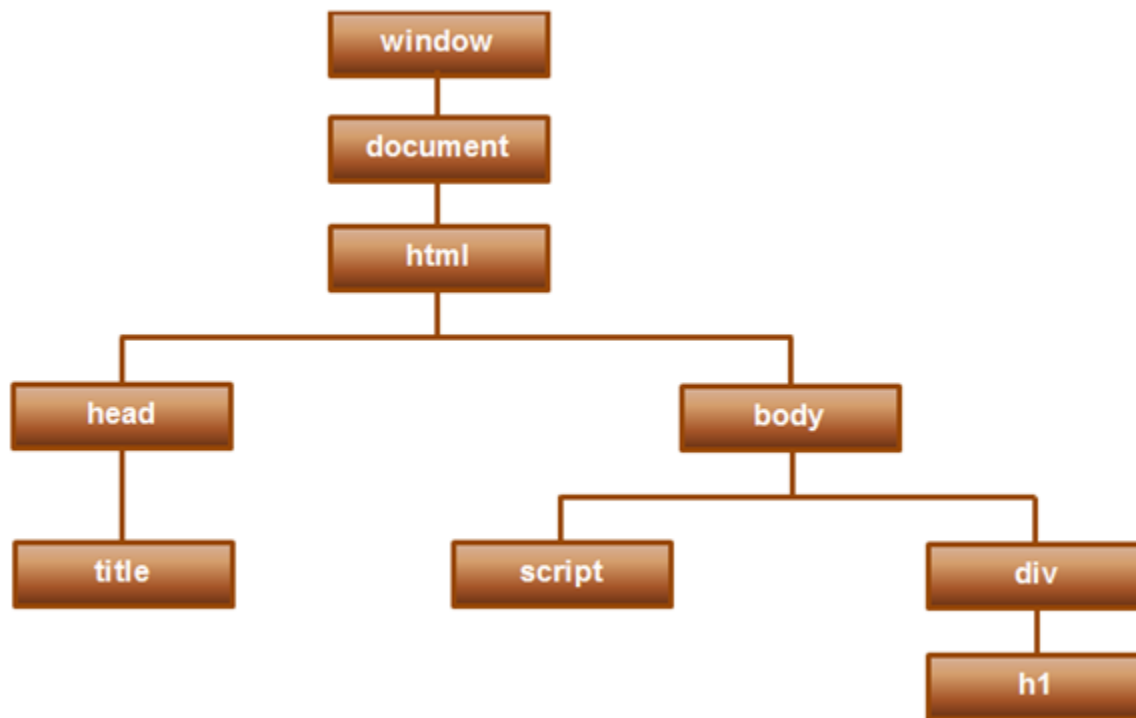
What is DOM

DOM stands for Document Object Model. When a browser loads a web page, the browser creates a Document Object Model of that page. The HTML DOM is created as a tree of Objects.

Example :

```
<html>
  <head>
    <title>My Page Title</title>
  </head>
  <body>
    <script type="text/javascript">
    </script>
    <div>
      <h1>This is browser DOM</h1>
    </div>
  </body>
</html>
```

For the above HTML a **graphical representation of the Document Object Model** is shown below.



JavaScript can be used to access and modify these DOM objects and their properties. For example, you can add, modify and remove HTML elements and their attributes. Along the same lines, you can use DOM object properties to assign event handlers to events.

Notice that, we are assigning event handlers using the DOM object properties (**onmouseover** & **onmouseout**) instead of using the attributes of the HTML tag. We are using this keyword to reference the current HTML element. In this example "**this**" references the button control.

```

<input type="button" value="Click me" id="btn"/>
<script type="text/javascript">
    document.getElementById("btn").onmouseover = changeColorOnMouseOver;
    document.getElementById("btn").onmouseout = changeColorOnMouseOut;

    function changeColorOnMouseOver()
    {
        this.style.background = 'red';
        this.style.color = 'yellow';
    }
    function changeColorOnMouseOut()
    {
        this.style.background = 'black';
        this.style.color = 'white';
    }
</script>

```

The following example is same as the above. In this case we are assigning an anonymous function to **onmouseover** & **onmouseout** properties.

```

<input type="button" value="Click me" id="btn" />
<script type="text/javascript">
    document.getElementById("btn").onmouseover = function ()
    {
        this.style.background = 'red';
        this.style.color = 'yellow';
    }
    document.getElementById("btn").onmouseout = function ()
    {
        this.style.background = 'black';
        this.style.color = 'white';
    }
</script>

```

If an **event handler** is assigned using both, i.e an HTML attribute and DOM object property, the handler that is assigned using the DOM object property overwrites the one assigned using HTML attribute. Here is an example.

```

<input type="button" value="Click me" id="btn" onclick="clickHandler1()"/>
<script type="text/javascript">
    document.getElementById("btn").onclick = clickHandler2;
    function clickHandler1()

```

```

{
    alert("Handler set using HTML attribute");
}
function clickHandler2()
{
    alert("Handler set using DOM object property");
}
</script>

```

Using this approach you can only assign one event handler method to a given event. The handler that is assigned last wins. In the following example, **Handler2()** is assigned after **Handler1**. So **Handler2()** overwrites **Handler1()**.

```

<input type="button" value="Click me" id="btn"/>
<script type="text/javascript">
    document.getElementById("btn").onclick = clickHandler1;
    document.getElementById("btn").onclick = clickHandler2;
    function clickHandler1()
    {
        alert("Handler 1");
    }
    function clickHandler2()
    {
        alert("Handler 2");
    }
</script>

```

Assigning Event Handlers in JavaScript using Special Methods

We can assign event handlers in JavaScript using the following special methods

- addEventListener
- removeEventListener
- attachEvent
- detachEvent

Internet Explorer 9 (and later versions) & all the other modern browsers support [addEventListener\(\)](#) and [removeEventListener\(\)](#) methods.

Syntax for assigning event handler using `addEventListener()` method
`element.addEventListener(event, handler, phase)`

Syntax for removing event handler using `removeEventListener()` method

element.removeEventListener(event, handler, phase)

Please note : The third parameter phase is usually set to false as it is not used.

Example : In this example, we are passing values for all the 3 parameters including phase.

```
<input type="button" value="Click me" id="btn"/>
<script type="text/javascript">
  btn.addEventListener("mouseover", changeColorOnMouseOver, false);
  btn.addEventListener("mouseout", changeColorOnMouseOut, false);
  function changeColorOnMouseOver()
  {
    this.style.background = 'red';
    this.style.color = 'yellow';
  }
  function changeColorOnMouseOut()
  {
    this.style.background = 'black';
    this.style.color = 'white';
  }
</script>
```

Since the third parameter "phase" is optional you can omit it if you wish.

```
btn.addEventListener("mouseover", changeColorOnMouseOver);
btn.addEventListener("mouseout", changeColorOnMouseOut);
```

Example : This example demonstrates removing event handlers.

```
<input type="button" value="Click me" id="btn"/>
<input type="button" value="Remove Event Handlers" onclick="removeEventHandlers()" />
<script type="text/javascript">
  btn.addEventListener("mouseover", changeColorOnMouseOver);
  btn.addEventListener("mouseout", changeColorOnMouseOut);
  function changeColorOnMouseOver()
  {
    this.style.background = 'red';
    this.style.color = 'yellow';
  }
  function changeColorOnMouseOut()
  {
    this.style.background = 'black';
    this.style.color = 'white';
  }
  function removeEventHandlers()
  {
    btn.removeEventListener("mouseover", changeColorOnMouseOver);
```

```
        btn.removeEventListener("mouseout", changeColorOnMouseOut);
    }
</script>
```

Using this approach you can assign more than one event handler method to a given event. The order in which handler methods are executed is not defined. In this example, 2 event handler methods (clickHandler1 & clickHandler2) are assigned to click event of the button control. When you click the button both the handler methods are executed.

```
<input type="button" value="Click me" id="btn"/>
<script type="text/javascript">
    btn.addEventListener("click", clickHandler1);
    btn.addEventListener("click", clickHandler2);
    function clickHandler1()
    {
        alert("Handler 1");
    }
    function clickHandler2()
    {
        alert("Handler 2");
    }
</script>
```

`attachEvent()` and `detachEvent()` methods only work in Internet Explorer 8 and earlier versions.

Syntax for assigning event handler using `attachEvent()` method
`element.attachEvent("on"+event, handler)`

Syntax for removing event handler using `detachEvent()` method
`element.detachEvent("on"+event, handler)`

Example : This example will only work in Internet Explorer 8 and earlier versions.

```
<input type="button" value="Click me" id="btn"/>
<script type="text/javascript">
    btn.attachEvent("onclick", clickEventHandler);
    function clickEventHandler()
    {
        alert("Click Handler");
    }
</script>
```

Cross browser solution : For the above example to work in all browsers, modify the script as shown below.

```
<input type="button" value="Click me" id="btn"/>
```

```

<script type="text/javascript">
  if (btn.addEventListener)
  {
    btn.addEventListener("click", clickEventHandler);
  }
  else
  {
    btn.attachEvent("onclick", clickEventHandler);
  }
  function clickEventHandler()
  {
    alert("Click Handler");
  }
</script>

```

JavaScript Event Object

Whenever an event (like click, mouseover, mouseout etc) occurs, the relevant data about that event is placed into the event object. For example, the event object contains event data like, the X and Y coordinates of the mouse pointer when the event occurred, the HTML element that fired the event, which mouse button is clicked etc.

Obtaining the event object is straightforward. The event object is always passed to the event handler method. Let us understand this with an example. When we click the button, we want to capture the following event data

1. Event name
2. Mouse X coordinate when the event occurred
3. Mouse Y coordinate when the event occurred
4. The control that raised the event
5. The HTML tag name that raised the event

Click me

Event = click

X = 45

Y = 20

Target Type = button

Target Tag Name = INPUT

Notice that in the example below, we are passing event object as a parameter to the event handler method. The type property gives us the event name that occurred. clientX and clientY

properties return the X and Y coordinates of the mouse pointer. Target property returns the HTML element that raised the event. Target property is supported by all modern browsers and Internet Explorer 9 and above. **The following code will not work in Internet Explorer 8 and earlier versions.** In addition to click event, the following example returns mouseover and mouseout event data.

```
<input type="button" value="Click me" id="btn"
  onclick="displayEventDetails(event)"
  onmouseover="displayEventDetails(event)"
  onmouseout="displayEventDetails(event)" />
<br /><br />
<div id="resultDiv"></div>
<script type="text/javascript">
  function displayEventDetails(event)
  {
    var eventDetails = "Event = " + event.type + "<br/> X = " + event.clientX
      + "<br/>Y = " + event.clientY + "<br/>Target Type = "
      + event.target.type + "<br/>Target Tag Name = "
      + event.target.tagName;
    document.getElementById("resultDiv").innerHTML = eventDetails;
  }
</script>
```

The following code works in all browsers including Internet Explorer 8 and earlier versions. IE 8 and earlier versions use **srcElement** property to return the HTML element that raised the event. IE 9 and all the other modern browsers use **target** property. So this is a cross browser solution.

```
<input type="button" value="Click me" id="btn"
  onclick="displayEventDetails(event)"
  onmouseover="displayEventDetails(event)"
  onmouseout="displayEventDetails(event)" />
<br /><br />
<div id="resultDiv"></div>
<script type="text/javascript">
  function displayEventDetails(event)
  {
    var sourceElement;
    if (event.srcElement)
    {
      sourceElement = event.srcElement;
    }
    else
    {
      sourceElement = event.target;
    }
    var eventDetails = "Event = " + event.type + "<br/> X = " + event.clientX
```

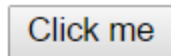


```

        + "<br/>Y = " + event.clientY + "<br/>Target Type = "
        + sourceElement.type + "<br/>Target Tag Name = "
        + sourceElement.tagName;
    document.getElementById("resultDiv").innerHTML = eventDetails;
}
</script>

```

The following example retrieves **mousemove** event data. Notice that as you move the mouse pointer over the button, the X & Y coordinates changes.



```

Event = mousemove
X = 70
Y = 25
Target Type = button
Target Tag Name = INPUT

```

```

<input type="button" value="Click me" id="btn"
    onmousemove="displayEventDetails(event)" />
<br /><br />
<div id="resultDiv"></div>
<script type="text/javascript">
    function displayEventDetails(event)
    {
        var sourceElement;
        if (event.srcElement)
        {
            sourceElement = event.srcElement;
        }
        else
        {
            sourceElement = event.target;
        }
        var eventDetails = "Event = " + event.type + "<br/> X = " + event.clientX
            + "<br/>Y = " + event.clientY + "<br/>Target Type = "
            + sourceElement.type + "<br/>Target Tag Name = "
            + sourceElement.tagName;

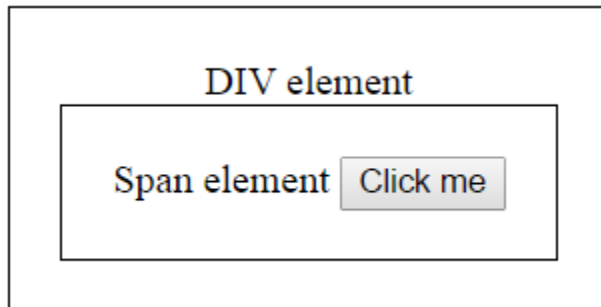
        document.getElementById("resultDiv").innerHTML = eventDetails;
    }
</script>

```

Event Bubbling in JavaScript

What is event bubbling

Let us understand this with an example. HTML elements can be nested inside each other. For example a button element can be nested inside a span element and the span element in turn can be nested inside a div element as shown below.



Notice that we have **onclick attribute specified for all the 3 elements.**

```
<html>
<head>
  <style type="text/css">
    .styleClass
    {
      display: table-cell;
      border: 1px solid black;
      padding: 20px;
      text-align: center;
    }
  </style>
</head>
<body>
  <div class="styleClass" onclick="alert('Div click handler')">
    DIV element
    <span class="styleClass" onclick="alert('Span click handler')">
      Span element
      <input type="button" value="Click me" onclick="alert('Button click handler')" />
    </span>
  </div>
</body>
</html>
```

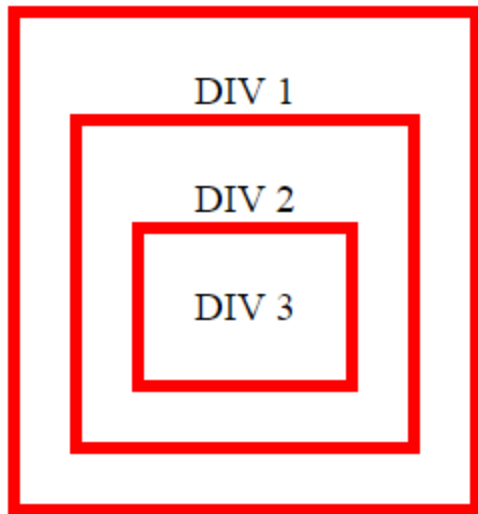
A click on the button, causes a click event to be fired on the button. The button click event handler method handles the event. The click event then bubbles up to the button element parent (span element), which is handled by the span element event handler method. The click event then bubbles up to the span element parent (div element). This is called event bubbling.

Notice that if you click on the element, it's event handler and it's parent(<div>) element event handler are called. If you click on the <div> element, just the <div> element event handler method is called. So, the event bubbling process starts with the element that triggered the event and then bubbles up to the containing elements in the hierarchy.

The following example is similar to the one above, except we removed the onclick attribute from button and span elements. Because of event bubbling, when you click on the button or the span element, the event gets handled by the div element handler.

```
<html>
<head>
  <style type="text/css">
    .styleClass
    {
      display: table-cell;
      border: 1px solid black;
      padding: 20px;
      text-align: center;
    }
  </style>
</head>
<body>
  <div class="styleClass"
    onclick="alert('Click event handled by DIV element')">DIV element
    <span class="styleClass">Span element
      <input type="button" value="Click me"/>
    </span>
  </div>
</body>
</html>
```

When the event gets bubbled, the keyword `this` references the current element to which the event is bubbled. In the example below, we are using `"this"` keyword to reference the current div element and change its border color. When you click on the innermost <div> element, all the 3 <div> elements border get changed due to event bubbling.



```
<html>
<head>
  <style type="text/css">
    .divStyle
    {
      display: table-cell;
      border: 5px solid black;
      padding: 20px;
      text-align: center;
    }
  </style>
</head>
<body>
  <div id="DIV1" class="divStyle">
    DIV 1
    <div id="DIV2" class="divStyle">
      DIV 2
      <div id="DIV3" class="divStyle">
        DIV 3
      </div>
    </div>
  </div>
  <script type="text/javascript">
    var divElements = document.getElementsByTagName('div');
    for (var i = 0; i < divElements.length; i++)
    {
      divElements[i].onclick = function ()
      {
        this.style.borderColor = "red";
        alert(this.getAttribute("id") + " background changed");
      }
    }
  </script>
</body>
</html>
```

```

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

```

Stopping event bubbling : If you don't want the event to be bubbled up, you can stop it.

With Internet Explorer 8 and earlier versions

event.cancelBubble = **true**

With Internet Explorer 9 (and later versions) & all the other browsers

event.stopPropagation()

In the example below we have stopped event bubbling. So, when you click on DIV 1, only DIV 1 border colour is changed.

```

<html>
<head>
  <style type="text/css">
    .divStyle
    {
      display: table-cell;
      border: 5px solid black;
      padding: 20px;
      text-align: center;
    }
  </style>
</head>
<body>
  <div id="DIV1" class="divStyle">
    DIV 1
    <div id="DIV2" class="divStyle">
      DIV 2
      <div id="DIV3" class="divStyle">
        DIV 3
      </div>
    </div>
  </div>
  <script type="text/javascript">
    var divElements = document.getElementsByTagName('div');
    for (var i = 0; i < divElements.length; i++)
    {
      divElements[i].onclick = function (event)
      {

```

```

        event = event || window.event;
        if (event.stopPropagation)
        {
            event.stopPropagation();
        } else
        {
            event.cancelBubble = true;
        }
        this.style.borderColor = "red";
        alert(this.getAttribute("id") + " background changed");
    }
}
</script>
</body>
</html>

```

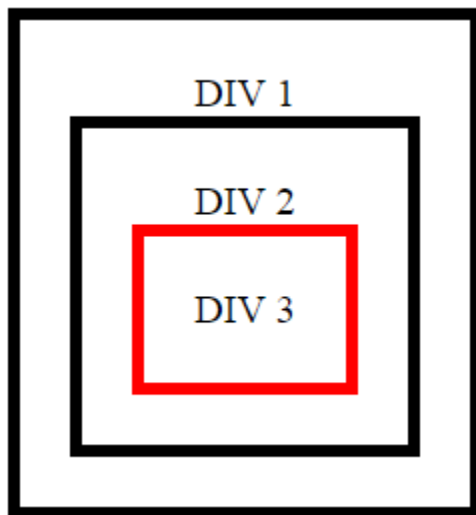


Image Gallery with Thumbnails in JavaScript

[Project Name: 15_ImageGalleryWithThumbnails]

We will take advantage of event bubbling to achieve this.

The **image gallery** should be as shown in the image below. When you click on the image thumbnail, the respective image should be displayed in the main section of the page.



For the purpose of this demo we will be using the images that can be found on any windows machine at the following path.

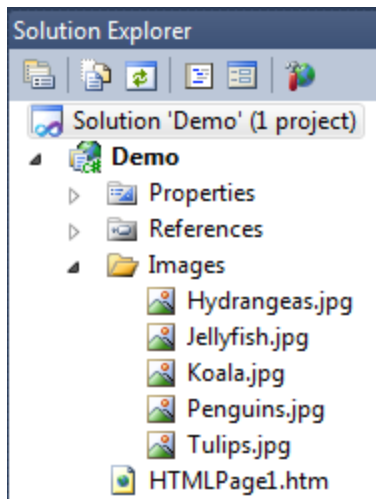
C:\Users\Public\Pictures\Sample Pictures

Step 1 : Open Visual Studio and create a new empty asp.net web application project. Name it Demo.

Step 2 : Right click on the Project Name in Solution Explorer in Visual Studio and create a new folder with name = Images.

Step 3 : Copy images from C:\Users\Public\Pictures\Sample Pictures to Images folder in your project.

Step 4 : Right click on the Project Name in Solution Explorer in Visual Studio and add a new HTML Page. It should automatically add HTMLPage1.htm. At this point your solution explorer should look as shown below.



Step 5 : Copy and paste the following HTML and JavaScript code in HTMLPage1.htm page.

```
<html>
<head>
  <style type="text/css">
    .imgStyle
    {
      width:100px;
      height:100px;
      border:3px solid grey;
    }
  </style>
</head>
<body>
  
  <br />
  <div id="divId" onclick="changeImageOnClick(event)">
    
    
    
    
    
  </div>
  <script type="text/javascript">

    var images = document.getElementById("divId")
      .getElementsByTagName("img");
```



```

for (var i = 0; i < images.length; i++)
{
    images[i].onmouseover = function ()
    {
        this.style.cursor = 'hand';
        this.style.borderColor = 'red';
    }
    images[i].onmouseout = function ()
    {
        this.style.cursor = 'pointer';
        this.style.borderColor = 'grey';
    }
}

function changeImageOnClick(event)
{
    event = event || window.event;
    var targetElement = event.target || event.srcElement;

    if (targetElement.tagName == "IMG")
    {
        mainImage.src = targetElement.getAttribute("src");
    }
}
</script>
</body>
</html>

```

Finally run the application and test it.

JavaScript Event Capturing

Event capturing is the opposite of event bubbling.

With event bubbling the event bubbles up from the inner most element to the outer most element in the DOM hierarchy. With event capturing the opposite happens, the event gets captured from the outer most element to innermost element. In real world event capturing is rarely used. Let us understand this with an example.

In the following example we have **3 <div>** nested elements. Notice that we are using

addEventListener() method to assign event handler to each <div> element. **To enable event capturing we have set the third parameter (phase) of addEventListener() method to true.**

Now when you click on the innermost <div> (DIV3), notice that the events are fired from the outermost <div> to innermost <div> (DIV1 => DIV 2 => DIV 3).

```
<html>
<head>
  <style type="text/css">
    .divStyle
    {
      display: table-cell;
      border: 5px solid black;
      padding: 20px;
      text-align: center;
    }
  </style>
</head>
<body>
  <div id="DIV1" class="divStyle">
    DIV 1
    <div id="DIV2" class="divStyle">
      DIV 2
      <div id="DIV3" class="divStyle">
        DIV 3
      </div>
    </div>
  </div>
  <script type="text/javascript">
    var divElements = document.getElementsByTagName('div');
    for (var i = 0; i < divElements.length; i++)
    {
      divElements[i].addEventListener("click", clickHandler, true);
    }
    function clickHandler()
    {
      alert(this.getAttribute("id") + " click event handled");
    }
  </script>
</body>
</html>
```

Please note : IE8 and earlier versions does not support addEventListener() method. This implies that event capturing is not supported in IE8 and earlier versions, and hence the above code will not work in IE 8 and earlier versions.

Stopping event capturing : Stopping event capturing is very similar to stopping event bubbling. With the example below, when you click on any <div> element notice that DIV 1 element handles the click event and it does not get captured down.

```
<html>
<head>
  <style type="text/css">
    .divStyle
    {
      display: table-cell;
      border: 5px solid black;
      padding: 20px;
      text-align: center;
    }
  </style>
</head>
<body>
  <div id="DIV1" class="divStyle">
    DIV 1
    <div id="DIV2" class="divStyle">
      DIV 2
      <div id="DIV3" class="divStyle">
        DIV 3
      </div>
    </div>
  </div>
  <script type="text/javascript">
    var divElements = document.getElementsByTagName('div');
    for (var i = 0; i < divElements.length; i++)
    {
      divElements[i].addEventListener("click", clickHandler, true);
    }
    function clickHandler(event)
    {
      event.stopPropagation();
      alert(this.getAttribute("id") + " click event handled");
    }
  </script>
</body>
</html>
```

Using addEventListener() method with last argument set to true is the only way to enable event capturing. If the third parameter (phase) is set to true event capturing is enabled and if it is set to false event bubbling is enabled. If you want both even bubbling and capturing to be enabled, then assign handlers 2 times, once with the phase parameter set to false and once

with the phase parameter set to true as shown below.

```
<html>
<head>
  <style type="text/css">
    .divStyle
    {
      display: table-cell;
      border: 5px solid black;
      padding: 20px;
      text-align: center;
    }
  </style>
</head>
<body>
  <div id="DIV1" class="divStyle">
    DIV 1
    <div id="DIV2" class="divStyle">
      DIV 2
      <div id="DIV3" class="divStyle">
        DIV 3
      </div>
    </div>
  </div>
  <script type="text/javascript">
    var divElements = document.getElementsByTagName('div');

    for (var i = 0; i < divElements.length; i++)
    {
      divElements[i].addEventListener("click", clickHandler, false);
      divElements[i].addEventListener("click", clickHandler, true);
    }

    function clickHandler()
    {
      alert(this.getAttribute("id") + " click event handled");
    }
  </script>
</body>
</html>
```

With both event capturing and bubbling enabled, events are first captured from the outermost element to the innermost element and then bubbles up from the innermost to the

outermost element.

Preventing Browser Default Action

First let's look at some of the browser default actions. For example,

1. When you click on a link, the browser navigates to the page specified in the link
2. When you right click on a web page, the browser displays the context menu

In some situations you may want to **prevent these default actions of the browser**. For example some of the websites prevent you from right clicking on the page. Disabling right click is annoying users. Many people say they disabled right click for security, because they do not want their content to be copied. But if you disable JavaScript in the browser, you will still be able to right click and copy the content. So you are achieving nothing by disabling right click.

Having said that, now let us see how to prevent the context menu from appearing when you right click on the web page. There are 2 ways you can do this.

Using oncontextmenu attribute of the body element to disable right click

```
<html>
  <head>
  </head>
  <body oncontextmenu="return false">
    <h1>On this page right click is disabled</h1>
  </body>
</html>
```

Using the event object to disable right click

IE 8 and earlier versions

```
event.returnValue = false;
```

IE 9 & later versions and all other browsers

```
event.preventDefault();
```

```
<h1>On this page right click is disabled</h1>
<script type="text/javascript">
  document.oncontextmenu = disableRightClick;

  function disableRightClick(event)
  {
```

```
event = event || window.event;
if (event.preventDefault)
{
    event.preventDefault();
}
else
{
    event.returnValue = false
}
}
</script>
```

When you click on a link, **how to prevent the browser from navigating to the page specified in the link**

```
<a href="http://pragimtech.com" onclick="return false">
```

Clicking on the link will not take you to PragimTech

```
</a>
```

OR

```
<a href="http://pragimtech.com" onclick="preventLinkNavigation(event)">
```

Clicking on the link will not take you to PragimTech

```
</a>
```

```
<script type="text/javascript">
function preventLinkNavigation(event)
{
    event = event || window.event;
    if (event.preventDefault)
    {
        event.preventDefault();
    }
    else
    {
        event.returnValue = false
    }
}
</script>
```

JavaScript to detect which mouse button is clicked

Depending on the browser, **event.button** or **event.which** properties of the event object are used to determine which mouse button is clicked.

IE 8 & earlier versions use event.button property

Property	Code	Button
event.button	1	Left
	4	Middle
	2	Right

IE 9 & later versions and most other W3C compliant browsers use event.which property

Property	Code	Button
event.which	1	Left
	2	Middle
	3	Right

Depending on the browser used, the following code returns **left, middle and right click codes**.

```
<input type="button" value="Click Me" onmouseup="getMouseClickCode(event)" />
<input type="button" value="Clear" onclick="clearText()" />
<br />
<br />
<textarea id="txtArea" rows="3" cols="10"></textarea>
```

```
<script type="text/javascript">
function clearText()
{
    document.getElementById("txtArea").value = "";
}
function getMouseClickCode(event)
{
    if (event.which)
    {
        document.getElementById("txtArea").value += "event.which = " +
            event.which + "\r\n";
    }
}
```

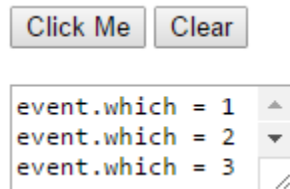
```

else
{
    document.getElementById("txtArea").value += "event.button = " +
        event.button + "\r\n";
}
}

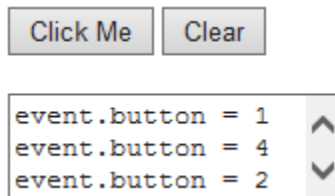
document.oncontextmenu = disableRightClick;
function disableRightClick(event)
{
    event = event || window.event;
    if (event.preventDefault)
    {
        event.preventDefault();
    }
    else
    {
        event.returnValue = false
    }
}
</script>

```

When the above script is tested with Google chrome or IE 9 (or later version), we get the following output. Notice that these browsers support **event.which** property.



When the above script is tested with IE 8 (or earlier version), we get the following output. Notice that IE & earlier versions support **event.button** property.



The following JavaScript **code detects which mouse button is clicked**. It works in all versions of IE and most other W3C complaint browsers.

```

<script type="text/javascript">
    function whichMouseButtonClicked(event)

```



```

{
    var whichButton;
    if (event.which)
    {
        switch (event.which)
        {
            case 1:
                whichButton = "Left Button Clicked";
                break;
            case 2:
                whichButton = "Middle Button Clicked";
                break;
            case 3:
                whichButton = "Right Button Clicked";
                break;
            default:
                whichButton = "Invalid Button Clicked";
                break;
        }
    }
    else
    {
        switch (event.button)
        {
            case 1:
                whichButton = "Left Button Clicked";
                break;
            case 4:
                whichButton = "Middle Button Clicked";
                break;
            case 2:
                whichButton = "Right Button Clicked";
                break;
            default:
                whichButton = "Invalid Button Clicked";
                break;
        }
    }

    alert(whichButton);
}

```

```

document.oncontextmenu = disableRightClick;

```

```

function disableRightClick(event)
{
    event = event || window.event;

    if (event.preventDefault)
    {
        event.preventDefault();
    }
    else
    {
        event.returnValue = false
    }
}
</script>
<button onmouseup="whichMouseButtonClicked(event)">Click Me</button>

```

JavaScript Mouse Events

mouseover	Occurs when the mouse pointer is moved over an element
mouseout	Occurs when the mouse pointer is moved out of an element
mousemove	Occurs when the mouse pointer is moving while it is over an element
mouseup	Occurs when the mouse button is released over an element
mousedown	Occurs when the mouse button is pressed over an element
click	Occurs when the mouse button is clicked. mousedown, mouseup & click events occur in sequence
dblclick	Occurs when the mouse button is double-clicked. mousedown, mouseup, mousedown, mouseup, click & dblclick events occur in sequence
contextmenu	Occurs when the mouse right button is clicked. mousedown, mouseup & contextmenu events occur in sequence

Here is an example which logs the mouse events to textarea element as they occur.

```
mouseover
mousedown
mouseup
click
mousedown
mouseup
contextmenu
mouseout
```

```
<input type="button" value="Single, Double or Right Click" onclick="logEvent(event)"
onmousedown="logEvent(event)" onmouseup="logEvent(event)"
onmouseover="logEvent(event)" onmouseout="logEvent(event)" ondblclick="logEvent(event)"
oncontextmenu="logEvent(event)" />
```

```
<input type="button" value="Clear" onclick="clearText()"/>
<br /><br />
<textarea id="txtArea" rows="10" cols="20"></textarea>
<script type="text/javascript">
    function logEvent(event)
    {
        event = event || window.event;
        document.getElementById("txtArea").value += event.type + "\r\n";
    }

    function clearText()
    {
        document.getElementById("txtArea").value = "";
    }
</script>
```

JavaScript Popup Window

To open a popup window, use **window.open()** method. All the parameters are optional.
 window.open(URL, name, features, replace)

URL	URL of the page to open. If URL is not specified, a new window with about:blank is opened	
name	Specifies the target attribute or the name of the window.	
	<u>_blank</u>	URL is loaded into a new window. Default value.
	<u>_parent</u>	URL is loaded into the parent frame
	<u>_self</u>	URL replaces the current page
	<u>_top</u>	URL replaces any framesets that may be loaded
	Name	name of the window
features	Must be a comma-separated list. Some browsers does not support all features.	
	menubar (yes/no)	Shows or hides the browser menu
	toolbar (yes/no)	Shows or hides the navigation bar
	location (yes/no)	Shows or hides the address field
	status (yes/no)	Shows or hides the status bar
	resizable (yes/no)	Whether or not the window is resizable
	scrollbars (yes/no)	Whether or not to display scroll bars
	top(pixels)	The top position of the window
	left(pixels)	The left position of the window
	height (pixels)	The height of the new window
	width (pixels)	The width of the new window
replace	Specifies whether the URL creates a new entry or replaces the current entry in the browser history. Works only if the url is loaded into the same window	
	true	URL replaces the current document in the history list
	false	URL creates a new entry in the browser history list

No parameters are passed to window.open() method. Since URL is not specified, a new window with about:blank will be opened.

```
<input type="button" value="Open popup" onclick="window.open()" />
```

Most modern browsers open new tabs instead of separate windows. If you want to open the popup in a new window, one workaround is to specify the URL, name, and features(height and width) parameters. This may not work in all browsers and it also depends on user's browser preferences.

```
<input type="button" value="Open popup" onclick="window.open('http://google.com', '_blank', 'height=200,width=200')" />
```

When name parameter is set to `_self`, the new window replaces the current window

```
<input type="button" value="Open popup"
      onclick="window.open('http://google.com', '_self')"/>
```

Specify where you want the new popup window to be positioned using top and left features.

```
<input type="button" value="Open popup" onclick="window.open('http://google.com', 'My
Window', 'height=300,width=300, top=400, left=400')"/>
```

Disable scrollbars and resizing. Works in IE but not in Chrome.

```
<input type="button" value="Open popup" onclick="window.open('http://google.com', 'My
Window', 'height=300,width=300, scrollbars=no, resizable=no')"/>
```

To close pouup use `window.close()` method.

```
<input type="button" value="Open popup" onclick="openPopup()" />
<input type="button" value="Close popup" onclick="closePopup()" />
<script type="text/javascript">
    var popup;
    function openPopup()
    {
        popup = window.open("http://google.com", "My Window", "height=300,width=300")
    }

    function closePopup()
    {
        popup.close();
    }
</script>
```

Regular Expression In JavaScript

What is a Regular Expression

A regular expression is a sequence of characters that forms a search pattern.

Let us understand the **use of regular expressions with an example**. The following strings contain words and numbers. From the string we want to extract all the numbers. Basically the program should work with any string.

Mark-9 Tim-890 Sam-10 Sara-9902

Result : 9, 890, 10, 9902

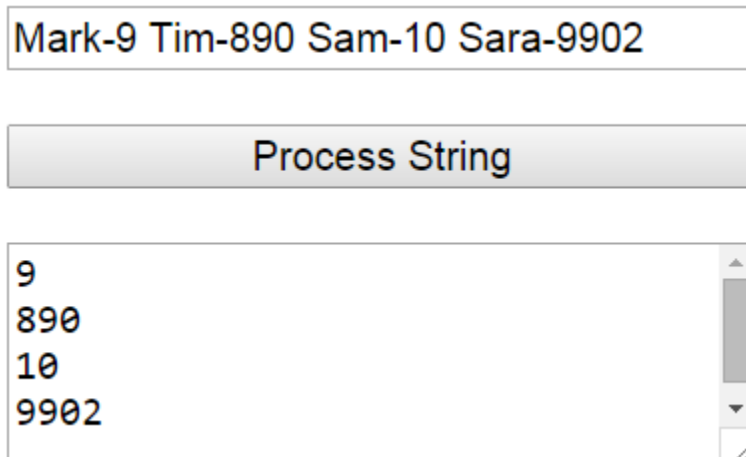
908ABC12XYZ34

Result : 908, 12, 34

\$1 \$2 \$901 ABC(100)

Result : 1, 2, 901, 100

Implement a web page as shown below.



Mark-9 Tim-890 Sam-10 Sara-9902

Process String

9
890
10
9902

Here is what we want the page to do

1. User enters the string in the first textbox
2. When "Process String" button is clicked, the numbers should be extracted from the string and displayed in the text area element.

It will be very complex and error prone if we have to achieve this without using regular expressions.

```
<input type="text" id="txtBox" style="width:250px" />
<br /><br />
<input type="button" value="Process String" onclick="processString()"
      style="width:250px" />
<br /><br />
<textarea id="txtArea" rows="4" cols="30"></textarea>
<script type="text/javascript">
```

```

function processString()
{
    // Clear the textarea element
    document.getElementById("txtArea").value = "";
    // Retrieve the user input from the textbox
    var inputString = document.getElementById("txtBox").value;
    // Regular expression should be in 2 forward slashes //
    // Letter g at the end of the regular expression performs a global match
    // match() method returns all substrings that match the given regular expression
    var result = inputString.match(/\d+/g);
    if (result != null)
    {
        // Add the retrieved numbers to the textarea element
        for (var i = 0; i < result.length; i++)
        {
            document.getElementById("txtArea").value += result[i] + "\r\n";
        }
    }
}
</script>

```

Tools for writing Regular Expressions

Basics of Regular Expressions :

Find the word expression in a given string. This will also match with the word expressions.
expression

To find the word "expression" as a whole word, include \b on either sides of the word expression
\bexpression\b

\d indicates to find a digit. To find a 5 digit number we could use the following
\b\d\d\d\d\d\b

We can avoid the repetition of \d by using curly braces as shown below. \d{5} means repeat \d 5 times.
\b\d{5}\b

The above example can also be rewritten as shown below.
\b[0-9]{5}\b

Find all the words with exactly 5 letters
\b[a-zA-Z]{5}\b

Brackets are used to find a range of characters

[a-z] - Find any of the characters between the brackets

[0-9] - Find any of the digits between the brackets. This is equivalent to \d

(a|b) - Find any of the characters a or b

The page at the following link explains the basics of regular expressions.

https://developer.mozilla.org/en/docs/Web/JavaScript/Guide/Regular_Expressions

Expresso is one of the free tools available. Here is the link to download.

<http://www.ultrapico.com/ExpressoDownload.htm>

Regular Expression Library

<http://regexlib.com>

JavaScript String and Regular Expressions

In JavaScript regular expressions can be used with the following string methods.

1. match()
2. replace()
3. split()
4. search()

Along with regular expressions you can use **modifiers** to specify the kind of search you want to perform.

g	Global search
i	Case-insensitive search
m	Multiline search

Let us look at some examples of using the above methods with regular expressions

Using regular expression with JavaScript string object's match() method. All the numbers in the string will be returned. The letter **g** at the end of the regular expression performs a global match. If the letter **g** is omitted we will only get the first match.

```
var string = "Tom contact number is 1011011010. His age is 35.";
string += "Mark contact number is 8912398912. His age is 45";
```

```
document.write(string.match(/\d+/g));
```

Output : 1011011010,35,8912398912,45

Using regular expression with JavaScript string object's replace() method. All the numbers in the

string will be replaced with XXX.

```
var string = "Tom contact number is 1011011010. His age is 35.";
string += "Mark contact number is 8912398912. His age is 45";
```

```
document.write(string.replace(/\d+/g, "XXX"));
```

Output : Tom contact number is XXX. His age is XXX.Mark contact number is XXX. His age is XXX

Using regular expression with JavaScript string object's split() method. split() method breaks a string into an array of substrings. The following example breaks the string into an array of substrings wherever it finds a number.

```
var string = "Tom contact number is 1011011010. His age is 35.";
string += "Mark contact number is 8912398912. His age is 45";
```

```
document.write(string.split(/\d+/))
```

Output : Tom contact number is ,. His age is ,.Mark contact number is ,. His age is ,

Using regular expression with JavaScript string object's search() method. search() method returns the index of the match, or -1 if the search item is not found. search() method returns the index of the first matching item only. The following example returns the index of the first occurrence of a number in the string.

```
var string = "Tom contact number is 1011011010. His age is 35.";
string += "Mark contact number is 8912398912. His age is 45";
```

```
document.write(string.search(/\d+/))
```

Output : 22

Global case insensitive match using a regular expression

```
var string = "TOM contact number is 1011011010. tom is 35";
document.write(string.match(/tom/gi));
```

Output : TOM,tom

JavaScript RegExp Object

There are 2 ways to **create a regular expression in JavaScript**

Using a regular expression literal

```
var regex = /\d+/g;
```

Regular expressions created using regular expression literals are automatically compiled when the script is loaded. So if you know that the regular expression is not going to change then use this approach for better performance.

The following example replaces all the numbers with XXX.

```
var string = "Tom contact number is 1011011010. His age is 35.";
string += "Mark contact number is 8912398912. His age is 45";
```

```
document.write(string.replace(/\d+/g, "XXX"));
```

Output : Tom contact number is XXX. His age is XXX.Mark contact number is XXX. His age is XXX

Using the constructor function of the RegExp object

```
var regexp = new RegExp("\\d+", "g");
```

Regular expressions created using the constructor function are compiled at runtime. Use this approach when the regular expression is expected to change.

Please note : Since the first argument of the RegExp constructor is a string, you have to escape the backslash.

The following example replaces all the numbers with XXX.

```
var string = "Tom contact number is 1011011010. His age is 35.";
string += "Mark contact number is 8912398912. His age is 45";
```

```
var regexp = new RegExp("\\d+", "g");
```

```
document.write(string.replace(regexp, "XXX"));
```

Output : Tom contact number is XXX. His age is XXX.Mark contact number is XXX. His age is XXX

Commonly used RegExp object properties

global	returns true if the global modifier (g) is set, otherwise false
ignoreCase	returns true if the case-insensitive modifier (i) is set, otherwise false
multiline	returns true if the multi-line modifier (m) is set, otherwise false
source	Returns the text of the regular expression

Example :

```
var regexp = new RegExp("\\d+", "gi");

document.write("g = " + regexp.global + "<br/>");
document.write("i = " + regexp.ignoreCase + "<br/>");
document.write("m = " + regexp.multiline + "<br/>");
document.write("source = " + regexp.source + "<br/>");
```

Output :

```
g = true
i = true
m = false
source = \d+
```

Commonly used RegExp object methods

exec()	Tests for a match in a given string and returns the first match if found otherwise null.
test()	Tests for a match in a given string and returns true or false
toString()	Returns the string value of the regular expression

exec() method returns the first match

```
var string = "Tom contact number is 1011011010. His age is 35.";
string += "Mark contact number is 8912398912. His age is 45";
```

```
var regexp = new RegExp("\\d+");
document.write(regexp.exec(string));
```

Output : 1011011010

To get all the matches call `.exec()` method repeatedly with the `g` flag as shown below

```
var string = "Tom contact number is 1011011010. His age is 35.";
string += "Mark contact number is 8912398912. His age is 45";
```

```
var regexp = new RegExp("\\d+", "g");
var result;
while ((result = regexp.exec(string)) != null)
{
    document.write(result[0] + "<br/>");
}
```

The following example calls **test()** method of the **RegExp** object to check if the string contains numbers.

```
var string = "Tom contact number is 1011011010. His age is 35.";
string += "Mark contact number is 8912398912. His age is 45";
```

```
var regexp = new RegExp("\\d+", "g");
document.write("String contain numbers - " + regexp.test(string))
```

Output : String contain numbers - true

You can also call `exec()` and `test()` methods of the **RegExp** object as shown below

```
var string = "Tom contact number is 1011011010. His age is 35.";
string += "Mark contact number is 8912398912. His age is 45";
```

```
var regexp = /\d+/g;
document.write("String contain numbers - " + regexp.test(string))
```

OR

```
var string = "Tom contact number is 1011011010. His age is 35.";
string += "Mark contact number is 8912398912. His age is 45";
document.write("String contain numbers - " + /\d+/g.test(string))
```

Client Side Validation using Regular Expression

On most of the websites it is common to check if the format of the email is valid. Using regular expressions we can achieve this very easily.

Invalid email :

Email :

Valid email :

Email :

Here is an example:

```
Email : <input type="text" id="txtEmail" onkeyup="validateEmail()" />
<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\d-0-9]+\.)+[a-zA-Z]{2,}))$/;

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

```
    }  
    else  
    {  
        emailTextBox.style.backgroundColor = "red";  
    }  
}  
</script>
```

JavaScript Minification

What is JavaScript Minification

JavaScript Minification is the process of reducing the script file size by removing comments, extra whitespaces and new line characters that are not needed for executing JavaScript code. JavaScript Minification may reduce the size of the file by 30 to 90%. The Minification process will not change its original functionality.

What are the benefits of JavaScript Minification

The JavaScript files need to be downloaded to the client machine before the browser can execute your JavaScript code. Since JavaScript Minification reduces the size of the file we have the following benefits.

1. Reduced download time
2. Less bandwidth consumption of your website
3. Reduced JavaScript execution time as all the comments, extra whitespaces and new line characters are removed from the minified version
4. Multiple JavaScript files can be compressed into one minified JavaScript file. This means there are now reduced number of HTTP requests to your server, which in turn reduces the load on the server and allows more users to access your website.

What are the disadvantages of JavaScript Minification

Readability is lost and debugging can be difficult as comments, extra whitespaces and new line characters are removed. However, if there is a production issue, use the non-minified version of the script file for debugging. So in production environment use minified version for performance and in development environment use non-minified version for readability and debugging.

JavaScript minification tools

JSMin : <http://www.crockford.com/javascript/jsmin.html>

Closure Compiler : <https://developers.google.com/closure/compiler/>

YUI Compressor : <http://yui.github.io/yuicompressor/>

There are also several websites that provide online JavaScript minification. The following is one such website

<http://marijnhaberbeke.nl/uglifyjs>

Contents of the script file :

```
function getBrowserInfo()
{
    var ua = navigator.userAgent, tem,
    M = ua.match(/(opera|chrome|safari|firefox|msie|trident(?=V))\V?\s*(\d+)/i) || [];
    if (/trident/i.test(M[1])) {
        tem = /\brv[ :]+\d+/g.exec(ua) || [];
        return 'IE ' + (tem[1] || '');
    }
    if (M[1] === 'Chrome') {
        tem = ua.match(/\bOPRV(\d+)/)
        if (tem != null) return 'Opera ' + tem[1];
    }
    M = M[2] ? [M[1], M[2]] : [navigator.appName, navigator.appVersion, '-?'];
    if ((tem = ua.match(/version(\d+)/i)) != null) M.splice(1, 1, tem[1]);
    return M.join(' ');
}
```

```
function getCurrentMonth()
{
    var today = new Date();
    var monthNumber = today.getMonth();
```

```
    var monthNames = ["January",
        "February",
        "March",
        "April",
        "May",
        "June",
        "July",
        "August",
        "September",
        "October",
        "November",
        "December"];
```

```
    return monthNames[monthNumber];  
}
```

```
function getCurrentWeekDay()  
{  
    var today = new Date();  
    var dayNumber = today.getDay();
```

```
    var weekDays = ["Sunday",  
                    "Monday",  
                    "Tuesday",  
                    "Wednesday",  
                    "Thursday",  
                    "Friday",  
                    "Saturday"];
```

```
    return weekDays[dayNumber];
```

```
}
```