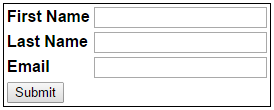
**1. Introduction to Javascript**

**In this video we will discuss**  
**1.** The use of JavaScript in a web application.   
**2.** Advantages of using JavaScript   
  
   
  
**Example :** The following web form captures user First Name, Last Name & Email.   
   
  
When we click the **Submit**button, we want to save the data to the following SQL Server **Users**table   
why use JavaScript   
  
**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>

**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.   
   
  
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 broswer.   
  
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 **OnClientClick="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"

    OnClientClick="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.

**2. Client Side v/s Server Side Validation.**

In this video we will discuss, **why we need both client side and server side validation**. This is a common interview question as well. This is continuation to [Part 1](http://csharp-video-tutorials.blogspot.com/2014/10/what-is-use-of-javascript-in-aspnet.html). Please watch [Part 1](http://csharp-video-tutorials.blogspot.com/2014/10/what-is-use-of-javascript-in-aspnet.html) before proceeding.   
  
   
  
**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 Customise 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 that we worked with in [Part 1](http://csharp-video-tutorials.blogspot.com/2014/10/what-is-use-of-javascript-in-aspnet.html), 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)

{

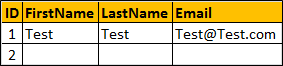
    //if (ValidateForm())

    //{

        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.   
   
  
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.

**3. Disadvantages of Javascript.**

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 that we worked with in [Part 1](http://csharp-video-tutorials.blogspot.com/2014/10/what-is-use-of-javascript-in-aspnet.html), 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()

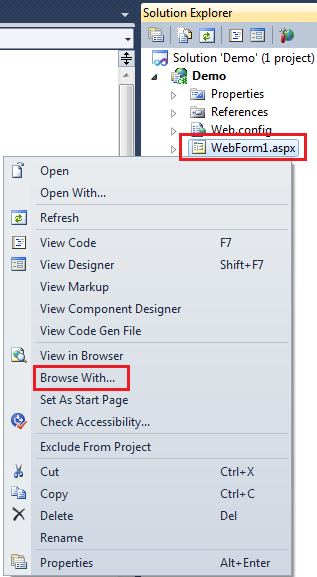
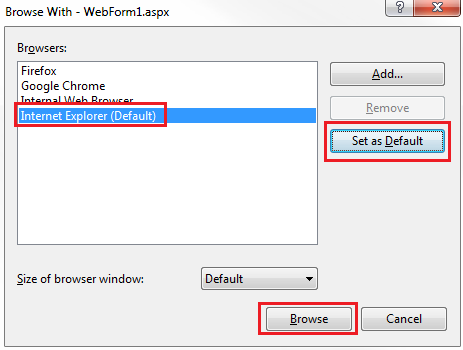
{

alert('You selected ' + document.getElementById('ddlGender').value);

}

</script>

4. How to Debug 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.

5. Tools required to learn Javascript

**In this video we will discuss**  
**1.** Tools required to learn JavaScript  
**2.** Websites that help write and test JavaScript  
**3.** Advantage of using Visual Studio to learn JavaScript   
  
   
  
**Tools required to learn JavaScript**  
To learn JavaScript all you need is a text editor like NOTEPAD and a broswer. 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. We will discuss JavaScript functions in detail in a later video session. 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 favourite 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](http://jsbin.com/). This website also provides intellisense 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 intellisense and autocomplete features which can reduce typos and increase developer productivity. It also provides rich JavaScript debugging support.

6. Inline vs. External javascript

**In this video we will discuss**  
**1.** Different places where JavaScript can be present  
**2.** Advantages of external JavaScript over inline JavaScript   
  
   
  
**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.

7. Where should the script tag be placed?

In this video we will discuss, **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>

<formid="form1"runat="server">

</form>

<scripttype="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>

<formid="form1"runat="server">

<asp:TextBoxID="TextBox1"runat="server"></asp:TextBox>

</form>

<scripttype="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>

<scripttype="text/javascript">

document**.**getElementById**(**"TextBox1"**).**style**.**backgroundColor="red"**;**

</script>

</head>

<body>

<formid="form1"runat="server">

<asp:TextBoxID="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 [Part 6](http://csharp-video-tutorials.blogspot.com/2014/11/inline-vs-external-javascript.html), we discussed that, 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.

8. Javascript Basics

**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;

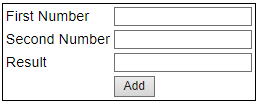
    var c = a - b;

    alert(c);

</script>

Output : 40

9. Converting strings to Numbers

In this video we will discuss **different methods that are available in JavaScript to convert strings to numbers**. Let's design the web form as shown below.    
   
  
   
  
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>

            <input type="button" value="Add" id="btnAdd"/>

        </td>

    </tr>

</table>

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

<script type="text/javascript">

    function addNumbers()

    {

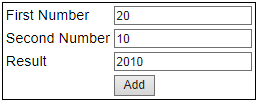
        var firstNumber = document.getElementById("txtFirstNumber").value;

        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.  
  
<inputtype="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 buton. 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.    
   
  
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()

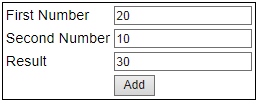
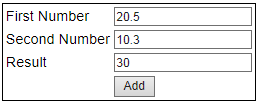
{

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

function addNumbers()

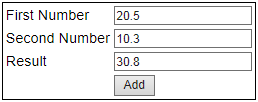
{

    var firstNumber = parseFloat(document.getElementById("txtFirstNumber").value);

    var secondNumber = parseFloat(document.getElementById("txtSecondNumber").value);

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

}

   
  
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;

}

10.String in Javascript:-

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

varstring1="string in double quotes"

varstring2='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**

varstring1="Hello"

varstring2="JavaScript"

varresult=string1+" "+string2**;**

alert**(**result**);**

Output : Hello JavaScript  
  
**Example : Concatenating strings using concat() method**

varstring1="Hello"

varstring2="JavaScript"

varresult=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 :

varmyString="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 :

varmyString='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 :

varupperCaseString="JavaScript"**;**

alert**(**upperCaseString**.**toUpperCase**());**

Output : JAVASCRIPT  
  
**Converting a string to lowercase :** Use toLowerCase() method  
  
Example :

varlowerCaseString="JavaScript"**;**

alert**(**lowerCaseString**.**toLowerCase**());**

Output : javascript  
  
**Length of string javascript :** Use length property  
  
Example : alert**(**"JavaScript"**.**length**);**  
  
Output : 10  
  
Example :

varmyString="Hello JavaScript"**;**

alert**(**myString**.**length**);**

Output : 16  
  
**Remove whitespace from both ends of a string** : Use trim() method  
  
Example :

varstring1=" AB "**;**

varstring2=" CD "**;**

varresult=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

varmyString="Hello JavaScript"**;**

varresult=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.

varmyString="A Blue bottle with a blue liquid is on a blue table"**;**

varresult=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.

varmyString="A Blue bottle with a blue liquid is on a blue table"**;**

varresult=myString**.**replace**(**/blue/gi,"green"**);**

alert**(**result**);**

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

11.SubString In Javascript:-

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**

varstr="JavaScript Tutorial"**;**

varresult=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**

varstr="JavaScript Tutorial"**;**

varresult=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**

varstr="JavaScript Tutorial"**;**

varresult=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**

varstr="JavaScript Tutorial"**;**

varresult=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**

varstr="JavaScript Tutorial"**;**

varresult=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**

varstr="JavaScript Tutorial"**;**

varresult=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

varstr="pragim@pragimtech.com"**;**

varresult=str**.**indexOf**(**"@"**);**

alert**(**result**);**

Output : 6  
  
In our next video, we will discuss a simple **real time example of where we can use indexOf() and substring() methods**

 12. SubString Examples

In this video we will discuss 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.   
   
  
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 attriibute of the button to call the JavaScript function**

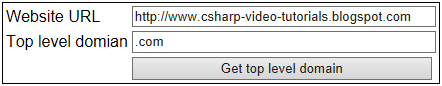
<input type="button" value="Get email & domain parts" style="width:250px"

    onclick="getEmailandDomainParts()"/>

In [Part 11](http://csharp-video-tutorials.blogspot.com/2014/11/substrings-in-javascript.html) of [JavaScript Tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhUA99nOsJkKXBqokT3MBK0b) we discussed **indexOf**() function. **lastIndexOf**() is also very useful function for manipulating strings.  
  
**lastIndexOf() method returns the position of the last occurrence of a specified value in a string.** Since it's 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 domian" textbox.   
   
  
**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 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 attriibute of the button to call the JavaScript function

<inputtype="button"value="Get top level domain"style="width: 300px"

onclick="getDomainName()"/>

13. Conditional statement in Javascript

**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");

}

14. Switch statement in Javascript

**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;

}

15. Ternary Operator in JavaScript

**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);

16. Loops in JavaScript:-

Here are the basic **loops in JavaScript**  
while  
do...while  
for   
  
   
  
In this video we will discuss **while loop in JavaScript with an example**. 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.  
  
**JavasSript 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/>");

}

17. Do.. While loop in Javascript

While loop:

* 1. While loops 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 represent a menu to the users

**What is the difference between While and do while loop 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 executes at least once, whereas while loop is not

var userchoice = "";

do{

var targetNumber = Number(prompt("please enter targetn 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")

18. for loop in Javascript:-

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 intialization (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 it's 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 it's 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;

}

19. 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 busing 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 option 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 that 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

20. JavaScript Array Push and Pop Method.

In this video we will discuss **push() and pop() methods in JavaScript**. Along the way we will also discuss shift() and unshift() methods. Let us understand these methods with examples.   
  
   
  
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;

}

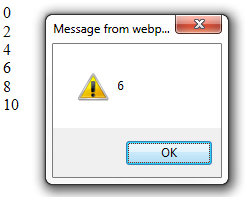
for (var i = 0; i <= 5; i++)

{

    document.write(myArray[i] + "<br/>");

}

alert(myArray.length);

**Output :**   
   
  
**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 exampe 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);

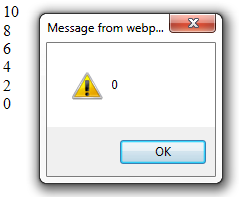
for (var i = 0; i <= 5; i++)

{

    document.write(myArray.pop() + "<br/>");

}

alert(myArray.length);

**Output :**   
   
  
**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 :**   
javascript array push example   
  
**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/>");

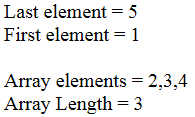
// removes the first element i.e 1 from the array

var firstElement = myArray.shift();

document.write("First element = " + firstElement + "<br/><br/>");

document.write("Array elements = " + myArray + "<br/>");

document.write("Array Length = " + myArray.length);

**Output :**   


21. JavaScript 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  
  
We discusssed push(), pop(), shift() and unshift() methods in [Part 20](http://csharp-video-tutorials.blogspot.com/2014/12/javascript-array-push-and-pop-methods.html). In this video we will discuss   
sort  
reverse  
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 a from b 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

22. JavaScript Array filter Method.

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

23. Creating Two Dimensional Array in JavaScript

**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++)

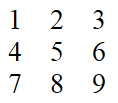
    {

        document.write(myArray1[i][j] + "&emsp;");

    }

    document.write("<br/>");

}

**Output :**   
   
  
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++)

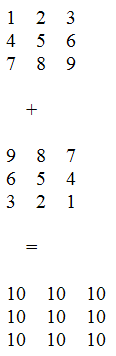
    {

        document.write(myArray1[i][j] + "&emsp;");

    }

    document.write("<br/>");

}

**Performing addition between 2 two dimensional arrays in JavaScript**  
  
**Example :**  
a) The first 3 x 3 array should contain numbers from 1 to 9  
b) The second 3 x 3 array should contain numbers from 9 to 1  
c) 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.   
  
 

// 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++)

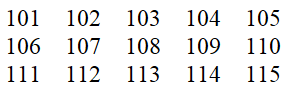
    {

        document.write(myArray1[i][j] + "&emsp;");

    }

    document.write("<br/>");

}

**Output :**   


24. Function 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);

25. Different ways of Defining Function in JavaScript

In JavaScript, there are several different ways of defining functions.   
  
   
  
**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 with in 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.  
  
**Self invoking 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

### 26. 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 it's 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

 27. 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 it's 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 additonal parentheses.

var result = addNumbers(10, 20)();

document.write(result);

28. JavaScript Closures Examples

In this video we will discuss 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 it's 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 I 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());" />

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/>");

    }

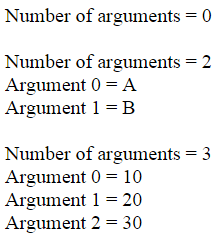
    document.write("<br/>");

}

printArguments();

printArguments("A", "B");

printArguments(10, 20, 30);

**Output :**   
   
  
**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];

    }

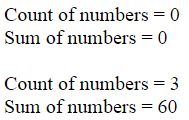
    document.write("Sum of numbers = " + sum);

    document.write("<br/><br/>");

}

addNumbers();

addNumbers(10, 20, 30);

**Output :**    
   
  
**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

29. JavaScript Object Argument

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/>");

    }

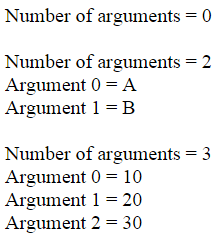
    document.write("<br/>");

}

printArguments();

printArguments("A", "B");

printArguments(10, 20, 30);

**Output :**   
   
  
**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];

    }

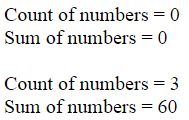
    document.write("Sum of numbers = " + sum);

    document.write("<br/><br/>");

}

addNumbers();

addNumbers(10, 20, 30);

**Output :**    
   
  
**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

30. Recursive Function in 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

31. 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)

{

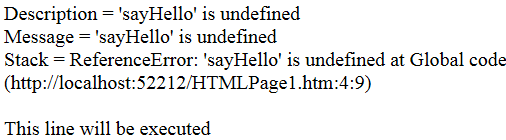
    document.write("Description = " + e.description + "<br/>");

    document.write("Message = " + e.message + "<br/>");

    document.write("Stack = " + e.stack + "<br/><br/>");

}

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

**Output :**   
   
  
**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/>");

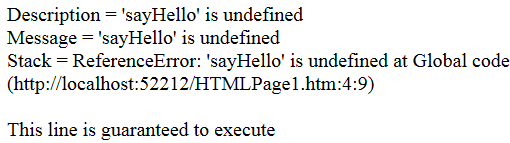
}

finally

{

    document.write("This line is guaranteed to execute");

}

**Output :**   
   
  
**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();

### 32. JavaScript 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);

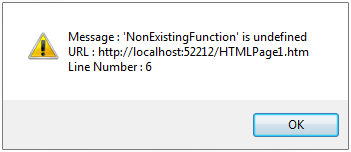
    // Return true to supress the browser error messages

    // (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()

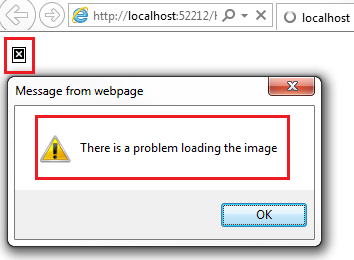
    {

        alert("There is a problem loading the image");

    }

</script>

<img src="NonExistingImage.jpg" onerror="imageErrorHandler()" />

**Output :**   


33. Working With Dates With 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 I have **(UTC) Dublin, Edinburgh, Lisbon, London** time zone selected on my 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()));

34. JavaScript Timing Event

**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);

    }

    function getCurrentDateTime()

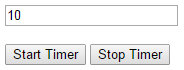
    {

        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()

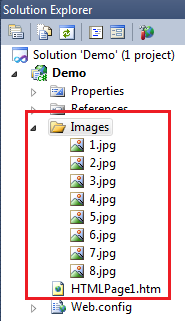
    {

        clearTimeout(intervalId);

    }

</script>

### 35. How to create image slideshow using JavaScript

In this video, we will discuss **creating a simple image slideshow using JavaScript**. We will be using **setInterval()**and **clearInterval()**JavaScript methods to achieve this. We discussed these functions in detail in [Part 34](http://csharp-video-tutorials.blogspot.com/2015/01/javascript-timing-events.html) of [JavaScript Tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhUA99nOsJkKXBqokT3MBK0b).   
  
   
  
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 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  
  
At the above location, on my machine I have 8 images. 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 8 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.  
<img id="image" src="/Images/1.jpg" style="width: 150px; height: 150px" />

<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.

36. Event in JavaScript

**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 this video we will discuss associating event handler methods to events using the attributes of HTML tags.   
  
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 colour 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 colour 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>

### 37. Assigning event handlers in JavaScript using DOM object property

In JavaScript there are several ways to associate an event handler to the event. In [Part 36](http://csharp-video-tutorials.blogspot.com/2015/01/events-in-javascript.html), we discussed, associating event handler methods to events using the attributes of HTML tags. In this video we will discuss using DOM object property to assign event handlers to events.   
  
   
  
**First let us understand, 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>

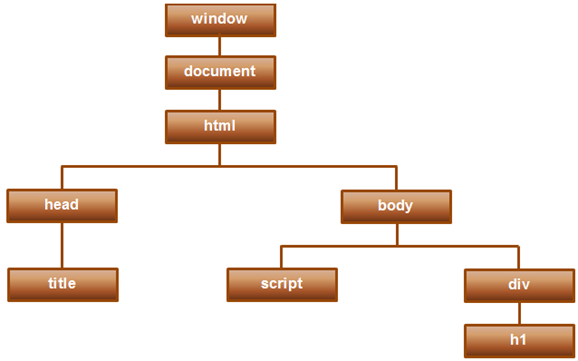
        <div>

            <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. We will discuss the DOM object in detail in a later video session.  
  
We will continue with the same examples that we worked with in [Part 36](http://csharp-video-tutorials.blogspot.com/2015/01/events-in-javascript.html). Notice that in this case, 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**() owerites **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>

### 38. addeventlistener and removeeventlistener in JavaScript

In this video we will discuss assigning 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.   
  
**Sytnax for assigning event handler using addEventListener() method**  
element.addEventListener(event, handler, phase)  
  
**Sytnax 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 & clickHandler3) 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.   
  
**Sytnax for assigning event handler using attachEvent() method**  
element.attachEvent( "on"+event, handler)  
  
**Sytnax 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()

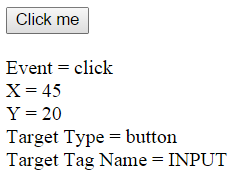
    {

        alert("Click Handler");

    }

</script>

39.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 occured  
3. Mouse Y coordinate when the event occured  
4. The control that raised the event  
5. The HTML tag name that raised the event  
  
   
  
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 occured. 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 = "

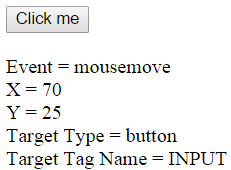
                           + sourceElement.type + "<br/>Target Tag Name = "

                           + 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.   
  
 

<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 = "

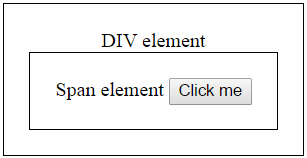
                            + sourceElement.tagName;

        document.getElementById("resultDiv").innerHTML = eventDetails;

    }

</script>

### 40. 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 <span> 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

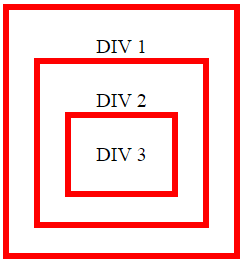
            <input type="button" value="Click me"/>

        </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 it's 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") + " backgound changed");

            }

        }

    </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") + " backgound changed");

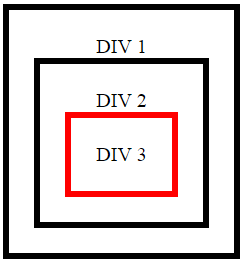
            }

        }

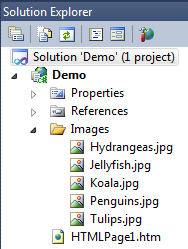
    </script>

</body>

</html>



### 41. Image gallery with thumbnails in JavaScript

In this video we will discuss creating **image gallery with thumbnails** in JavaScript. 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 thumnail, 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>

    <img id="mainImage" style="border:3px solid grey"

         src="/Images/Hydrangeas.jpg" height="500px" width="540x"/>

    <br />

    <div id="divId" onclick="changeImageOnClick(event)">

        <img class="imgStyle" src="/Images/Hydrangeas.jpg" />

        <img class="imgStyle" src="/Images/Jellyfish.jpg" />

        <img class="imgStyle" src="/Images/Koala.jpg" />

        <img class="imgStyle" src="/Images/Penguins.jpg" />

        <img class="imgStyle" src="/Images/Tulips.jpg" />

    </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.

42. JavaScript Event Capturing

**Event capturing is the opposite of event bubbling**. We discussed **event bubbling** in detail in [Part 40](http://csharp-video-tutorials.blogspot.com/2015/01/image-gallery-with-thumbnails-in.html) of [JavaScript tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhUA99nOsJkKXBqokT3MBK0b).   
  
   
  
With event bubbling the event bubbles up from the inner most element to the outer most element in the DOM hierarchy. With event capuring 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 thid 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.

43. Prevent Browser Default Action

In this video we will discuss **how to prevent 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

        {

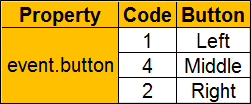
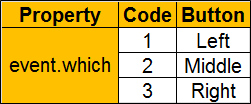
            event.returnValue = false

        }

    }

</script>

### 44. JavaScript to detect which mouse button is clicked

In this video, we will discuss **how to use JavaScript and 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**   
   
  
**IE 9 & later versions and most other W3C compliant browsers use event.which property**   
   
  
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

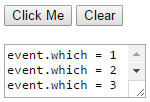
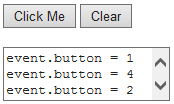
        {

            event.returnValue = false

        }

    }

</script>

When the above script is tested with Google chrome or IE 9 (or later version), I get the following output. Notice that these browsers support **event.which** property.    
   
  
When the above script is tested with IE 8 (or earlier version), I 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

        }

    }

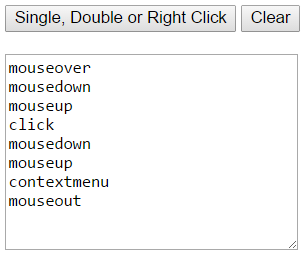
</script>

<button onmouseup="whichMouseButtonClicked(event)">Click Me</button>

45. JavaScript Mouse Event

In this video we will discuss the commonly used **JavaScript mouse events**. Most browsers support these 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.  
  
 

<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()

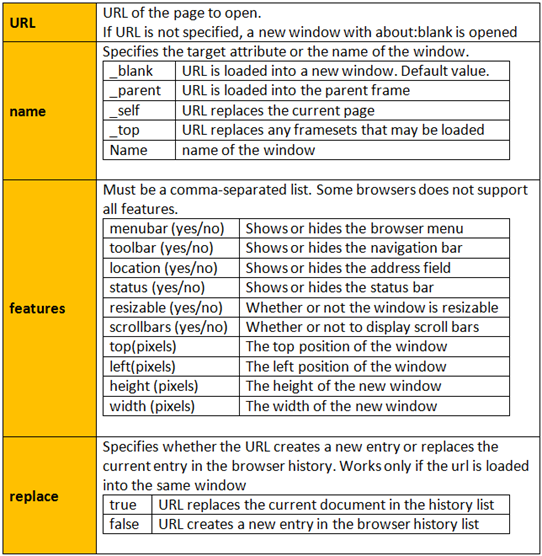
    {

        document.getElementById("txtArea").value = "";

    }

</script>

46. JavaScript Pop up window

In this video we will discuss, **how to open a popup window using JavaScript**.   
  
   
  
To open a popup window, use **window.open()** method. All the parameters are optional.  
window.open(URL, name, features, replace)   
  
   
  
**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 I have found 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()

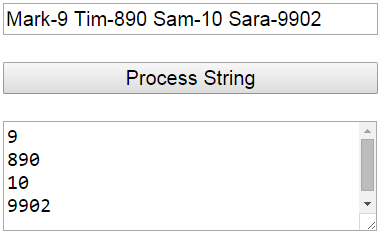
    {

        popup.close();

    }

</script>

47. Using 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. Bsically 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.   
   
  
**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 intput 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";

            }

        }

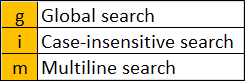
    }

</script>

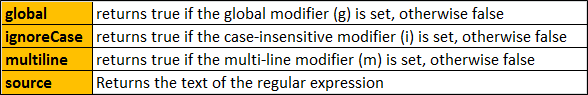
48. Tools for Writing Regular Expressions

In this video we will discuss the **basics of regular expressions and then look at some of the tools available to learn, write and test 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](http://regexlib.com/)

49. JavaScript String and Regular Expression.

In this video we will discuss **using regular expressions with some of the JavaScript string object methods**.   
  
   
  
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.   
   
  
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

50. Regular Expression Object.

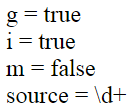
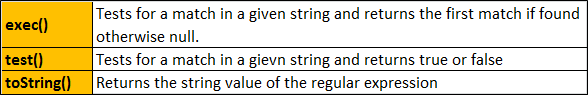
In this video we will discuss RegExp object in JavaScript.   
  
   
  
There are 2 ways to **create a regular expression in JavaScript**  
  
**Using a regular expression literal**  
var regex = /\d+/g;  
  
All the examples so far in this video series used regular expression literal to create a regular expression. 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**   
  
  
**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 :**   
   
  
**Commonly used RegExp object methods**   
  
  
**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))

51. Client Side Validation Using Regular Expression

In this video we will discuss, **how regular expressions can be used to perform client side validation**.   
  
   
  
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 :**   
javascript validate email regex   
  
**Valid email :**   
javascript validate email regular expression   
  
**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\-0-9]+\.)+[a-zA-Z]{2,}))$/;

        emailTextBox.style.color = "white";

        if (emailRegEx.test(email))

        {

            emailTextBox.style.backgroundColor = "green";

        }

        else

        {

            emailTextBox.style.backgroundColor = "red";

        }

    }

</script>

52.JavaScrip Magnification

**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://marijnhaverbeke.nl/uglifyjs>   
  
**Contents of the script file used in the demo**

function getBrowserInfo()

{

    var ua = navigator.userAgent, tem,

    M = ua.match(/(opera|chrome|safari|firefox|msie|trident(?=\/))\/?\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(/\bOPR\/(\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];

}

53. Object Oriented programming and JavaScript

**JavaScript is object oriented programming language. The following are the 4 pillars of any object oriented programming language**.  We will discuss examples of these in a later video session.  
1. Inheritance  
2. Encapsulation  
3. Abstraction  
4. Polymorphism   
  
   
  
In this video let's focus on **creating objects in JavaScript**. Objects in JavaScript can be broadly classified into 2 categories.  
1. Standard built-in objects  
2. Custom objects  
  
**Standard built-in objects :**So far in this video series, we have already seen many of the JavaScript standard built-in objects. **Examples**include **string, array, RegExp, Date**etc. In the example below we are creating Date object using the Date constructor function and then using it's **getFullYear()** method to get the year.  
  
var currentDate = new Date();  
document.write(currentDate.getFullYear());  
  
**Custom objects :** In C#, to create a custom object, we create a Custom class and then create an instance of a class. In JavaScript we don't have classes. Instead we use functions. In JavaScript there are two ways to create a custom object.  
1. Constructor function   
2. Literal notation  
  
**Creating an object in JavaScript using constructor function**

<script type="text/javascript">

    function Employee(firstName, lastName)

    {

        this.firstName = firstName;

        this.lastName = lastName;

        this.getFullName = function ()

        {

            return this.firstName + " " + this.lastName;

        }

    }

    var employee = new Employee("Pragim", "Tech");

    document.write("FirstName = " + employee.firstName + "<br/>");

    document.write("LastName = " + employee.lastName + "<br/>");

    document.write("FullName = " + employee.getFullName() + "<br/>");

</script>

**Creating an object in JavaScript using literal notation**

<script type="text/javascript">

    var employee =

    {

        firstName: "Pragim",

        lastName: "Tech",

        getFullName: function ()

        {

            return this.firstName + " " + this.lastName;

        }

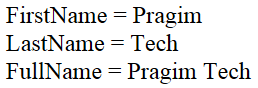
    }

    document.write("FirstName = " + employee.firstName + "<br/>");

    document.write("LastName = " + employee.lastName + "<br/>");

    document.write("FullName = " + employee.getFullName() + "<br/>");

</script>

Both the examples above produce the same output   
   
  
What is the **difference between creating an object using constructor function and literal notation**.   
1. In the constructor function the properties and their values separated using an equal-to sign(=) whereas in the literal version, they are separated using a colon (:)  
  
2. In constructor function at the end of each property you can have a semi-colons (;) whereas in the literal version properties must be separated with a comma (,)  
  
3. With literal notation you have already created an object, so to access firstName value you simply use employee.firstName. With the constructor function you have to first create an instance and then use the created instance and the property name separated by DOT as shown below.   
  
var employee = new Employee("Pragim", "Tech");  
employee.firstName

### 54. Object literal vs object constructor

In this video we will discuss the main **difference between objects created using object literal and constructor function** and when to use one over the other. In [Part 53](http://csharp-video-tutorials.blogspot.com/2015/01/javascript-and-object-oriented.html)of [JavaScript tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhUA99nOsJkKXBqokT3MBK0b) we discussed some of the syntactical differences.   
  
   
  
**Creating an object using object literal notation**

<script type="text/javascript">

    var employee =

    {

        name: "John"

    }

    // Create a new variable and assign the employee object

    var newEmployee = employee;

    // Change the name property of the employee object using the new variable

    newEmployee.name = "Mary";

    // Retrieve the name property from the original employee object

    // Notice that name is changed to Mary

    document.write(employee.name);

</script>

**Output :** Mary   
  
Objects created using object literals are singletons. This means when a change is made to the object, it affects that object across the entire script.   
  
**Creating an object using constructor function**

<script type="text/javascript">

    var emp = function ()

    {

        this.name = "John";

    }

    // Create an instance of employee

    // employee.name will return John

    var employee = new emp();

    // Create an other instance of employee

    // newEmployee.name will return John

    var newEmployee = new emp();

    // Change the name property of the newEmployee object

    newEmployee.name = "Mary";

    // Retrieve the name property from the original employee object

    // Notice that name is not changed to Mary, it is still John

    document.write(employee.name);

</script>

**Output :** John   
  
Object defined with a function constructor lets you have multiple instances of that object. This means change made to one instance, will not affect other instances.   
  
**So, when to use one over the other?**  
If you need multiple instances of the object use constructor function where as if you need just one instance of the object then use literal notation.

### Global namespace pollution in JavaScript

### 55. Global namespace pollution in JavaScript

In this video we will understand the **problem of global namespace pollution in JavaScript**with an example. When working with JavaScript on a big project, you might hear senior developers saying during the code review, this code pollutes the global scope. In this video let's understand the problem of global namespace pollution. In our next video we will discuss how to write JavaScript in such a way that it does not pollute the global scope.   
  
   
  
Let us say we have 2 software development teams (**TeamA & TeamB**) working on a large project for **Pragim Technologies**.   
  
**TeamA**has created a **customer**constructor function with 2 parameters (firstName & lastName) and added it to **TeamA.js** file

function customer(firstName, lastName)

{

    this.firstName = firstName;

    this.lastName = lastName;

    this.getFullName = function ()

    {

        return this.firstName + " " + this.lastName;

    }

}

**TeamB**has also created a customer constructor function but with 3 parameters (firstName, middleName & lastName) and added it TeamB.js file. 

function customer(firstName, middleName, lastName)

{

    this.firstName = firstName;

    this.middleName = middleName;

    this.lastName = lastName;

    this.getFullName = function ()

    {

        return this.firstName + " " + this.middleName + " " + this.lastName;

    }

}

So at the moment we have **2 customer constructor functions**. One version has 2 parameters (firstName & lastName) and the other version has 3 parameters (firstName, middleName & lastName).    
  
**HTML Page code :**In this HTML page we want to use the customer constructor function that has 2 parameters (firstName & lastName). Notice that in the header section we are first loading TeamA.js and then TeamB.js.

<html>

    <head>

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

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

    </head>

    <body>

        <div id="resultDiv"></div>

        <script type="text/javascript">

            document.getElementById("resultDiv").innerHTML =

                        new customer("Tom", "Grover").getFullName();

        </script>

    </body>

</html>

When you view the page in the browser, we get the following output. We expected it to print to "Tom Grover"  
**Tom Grover undefined**  
  
**This is because JavaScript does not support function overloading and as a result**, the customer() constructor function with 3 parameters simply overwrites the customer() constructor function with 2 parameters.   
  
You may be wondering why didn't it happen the other way round. Why didn't customer() constructor function with 2 parameters overwrite the customer() constructor function with 3 parameters. That is because TeamB.js is loaded after TeamA.js. So the customer() constructor function in TeamB.js has overwritten the customer() constructor function in TeamA.js.   
  
The reason, one of the customer() constructor function got overwritten is because we have already **polluted the global scope**. Let's understand what we mean by this.    
  
1. window is the Global object in JavaScript   
  
2. When TeamA.js script file is loaded, the customer() function in TeamA.js file is added to the global namespace   
  
3. When TeamB.js script file is loaded, the customer() function in TeamB.js file is added to the global namespace. Since 2 functions with the same name cannot exist in the global namespce, the customer() function in TeamB.js overwrites the customer() function in TeamA.js. As a result we cannot use the customer() function in TeamA.js file.  
  
4. In JavaScript if you declare a variable or a function second time, it simply overwrites the one you created earlier. JavaScript does not raise any errors like C# or Java if you redefine a variable or a function.   
  
**Here is the summary.**  
Polluting global namespace causes name collision. This is especially true in large projects where you may be using several JavaScript libraries (both internally developed as well as third party libraries). That's why it is very important not to add everything to the global namespace. If someone else use the same variable or function names it can lead to name collision.  
  
In our next we will discuss, how to write JavaScript code in such a way that it does not pollute the global scope.

56. Namespace In JavaScript

In this video, we will discuss **how to use namespaces to avoid polluting the global scope**. This is continuation to [Part 55](http://csharp-video-tutorials.blogspot.com/2015/02/global-namespace-pollution-in-javascript.html). Please watch [Part 55](http://csharp-video-tutorials.blogspot.com/2015/02/global-namespace-pollution-in-javascript.html) of [JavaScript tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhUA99nOsJkKXBqokT3MBK0b) to understand the problem of global namespace pollution.   
  
   
  
JavaScript lack namespaces. However we can use objects to create namespaces.   
  
The following line says use the **PragimTech**object if it already exists, otherwise create an empty object.  
var PragimTech = PragimTech || {};  
  
The following line adds a nested namespace. A nested namespace is a namespace inside another namespace. In JavaScript to define a nested namespace, you define an object inside another object.   
PragimTech.TeamA = PragimTech.TeamA || {};  
  
Modify the script in **TeamA.js** file as shown below. In this example we are adding **customer**() function to **PragimTech.TeamA** namespace.   
  
var PragimTech = PragimTech || {};  
PragimTech.TeamA = PragimTech.TeamA || {};  
  
PragimTech.TeamA.customer = function (firstName, lastName)   
{  
    this.firstName = firstName;  
    this.lastName = lastName;  
  
    this.getFullName = function ()   
    {  
        return this.firstName + " " + this.lastName;  
    }  
  
    return this;  
}  
  
PragimTech will be added to the global namespace. window is the alias for global namespace in JavaScript. You can now access customer() function as shown below.  
  
PragimTech.TeamA.customer("Tom", "Grover")  
  
OR  
  
window.PragimTech.TeamA.customer("Tom", "Grover")  
  
Modify the script in **TeamB.js** file as shown below. In this example we are adding **customer**() function to **PragimTech.TeamB**namespace.   
  
var PragimTech = PragimTech || {};  
PragimTech.TeamB = PragimTech.TeamB || {};   
  
PragimTech.TeamB.customer = function (firstName, middleName, lastName)   
{  
    this.firstName = firstName;  
    this.middleName = middleName;   
    this.lastName = lastName;  
  
    this.getFullName = function ()   
    {  
        return this.firstName + " " + this.middleName + " " + this.lastName;  
    }   
  
    return this;  
}  
  
PragimTech will be added to the global namespace. You can now access customer() function as shown below.  
  
PragimTech.TeamB.customer("Tom", "T", "Grover")  
  
OR  
  
window.PragimTech.TeamB.customer("Tom", "T", "Grover")  
  
On any given HTML page you should be able to access both the versions of customer() function as shown below.

<html>

    <head>

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

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

    </head>

    <body>

        <script type="text/javascript">

            // Call the customer function that is defined in TeamA.js

            alert(PragimTech.TeamA.customer("Tom", "Grover").getFullName());

            // Call the customer function that is defined in TeamB.js

            alert(PragimTech.TeamB.customer("Tom", "T", "Grover").getFullName());

        </script>

    </body>

</html>

57.Private Members In JavaScript

In any object oriented programming language, **classes can have private and public members**. For example a class in C# can have private and public fields and functions.   
  
   
  
An example is shown below.

public class Employee

{

    // Private Field

    private string privateFullName;

    // Public Fields

    public string firstName;

    public string lastName;

    // Private Function

    private string privateGetFullName()

    {

        privateFullName = this.firstName + " " + this.lastName;

        return privateFullName;

    }

    // Public Function

    public string publicGetFullName()

    {

        return privateGetFullName();

    }

}

JavaScript is object oriented programming language, so objects in JavaScript can also have private and public fields and functions. An example is shown below. 

function Employee(firstName, lastName)

{

    // Private Field

    var privateFullName;

    // Public Fields

    this.firstName = firstName;

    this.lastname = lastName;

    // Private Function

    var privateGetFullName = function ()

    {

        privateFullName = firstName + " " + lastName;

        return privateFullName;

    }

    // Privileged Function

    this.privilegedGetFullName = function ()

    {

        return privateGetFullName();

    }

    // Public Function

    Employee.prototype.publicGetFullName = function ()

    {

        return this.privilegedGetFullName();

    }

}

In the example above, we also have a privileged Function. So, **what is a Privileged Function**?  
  
1. Privileged methods are created using "this" keyword and Public methods are created using prototype property of the constructor function.  
  
2. Privileged method can access private variables and methods  
  
3. Public methods can call Privileged methods but not Private methods.  
  
4. Like Public methods, Privileged methods are also available outside the constructor function.  
  
**Public fields and functions** are available both inside and outside of the Employee() constructor function. **Private fields and functions**are available only inside the Employee() constructor function. Attempting to access private fields and properties outside of the constructor function will result in undefined error.

var employee = new Employee("Tom", "Grover");

// Calling public function works fine

document.write(employee.publicGetFullName() + "<br/>");

// Calling Privileged function works fine

document.write(employee.privilegedGetFullName() + "<br/>");

// Calling private method - undefined error

employee.privateGetFullName()

// Calling private field - undefined error

employee.privateFullName

**Can we modify a private field outside of the constructor function?**  
Straight answer, no you can't.  
  
In the example below, when we call the private field (employee.privateFullName) it results in undefined error. On the next line we are adding a new public field with same name as the private field to the employee object. Is this going to change the private field (privateFullName). The answer is NO. You cannot access or modify private fields outside of the object. In this example, you are just adding new public field (employee.privateFullName) to the employee object. 

var employee = new Employee("Tom", "Grover");

employee.privateFullName // Calling private field - undefined error

// Add a field with same name as private field

employee.privateFullName = "ABC";

document.write(employee.publicGetFullName() + "<br/>");

document.write(employee.privateFullName);

**Here is the summary**  
  
**Private fields -**Declared using the var keyword inside the object, and can only be accessed by private functions and privileged methods.  
  
**Public fields -**Declared using this keyword and are available outside the object.  
  
**Private functions -**Declared inside the object using one of the 2 ways shown below. Can be called only by privileged methods.  
  
var privateGetFullName = function ()   
{  
    privateFullName = firstName + " " + lastName;  
    return privateFullName;  
}  
  
**OR**  
  
function privateGetFullName()  
{  
    privateFullName = firstName + " " + lastName;  
    return privateFullName;  
}  
  
**Privileged methods -** Declared using **this**keyoword and are available both within and outside the object.  
  
// Privileged Function  
this.privilegedGetFullName = function ()   
{  
    return privateGetFullName();  
}  
  
**Public methods -**Defined by using the object's prototype property and are available both within and outside the object.  
  
// Public Function  
Employee.prototype.publicGetFullName = function ()   
{  
    return this.privilegedGetFullName();  
}

58. Properties in JavaScript

In an **object oriented programming language**, classes can have properties. For example a class in C# can have the following types of properties.   
1. Read / Write properties  
2. Read only properties  
3. Write only properties   
  
   
  
An example of c# **Employee**class is shown below. name is read-only property and age is read/write property.

public class Employee

{

    string \_name;

    int \_age;

    public Employee(string name, int age)

    {

        \_name = name;

        \_age = age;

    }

    // Read/Write property

    public int age

    {

        get

        {

            return \_age;

        }

        set

        {

            \_age = value;

        }

    }

    // Readonly property

    public string name

    {

        get

        {

            return \_name;

        }

    }

}

Since JavaScript is also an object oriented programming language, objects in JavaScript can also have properties.   
  
**Why do we need properties when we have public fields**  
Encapsulation is one of the pillars of object oriented programming language. Properties provide encapsulation. If you use public fields you cannot control on what is assigned and returned from that public field.  
  
In the example below we have an employee object with age public field. There is nothing stopping us from setting the age value of the employee object to 1000. Using properties you can control on what values can be assigned. You can also use properties to create just read-only or write-only properties.

function Employee(age)

{

    this.age = age;

}

var employee = new Employee(50);

// Nothing stops you from assigning a value of 1000 to age field

employee.age = 1000;

An example is shown below. In the example **name is read-only property**and **age is read/write property**. 

function Employee(name, age)

{

    var \_name = name;

    var \_age = age;

    Object.defineProperty(this, "age",

    {

        get: function () { return \_age; },

        set: function (value)

        {

            if (value > 100 || value < 1)

            {

                alert("Invalid age");

            }

            else

            {

                \_age = value;

            }

        }

    });

    Object.defineProperty(this, "name",

    {

        get: function () { return \_name; }

    });

}

var employee = new Employee("Tom", 55);

// Cannot change the value of name property (read-only)

employee.name = "Tommy";

// Will alert an error - Invalid age

employee.age = 195;

document.write(employee.name + " " + employee.age);

59. Static Member in JavaScript

In an **object oriented programming language classes can have static members**, that is **static methods**and **static fields**. For example a class in C# is shown below. In this example PI is a static field and \_radius is an instance field.   
  
 

public class Circle

{

    // Static field

    static float PI = 3.141F;

    // Instance field

    public int Radius;

    public Circle(int raidus)

    {

        this.Radius = raidus;

    }

    public float CalculateArea()

    {

        return Circle.PI \* this.Radius \* this.Radius;

    }

}

What is the **difference between static member and instance member and when to use one over the other**  
An instance member belong to a specific instance. So if I create 3 instances (objects) there will be 3 copies of instance fields in memory where as there will ever be only one copy of a static field no matter how many instances we create.   
  
In the above example, radius is specific to the circle instance you are creating but the PI value need to be shared by all the circle objects. So, if you want a member to be shared by all instances then make it a static member otherwise make it an instance member.   
  
Since JavaScript is also an object oriented programming language, objects in JavaScript can also have static and instance fields and methods. Here is an example. In this example PI is a static field and \_radius is an instance field.  

function Circle(radius)

{

    // Instance field

    this.Radius = radius;

    // Static field

    Circle.PI = 3.141;

    this.CalculateArea = function ()

    {

        return Circle.PI \* this.Radius \* this.Radius;

    }

}

var circleObject = new Circle(10);

document.write("Area = " + circleObject.CalculateArea());

**Points to remember**  
1. Define a static member using the name of the constructor function.  
2. To invoke a static member use the name of the constructor function.  
3. To invoke an instance member use the instance of the constructor function.   
  
**Static methods in JavaScript :**In the following example **ShowCount()**is a static method. Notice that, just like a static variable, static method is also defined using the name of the constructor function. To keep track of the number of Shape objects created we are using Shape.Count static field. **Shape.ShowCount()**method returns the count of shapes when called.

function Shape(shapeName)

{

    // Instance field

    this.ShapeName = shapeName;

    // Static Field

    Shape.Count = ++Shape.Count || 1;

    // Static method

    Shape.ShowCount = function ()

    {

        return Shape.Count;

    }

}

var shape1 = new Shape("Circle");

var shape2 = new Shape("Rectangle");

var shape3 = new Shape("Triangle");

document.write("Shape.Count = " + Shape.ShowCount());

Since we have created 3 shapes, the output will be 3.

60. Prototype in JavaScript

In this video we will discuss **Prototype object in JavaScript**. Let us understand it's use with an example.   
  
   
  
The following **Employee**constructor function constructs **Employee**object.   
function Employee(name)   
{  
    this.name = name;  
}   
  
There are several ways to add a function to the Employee object. One way is as shown below. This works but the problem with this approach is that, if you create 100 employee objects there will be 100 copies of getName() function. We don't want to be creating copies of functions, instead we want all the objects to share the same function code. This can be achieved using JavaScript prototype object.   
  
function Employee(name)   
{  
    this.name = name;   
  
    this.getName = function ()   
    {  
        return this.name;  
    }  
}   
  
var e1 = new Employee("Mark");  
var e2 = new Employee("Sara");   
  
document.write("e1.name = " + e1.getName() + "<br/>");  
document.write("e2.name = " + e2.getName() + "<br/>");   
  
**Output :**  
e1.name = Mark  
e2.name = Sara   
  
In this example, **getName()** function is added just to e1 object, and not to e2 object. So e2.getName() would throw an undefined error. 

function Employee(name)

{

    this.name = name;

}

var e1 = new Employee("Mark");

e1.getName = function ()

{

    return this.name;

}

var e2 = new Employee("Sara");

document.write("e1.name = " + e1.getName() + "<br/>");

document.write("e2.name = " + e2.getName() + "<br/>");

In the following example **getName()**function is added to the Employee object using the name of the constructor function. This would also result in undefined error. 

function Employee(name)

{

    this.name = name;

}

Employee.getName = function ()

{

    return this.name;

}

var e1 = new Employee("Mark");

var e2 = new Employee("Sara");

document.write("e1.name = " + e1.getName() + "<br/>");

document.write("e2.name = " + e2.getName() + "<br/>");

**Using the prototype object to add functions :**The following are the advantages of using the prototype object to add functions.  
1. No matter how many objects you create, functions are loaded only once into memory  
2. Allows you to override functions if required. 

function Employee(name)

{

    this.name = name;

}

Employee.prototype.getName = function ()

{

    return this.name;

}

var e1 = new Employee("Mark");

var e2 = new Employee("Sara");

document.write("e1.name = " + e1.getName() + "<br/>");

document.write("e2.name = " + e2.getName() + "<br/>");

**Output :**  
e1.name = Mark  
e2.name = Sara

61. Overriding JavaScript functions

In this video we will discuss **how to override a JavaScript function**. This is continuation to [Part 60](http://csharp-video-tutorials.blogspot.com/2015/02/prototype-in-javascript.html). Please watch [Part 60](http://csharp-video-tutorials.blogspot.com/2015/02/prototype-in-javascript.html) from [JavaScript tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhUA99nOsJkKXBqokT3MBK0b) before proceeding.   
  
   
  
In [Part 60](http://csharp-video-tutorials.blogspot.com/2015/02/prototype-in-javascript.html), we discussed that one of the advantages of using prototype property to add functions is that it enables us to **override an existing function** if required. Let us continue with the same example we worked with in [Part 60](http://csharp-video-tutorials.blogspot.com/2015/02/prototype-in-javascript.html). 

function Employee(name)

{

    this.name = name;

}

Employee.prototype.getName = function ()

{

    return this.name;

}

var e1 = new Employee("Mark");

var e2 = new Employee("Sara");

document.write("e1.name = " + e1.getName() + "<br/>");

document.write("e2.name = " + e2.getName() + "<br/>");

**Output :**  
e1.name = Mark  
e2.name = Sara  
  
Let us say for some reason we want to override **getName**() function of **Employee**object. Notice that in **GetEmployeeDetails()**function we have overridden **getName()**function of the Employee object. The overridden version converts the name of the Employee to UPPERCASE.

<script type="text/javascript">

    GetEmployeeDetails();

    function GetEmployeeDetails()

    {

        Employee.prototype.getName = function ()

        {

            return this.name.toUpperCase();

        }

        var e1 = new Employee("Mark");

        var e2 = new Employee("Sara");

        document.write("e1.name = " + e1.getName() + "<br/>");

        document.write("e2.name = " + e2.getName() + "<br/>");

    }

    function Employee(name)

    {

        this.name = name;

    }

    Employee.prototype.getName = function ()

    {

        return this.name;

    }

</script>

**Output :**  
e1.name = MARK  
e2.name = SARA  
  
In this example, all the JavaScript is in the same file, i.e  
1. The JavaScript that creates Employee constructor function and getName() function &  
2. The script that overrides getName() function   
  
The JavaScript that creates Employee constructor function and getName() function could even be present in a separate JavaScript file.   
  
1. To your project add a new JScript file with name = EmployeeScript.js.  
  
2. Copy and paste the following JavaScript code in EmployeeScript.js file  
  
function Employee(name)   
{  
    this.name = name;  
}  
  
Employee.prototype.getName = function ()   
{  
    return this.name;  
}   
  
3. Modify the code on the HTML page as shown below.

<html>

    <head>

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

        </script>

    </head>

    <body>

        <script>

            GetEmployeeDetails();

            function GetEmployeeDetails()

            {

                Employee.prototype.getName = function ()

                {

                    return this.name.toUpperCase();

                }

                var e1 = new Employee("Mark");

                var e2 = new Employee("Sara");

                document.write("e1.name = " + e1.getName() + "<br/>");

                document.write("e2.name = " + e2.getName() + "<br/>");

            }

        </script>

    </body>

</html>

Run the page and the output should be exactly the same as the previous example.  
  
JavaScript built-in methods can also be overridden. The following example overrides the built-in JavaScript alert() function.

<script type="text/javascript">

    // Overriding JavaScript alert function to write to the page

    // instead of displaying the alert dialog box

    var alert = function (msg)

    {

        document.write(msg);

    }

    // The following calls will invoke the overridden alert() method

    alert("Hello");

    window.alert(" JavaScript");

</script>

**Output :**Hello JavaScript

### 62. Inheritance in JavaScript

In this video we will discuss **Inheritance in JavaScript with an example**.  
  
Object oriented programming languages support inheritance. Since JavaScript is also an object oriented programming language, it supports inheritance.    
  
   
  
In Object oriented programming languages like C# and Java to implement inheritance, a class inherits from another class. In JavaScript, we don't have the concept of classes, so **inheritance in JavaScript is prototype-based**. This means to implement inheritance in JavaScript, an object inherits from another object. Let us understand this with an example. 

<script type="text/javascript">

    // Employee will be the base object (Similar to base class in c#)

    var Employee = function (name)

    {

        this.name = name;

    }

    // getName() function is added to the base object (Employee)

    Employee.prototype.getName = function ()

    {

        return this.name;

    }

    // PermanentEmployee will be the derived object

    var PermanentEmployee = function (annualSalary)

    {

        this.annualSalary = annualSalary;

    }

    // Use prototype to associate Employee as the base object for PermanentEmployee

    PermanentEmployee.prototype = new Employee("Mark");

    var pe = new PermanentEmployee(50000);

    // Derived object (permanentEmployee) can see the

    // base object (Employee) getName() method

    document.write(pe.getName());

    alert(pe instanceof Employee);          // Returns true

    alert(pe instanceof PermanentEmployee); // Returns true

</script>

Notice that the derived object (PermanentEmployee) can see the base object (Employee) getName() method. When getName() method is called, JavaScript first tries to find this method in the derived object (). It can't find the method there so it goes to the parent object and finds it there.   
  
If you add a new method to the parent object, it becomes available in the derived object.

<script type="text/javascript">

    var Employee = function (name)

    {

        this.name = name;

    }

    Employee.prototype.getName = function ()

    {

        return this.name;

    }

    // Adding getNameLength() method to the parent object

    // which becomes available in the derived object

    Employee.prototype.getNameLength = function ()

    {

        return this.name.length + " characters";

    }

    // PermanentEmployee will be the derived object

    var PermanentEmployee = function (annualSalary)

    {

        this.annualSalary = annualSalary;

    }

    PermanentEmployee.prototype = new Employee("Mark");

    var pe = new PermanentEmployee(50000);

    // Call getNameLength() method added to the parent object

    document.write(pe.getNameLength()); // Output : 4 characters

</script>

Use **hasOwnProperty()**method to determine if a property is defined on the actual object or it's prototype. Here is an example.

<script type="text/javascript">

    var Employee = function (name)

    {

        this.name = name;

    }

    var PermanentEmployee = function (annualSalary)

    {

        this.annualSalary = annualSalary;

    }

    var employee = new Employee("Mark");

    PermanentEmployee.prototype = employee;

    var pe = new PermanentEmployee(50000);

    document.write("Employee.name: " +

                     employee.hasOwnProperty('name') + "<br/>");

    document.write("Employee.annualSalary: " +

                     employee.hasOwnProperty('annualSalary') + "<br/><br/>");

    document.write("PermanentEmployee.name: " +

                     pe.hasOwnProperty('name') + "<br/>");

    document.write("PermanentEmployee.annualSalary: " +

                     pe.hasOwnProperty('annualSalary') + "<br/>");

</script>

### Output : Employee.name: true Employee.annualSalary: false PermanentEmployee.name: false PermanentEmployee.annualSalary: true

### 63. Abstract classes in JavaScript

In this video we will discuss, **how to implement abstract classes concept in JavaScript**.   
  
   
  
Object oriented programming languages like C# and Java, support **abstract classes**. Abstract classes are incomplete. So, trying to create an instance of an abstract class raises a compiler error. Abstract classes can only be used as base classes.    
  
Let us first look at a simple C# example.  
  
1. Open visual studio and create a new empty asp.net web application project.   
  
2. Add a web form to the project. Name it WebForm1.aspx. Copy and past the following code in the code-behind file.

using System;

namespace Demo

{

    public partial class WebForm1 : System.Web.UI.Page

    {

        protected void Page\_Load(object sender, EventArgs e)

        {

            // Error: Cannot create an instance of an abstract class

            // Shape s = new Shape();

            Circle circle = new Circle();

            circle.shapeName = "Circle";

            Response.Write(circle.draw() + "<br/><br/>");

            Response.Write(circle is Shape + "<br/>");  // Returns true

            Response.Write(circle is Circle + "<br/>"); // Returns true

        }

    }

    public abstract class Shape

    {

        public string shapeName = "None";

        public string draw()

        {

            return "Drawing " + this.shapeName;

        }

    }

    public class Circle : Shape

    {

        // Code specific to Circl class

    }

}

Since JavaScript is also an object oriented programming language, it also supports the concept of abstract classes. Here is an example.   
  
Add a new HTML page to the project. Name it HTMLPage1.htm. Copy and paste the following code in HTMLPage1.htm. 

<script type="text/javascript">

    // Create a Shape object which is abstract

    var Shape = function ()

    {

        this.shapeName = "None";

        throw new Error("Cannot create an instance of abstract class");

    }

    // Error : Cannot create an instance of abstract class

    // var shape = new Shape();

    // Add draw function to the Shape prototype

    // Objects derived from Shape should be able to call draw() method

    Shape.prototype.draw = function ()

    {

        return "Drawing " + this.shapeName;

    }

    // Create a Circle object

    var Circle = function (shapeName)

    {

        this.shapeName = shapeName;

    }

    // Make shape the parent for Circle

    // Object.create() allows to create an object without using constructor

    Circle.prototype = Object.create(Shape.prototype);

    var circle = new Circle("Circle");

    // Since Circle inherits from abstract Shape object, it can call draw() method

    document.write(circle.draw());

    alert(circle instanceof Circle); // Returns true

    alert(circle instanceof Shape);  // Returns true

</script>

64. Polymorphism in JavaScript

In this video we will discuss, **how to implement polymorphism in JavaScript with an example**.   
Object oriented programming languages like C# and Java, support polymorphism. Here is a C# example. 

public partial class WebForm1 : System.Web.UI.Page

{

    protected void Page\_Load(object sender, EventArgs e)

    {

        Shape[] shapes = new Shape[]

        {

            new Shape(), new Circle(), new Square(), new Triangle()

        };

        foreach (Shape shape in shapes)

        {

            Response.Write(shape.draw() + "<br/>");

        }

    }

}

public class Shape

{

    public virtual string draw()

    {

        return "I am a generic shape";

    }

}

public class Circle : Shape

{

    public override string draw()

    {

        return "I am a circle";

    }

}

public class Square : Shape

{

    public override string draw()

    {

        return "I am a square";

    }

}

public class Triangle : Shape

{ }

**Output :**  
I am a generic shape  
I am a circle  
I am a square  
I am a generic shape   
  
Since JavaScript is also an object oriented programming language, it also supports the concept of polymorphism. Here is an example. 

<script type="text/javascript">

    // Shape object is be the base object

    var Shape = function () { }

    // Add draw function to the Shape prototype

    // Objects derived from Shape should be able to override draw() method

    Shape.prototype.draw = function ()

    {

        return "I am a generic shape";

    }

    // Create a Circle object

    var Circle = function () { }

    // Make shape the parent for Circle

    Circle.prototype = Object.create(Shape.prototype);

    // Circle object overrides draw() method

    Circle.prototype.draw = function ()

    {

        return "I am a circle";

    }

    var Square = function () { }

    Square.prototype = Object.create(Shape.prototype);

    Square.prototype.draw = function ()

    {

        return "I am a square";

    }

    var Triangle = function () { }

    Triangle.prototype = Object.create(Shape.prototype);

    var shapes = [new Shape(), new Circle(), new Square(), new Triangle()];

    shapes.forEach(function (shape)

    {

        document.write(shape.draw() + "<br/>")

    });

</script>

**Ouptut:**  
I am a generic shape  
I am a circle  
I am a square  
I am a generic shape

65. Object Reflection in JavaScript

In this video we will discuss, **object reflection in JavaScript with an example**.   
Object oriented programming languages like C# and Java, support reflection. Reflection allows us to inspect meta data of assemblies, modules and types. Since JavaScript is also an object oriented programming language, it also supports the concept of reflection. Let's understand Object reflection in JavaScript with an example.   
  
Employee object in the following example has **4 public properties**  
1. firstName  
2. lastName  
3. email  
4. gender  
  
and **3 public functions**  
1. getFullName  
2. getEmail  
3. getGender

var Employee = function (firstName, lastName, gender, email)

{

    this.firstName = firstName;

    this.lastName = lastName;

    this.gender = gender;

    this.email = email;

}

Employee.prototype.getFullName = function ()

{

    return this.firstName + " " + this.lastName;

}

Employee.prototype.getEmail = function ()

{

    return this.email;

}

Employee.prototype.getGender = function ()

{

    return this.gender;

}

var employee = new Employee("Mark", "Matt", "Male", "a@a.com");

The following code retrieves all the public properties and methods of the Employee object.   
  
for (var property in employee)   
{  
    document.write(property + "<br/>");  
}   
  
**Output :**  
firstName  
lastName  
gender  
email  
getFullName  
getEmail  
getGender   
  
The following code retrieves all the **public properties & functions of the Employee object** along with their values.   
  
for (var property in employee)   
{  
    document.write(property + " : " + employee[property] + "<br/>");  
}   
  
**Output :**  
firstName : Mark  
lastName : Matt  
gender : Male  
email : a@a.com  
getFullName : function () { return this.firstName + " " + this.lastName; }  
getEmail : function () { return this.email; }  
getGender : function () { return this.gender; }   
  
The following code retrieves only the public properties of the Employee object.  
  
for (var property in employee)   
{  
    if (typeof employee[property] != "function")   
    {  
        document.write(property + " : " + employee[property] + "<br/>");  
    }  
}   
  
**Output:**  
firstName : Mark  
lastName : Matt  
gender : Male  
email : a@a.com   
  
The following code retrieves only the public functions of the Employee object.   
  
for (var property in employee)   
{  
    if (typeof employee[property] == "function")   
    {  
        document.write(property + " : " + employee[property] + "<br/>");  
    }  
}   
  
**Output :**  
getFullName : function () { return this.firstName + " " + this.lastName; }  
getEmail : function () { return this.email; }  
getGender : function () { return this.gender; }   
  
When 2 objects are related thru inheritance, the child object will have access to parent object methods and properties. In this example, PermanentEmployee is a child of Employee. The following code shows all the properties and methods of the PermanentEmployee object including those it inherited from the Employee object.   
  
var employee = new Employee("Mark", "Matt", "Male", "a@a.com");   
  
var PermanentEmployee = function (annualSalry)   
{  
    this.annualSalary = annualSalry;  
}   
  
PermanentEmployee.prototype = employee;   
  
var pe = new PermanentEmployee(50000);   
  
for (var property in pe)   
{  
    document.write(property + " : " + pe[property] + "<br/>");  
}   
  
**Output :**  
annualSalary : 50000  
firstName : Mark  
lastName : Matt  
gender : Male  
email : a@a.com  
getFullName : function () { return this.firstName + " " + this.lastName; }  
getEmail : function () { return this.email; }  
getGender : function () { return this.gender; }   
  
In this example, we are using **hasOwnProperty()** method to determine if a property is defined on the actual object or it's prototype. This method returns true if the property is defined by the object itself, otherwise false. The following code retrieves only the public members that are defined in the PermanentEmployee object. The members inherited from the base Employee object are excluded.   
  
for (var property in pe)   
{  
    if (pe.hasOwnProperty(property))   
    {  
        document.write(property + " : " + pe[property] + "<br/>");  
    }  
}   
  
Output : annualSalary : 50000   
  
The following code retrieves only the public members inherited from the parent object. Public members defined in PermanentEmployee object are excluded.  
for (var property in pe)   
{  
    if (!pe.hasOwnProperty(property))   
    {  
        document.write(property + " : " + pe[property] + "<br/>");  
    }  
}   
  
**Output :**  
firstName : Mark  
lastName : Matt  
gender : Male  
email : a@a.com  
getFullName : function () { return this.firstName + " " + this.lastName; }  
getEmail : function () { return this.email; }  
getGender : function () { return this.gender; }

66. Strict Mode In JavaScript

Just like Java and C#, JavaScript is also an object oriented programming language. However, **JavaScript is not very strict in reporting or throwing errors**. Let us understand what this statement means with an example.   
  
   
  
In C#, if you assign a value to a variable that is not declared, you get an error stating - The name 'myString' does not exist in the current context.   
  
myString = "This is a string";  
  
Where as the same code in JavaScript does not raise any error. JavaScript automatically creates a global variable with name **myString**. This kind of a behavior may lead the developer to make more mistakes.    
  
myString = "This is a string";  
document.write(myString);  
  
**Output :**This is a string  
  
Just like Java & C#, if you want JavaScript also to throw an error, then **use Strict Mode**.   
  
**How to enable strict mode in JavaScript :**Just add "use strict" statement in your script file as shown below. When you run the page, you will now get an error stating - Variable undefined in strict mode. To see the error in Google chrome, please go the console window in developer tools. 

<script type="text/javascript">

    "use strict";

    myString = "This is a string";

    document.write(myString);

</script>

Since **"use strict"**is specified at the top of the JavaScript file, strictness will be enforced across the entire script file.   
  
**How to enforce JavaScript strictness in a specific function :**Just add "use strict" statement in the function as shown below. In this example, strictness is enforced only in **myFunction**(). 

<script type="text/javascript">

    myString = "This is a string<br/>";

    document.write(myString);

    function myFunction()

    {

        "use strict";

        var myOtherString = "This is also a string";

        document.write(myOtherString);

    }

    myFunction();

</script>

**Output :**  
This is a string  
This is also a string   
  
**Another example :**In C# if you assign a value to a read-only property you get an error. For example, the following C# code would raise an error stating - Property or indexer 'Demo.Employee.Name' cannot be assigned to -- it is read only. 

public partial class WebForm1 : System.Web.UI.Page

{

    protected void Page\_Load(object sender, EventArgs e)

    {

        Employee employee = new Employee("Mark");

        employee.Name = "Mark M";

    }

}

public class Employee

{

    private string \_name;

    public Employee(string name)

    {

        this.\_name = name;

    }

    public string Name

    {

        get

        {

            return this.\_name;

        }

    }

}

We discussed [JavaScript properties](http://csharp-video-tutorials.blogspot.com/2015/02/properties-in-javascript.html) in [Part 58](http://csharp-video-tutorials.blogspot.com/2015/02/properties-in-javascript.html) of [JavaScript tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhUA99nOsJkKXBqokT3MBK0b).  
  
In JavaScript, if you assign a value to a read-only property, JavaScript silently fails without raising an error.

<script type="text/javascript">

    var Employee = function (name)

    {

        var \_name = name;

        Object.defineProperty(this, "name", {

            get: function ()

            {

                return \_name;

            }

        });

    }

    var employee = new Employee("Mark");

    // name is readonly property. It is an error to assign a value to a read-only property

    // JavaScript silently fails the following line without raising an error

    employee.name = "Mark M";

    document.write(employee.name);

</script>  
  
If you want JavaScript to raise an error instead of failing silently, use JavaScript strict mode. The code below raises an error stating - Assignment to read-only properties is not allowed in strict mode.

<script type="text/javascript">

    "use strict";

    var Employee = function (name)

    {

        var \_name = name;

        Object.defineProperty(this, "name",

        {

            get: function () {

                return \_name;

            }

        });

    }

    var employee = new Employee("Mark");

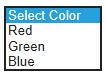
    employee.name = "Mark M";

    document.write(employee.name);

</script>

**ECMAScript version 5**introduced **strict mode to JavaScript**. With strict mode on it is easier to detect JavaScript silent errors as they would throw an error now. This makes debugging much easier and the chances of developers making mistakes is reduced. Most modern browsers support strict mode.  
  
For the list of most important restrictions that apply in strict mode, please refer the following MSDN article  
<https://msdn.microsoft.com/en-us/library/ie/br230269(v=vs.94).aspx>

67. JavaScript Cookies

**In this video we will discuss**  
1. What are cookies and why are they needed  
2. How to write a cookie and read it later   
  
   
  
**Why are cookies needed**  
Web applications work on HTTP protocol which is a stateless protocol. This means after the web server has processed the client request for a web page, the web server will not remember anything about the client that made the request. Let us understand what this statement means with an example.   
  
1. Design a web page with a DropDownList as show below.   
   
  
2. When the user selects a color from the DropDownList we want to change the background color of the page to the selected color.   
   
  
3. We want the web application to remember the user's preferred color. On a subsequent visit to the same page it should display the page with the background color that we previously selected.    
  
4. Now close the browser. Open a new instance of the browser and navigate to the same page. Notice that, the page does not remember the color we selected earlier. This is because web applications work on HTTP protocol which is a stateless protocol. This means after the web server has processed the initial client request, it does not remember aynthing about the settings the client made. There are several ways to make a web application remember these settings. One of the easiest and common ways is by using cookies.   
  
**What are cookies**  
Cookies are small text files that a browser stores in the visitor's computer. A cookie is basically a string of name-value pairs separated by semi-colons.  
  
**How does a cookie look**  
**color=red;expires=Fri, 5 Aug 2016 01:00:00 UTC;**  
  
**In the cookie string above we have 2 name-value pairs.**  
The **expires**attribute specifies when the cookie is going to expire. By default cookies are deleted when the current browser session ends. If you want to store the cookie on the client computer even after the browser session has ended then specify either expires or max-age attributes. The color attribute is set to "red".  
  
**How to write a cookie** : Use the document object's cookie property to write the cookie.  
document.cookie = "color=red;expires=Fri, 5 Aug 2016 01:00:00 UTC;"  
  
**How to read a cookie :** Use the document object's cookie property to read the cookie.  
var cookieString = document.cookie;  
  
**Here is the complete example which remembers the user color preference**

<select id="ddlTheme" onchange="setColorCookie()">

    <option value="Select Color">Select Color</option>

    <option value="red">Red</option>

    <option value="green">Green</option>

    <option value="blue">Blue</option>

</select>

<script type="text/javascript">

    window.onload = function ()

    {

        if (document.cookie.length != 0)

        {

            var nameValueArray = document.cookie.split("=");

            document.getElementById("ddlTheme").value = nameValueArray[1];

            document.bgColor = nameValueArray[1];

        }

    }

    function setColorCookie()

    {

        var selectedValue = document.getElementById("ddlTheme").value;

        if (selectedValue != "Select Color")

        {

            document.bgColor = selectedValue;

            document.cookie = "color=" + selectedValue +

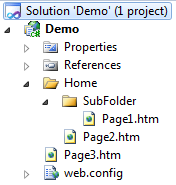
                            ";expires=Fri, 5 Aug 2016 01:00:00 UTC;";

        }

    }

</script>

### 68. JavaScript cookie attributes

In this video we will discuss some of the optional cookie attribute values that can be specified when creating a cookie. This is continuation to [Part 67](http://csharp-video-tutorials.blogspot.com/2015/02/javascript-cookies.html), please watch [Part 67](http://csharp-video-tutorials.blogspot.com/2015/02/javascript-cookies.html)of [JavaScript tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhUA99nOsJkKXBqokT3MBK0b) before proceeding.   
  
   
  
**Optional Cookie Attributes**  
expires  
max-age  
domain  
path  
secure   
  
The following code creates a cookie that expires as soon as the browser is closed  
document.cookie = "color=red";   
  
**expires and max-age attributes :** If you want to create a persistent cookie, that is a cookie that is not deleted after the browser is closed either use expires or max-age attributes   
  
document.cookie = "color=red;expires=Fri, 5 Aug 2016 01:00:00 UTC";  
OR  
document.cookie = "color=red;max-age=" + (60 \* 60 \* 24 \* 30) + ";"  
// The above code creates a cookie that expires in 30 days   
  
**What is the difference between expires and max-age attributes**  
With expires attribute you set an expiry date. The expires attribute is obsolete. Very few modern browsers support this attribute. Internet Explorer is one of them.  
With max-age attribute you specify the expiry time in seconds. Most modern browsers support this attribute except Internet Explorer.   
  
If you want to create a persistent cookie that works in IE and all the other browsers then specify both expires and max-age attributes.   
  
**domain attribute :**Specifies the domain for which the cookie is valid. If you specify the domain as PragimTech.Blog.com then that cookie will be valid only for that sub-domain. It will not be valid for PragimTech.com.    
  
If you want a cookie to be valid for all sub-domains of PragimTech.com then specify domain=PragimTech.com. So this cookie will be valid for  
PragimTech.com  
PragimTech.Blog.com  
PragimTech.Blog.KudVenkat.com   
  
**path attribute :** By default cookies are valid only for web pages in the directory of the current web page that stored them, as well as its descendants.    
  
**Example :**The following diagram shows the directory structure of a web application   
   
  
If a cookie is set by http://localhost/Home/Page2.htm, it will be valid for http://localhost/Home/SubFolder/Page1.htm but not for http://localhost/Page3.htm   
  
If you want to create a cookie that is valid across all your pages in your website, then set the path attribute to the root of your web directory, that is, **"/"**.   
  
document.cookie = "color=red;max-age=" + (60 \* 60 \* 24 \* 30) + ";path=/";   
  
**secure attribute :**secure attribute specifies that the cookie is secure and is only used over HTTPS protocol which ensures that the cookie is always encrypted when transmitting from client to server.

69. Store Multiple Key Values in a Cookies

In this video we will discuss, if it's possible to **store multiple key value pairs in a cookie**. Let us understand this with an example.   
  
   
  
Design a web page as shown in the image below   
   
  
When we click **"Set Cookie"** button, we want to store name, email & gender along with their values in the cookie.   
  
**Copy and paste the following code in HTMLPage1.htm**

<html>

<head></head>

<body>

    <table border="1">

        <tr>

            <td>Name</td>

            <td><input type="text" id="txtName" /></td>

        </tr>

        <tr>

            <td>Email</td>

            <td><input type="text" id="txtEmail" /></td>

        </tr>

        <tr>

            <td>Gender</td>

            <td><input type="text" id="txtGender" /></td>

        </tr>

        <tr>

            <td colspan="2">

                <input type="button" value="Set Cookie" onclick="setCookie()" />

                <input type="button" value="Get Cookie" onclick="getCookie()" />

                <input type="button" value="Clear" onclick="clearTextBoxes()" />

            </td>

        </tr>

    </table>

    <script type="text/javascript">

        function setCookie()

        {

            var cookieString = "name=" + document.getElementById("txtName").value +

                               ";email=" + document.getElementById("txtEmail").value +

                               ";gender=" + document.getElementById("txtGender").value;

            document.cookie = cookieString;

        }

        function getCookie()

        {

            alert(document.cookie);

        }

        function clearTextBoxes()

        {

            document.getElementById("txtName").value = "";

            document.getElementById("txtEmail").value = "";

            document.getElementById("txtGender").value = "";

        }

    </script>

</body>

</html>

1. Run the application.   
2. Fill in data for Name, Email and Gender.   
3. Click "Set Cookie" button  
4. Click "Get Cookie" button   
  
Notice that we only get the first key-value pair. This is because you can only store one key-value pair in one cookie  
name=Venkat  
  
**If you want to store all the 3 key-value pairs, you have 2 options**  
1. Create a custom object, serialize the object to a JSON string and store the serialized string in a cookie  
2. Use a different cookie for each key-value pair you want to store  
  
Create a custom object, serialize the object to a JSON string and store the serialized string in a cookie :   
  
Modify the code in **setCookie()**function as shown below.  

function setCookie()

{

    var customObject = {};

    customObject.name = document.getElementById("txtName").value;

    customObject.email = document.getElementById("txtEmail").value;

    customObject.gender = document.getElementById("txtGender").value;

    var jsonString = JSON.stringify(customObject);

    document.cookie = "cookieObject=" + jsonString;

}

**JSON.stringify()** method converts a JavaScript object to a JavaScript Object Notation (JSON) string.   
  
Modify the code in **getCookie()** function as shown below.  

function getCookie()

{

    var nameValueArray = document.cookie.split("=");

    var customObject = JSON.parse(nameValueArray[1]);

    document.getElementById("txtName").value = customObject.name;

    document.getElementById("txtEmail").value = customObject.email;

    document.getElementById("txtGender").value = customObject.gender;

}

**JSON.parse()** method parses a JSON string and returns the corresponding object.   
  
In our next video we will discuss the second option, i.e using a different cookie for each key-value pair that we want to store.

70. Set and Get Multiple Cookies in JavaScript

In this video, we will discuss how to **set and get multiple cookies in JavaScript**. This is continuation to [Part 69](http://csharp-video-tutorials.blogspot.com/2015/02/store-multiple-key-value-pairs-in-cookie.html). Please watch [Part 69](http://csharp-video-tutorials.blogspot.com/2015/02/store-multiple-key-value-pairs-in-cookie.html) before proceeding.   
  
   
  
   
  
When we click **"Set Cookie"** button we want to store the following 3 key-value pairs in 3 cookies.  
name=Venkat;  
email=Kudevnkat@gmail.com;  
gender=Male;   
  
When we click **"Get Cookie"** button we want to retrieve all the 3 key-value pairs from the 3 cookies   
  
Modify the code in **setCookie()** function as shown below.

function setCookie()

{

    document.cookie = "name=" + document.getElementById("txtName").value;

    document.cookie = "email=" + document.getElementById("txtEmail").value;

    document.cookie = "gender=" + document.getElementById("txtGender").value;

}

The above code creates **3 cookies** and stores the **3 key-value pairs**. At this point document.cookie property contains the following string  
"name=Venkat; email=Kudvenkat@gmail.com; gender=Male"  
  
Now, modify the code in **getCookie()** function as shown below.

function getCookie()

{

    if (document.cookie.length != 0)

    {

        var cookiesArray = document.cookie.split("; ");

        for (var i = 0; i < cookiesArray.length; i++)

        {

            var nameValueArray = cookiesArray[i].split("=");

            if (nameValueArray[0] == "name")

            {

                document.getElementById("txtName").value = nameValueArray[1];

            }

            else if (nameValueArray[0] == "email")

            {

                document.getElementById("txtEmail").value = nameValueArray[1];

            }

            else if (nameValueArray[0] == "gender")

            {

                document.getElementById("txtGender").value = nameValueArray[1];

            }

        }

    }

    else

    {

        alert("Cookies not found");

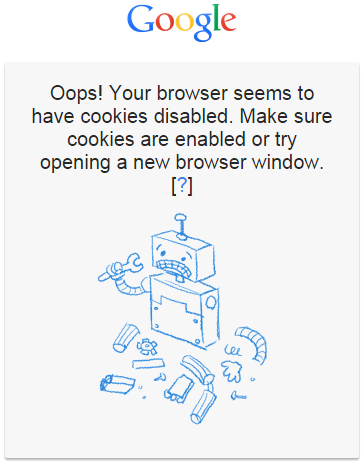
    }

}

71. Update and Delete Cookie

**Updating a cookie :** To update a cookie, set a cookie with the same name again. The following code updates the color cookie with the New\_Color if a cookie with name=color exists otherwise it creates a new cookie with that name.  
document.cookie = "color=New\_Color";   
  
**Deleting a cookie :** To delete a cookie, set a cookie with **max-age**attribute set to a negative value  
document.cookie = "color=red;max-age=-60";  
  
If you are using a browser that supports **expires**attribute, then set the expires attribute to a date in the past.  
document.cookie = "color=red;expires=Mon, 01 Jan 1900 01:00:00 UTC;";   
  
**Please note:** Users can also use the browser interface to delete cookies anytime they want.   
  
**Cookie limitations: -**  
1. What is the maximum allowed cookie size per cookie  
2. How many cookies are allowed per domain  
  
The answers to these questions really depend on the browser you are using. The following website can be used to find out what the limits are  
[http://browsercookielimits.squawky.net](http://browsercookielimits.squawky.net/)

### 72. How to check if cookies are enabled

In this video we will discuss, **how to check if cookies are enabled**.   
  
   
  
**Most websites rely on browser cookies being enabled**. For example, if you have disabled cookies and if you try to log into gmail.com, you will receive an error message saying - Oops! Your browser seems to have cookies disabled. Make sure cookies are enabled or try opening a new browser window.    
  
   
  
The following JavaScript code detects if cookies are enabled. Most modern browser's support **navigator.cookieEnabled** property.    
  
This property returns true if cookies are enabled and false if cookies are disabled. Some old browsers don't support this property.    
  
For those browsers that don't support navigator.cookieEnabled property, we are setting **"mytestcookie"** and then reading again to make sure cookies are enabled. If we are not able to read **"mytestcookie"**then we know that cookies are disabled.

<div id="msg"></div>

<script type="text/javascript">

    function cookiesEnabled()

    {

        var cookiesEnabled = (navigator.cookieEnabled) ? true : false;

        if (typeof navigator.cookieEnabled == "undefined" && !cookiesEnabled)

        {

            document.cookie = "mytestcookie";

            cookiesEnabled = (document.cookie.indexOf("mytestcookie") != -1) ? true : false;

        }

        return cookiesEnabled;

    }

    if (cookiesEnabled())

    {

        document.getElementById("msg").innerHTML = "Cookies enabled";

    }

    else

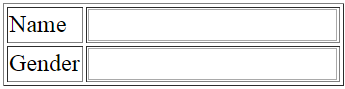
    {

        document.getElementById("msg").innerHTML = "Cookies disabled";

    }

</script>

### 73. How to check if JavaScript is enabled

In this video we will discuss **how to check if JavaScript is enabled or disabled**. Let us understand this with an example.   
  
   
  
We have a page that captures **Name**and **Gender.**Both fields are required. The page relies on JavaScript being enabled for performing validation. We should always have server side validation irrespective of whether we have client side validation or not. It is not a good practice to rely just on client side validation, but for the purpose of this demo, let us say we only have client side validation.   
  
   
  
**If JavaScript is enabled** then we want to display the Name and Gender elements and the user can continue to use the application.   
  
**If JavaScript is disabled** then we want to hide the Name and Gender elements and we want to display the following message.  
It seems that you have disabled JavaScript. Please enable JavaScript.  
  
**The easiest way to detect if JavaScript is enabled is by using noscript tag**. The content inside the <noscript> element will be displayed only if scripts are not supported, or are disabled in the user's browser. 

<noscript>

    <style type="text/css">

        .rootDiv

        {

            display: none;

        }

    </style>

    <h1>It seems that you have disabled JavaScript. Please enable JavaScript.</h1>

</noscript>

<div id="rootElement" class="rootDiv">

    <table border="1">

        <tr>

            <td>Name</td>

            <td>

                <input id="txtName" type="text" onfocus="validateIfEmpty('txtName')"

           onblur="validateIfEmpty('txtName')" onkeyup="validateIfEmpty('txtName')"/>

            </td>

        </tr>

        <tr>

            <td>Gender</td>

            <td>

                <input id="txtGender" type="text" onfocus="validateIfEmpty('txtGender')"

           onblur="validateIfEmpty('txtGender')" onkeyup="validateIfEmpty('txtGender')"/>

            </td>

        </tr>

    </table>

</div>

<script type="text/javascript">

    function validateIfEmpty(control)

    {

        var controlToValidate = document.getElementById(control);

        if (controlToValidate.value == "")

        {

            controlToValidate.style.background = "red";

        }

        else

        {

            controlToValidate.style.background = "white";

        }

    }

</script>

### 74.window.location in JavaScript

The **Window.location** property returns a **Location**object that can be used to get information about the current page. Window.location property can also be used to redirect the browser to a new page.   
  
In [Part 73](http://csharp-video-tutorials.blogspot.com/2015/02/how-to-check-if-javascript-is-enabled.html) of [JavaScript tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhUA99nOsJkKXBqokT3MBK0b) we discussed how to detect if JavaScript is enabled by using **<noscript>** element. Another way is by using window.location property. Let us use the example we worked with in [Part 73](http://csharp-video-tutorials.blogspot.com/2015/02/how-to-check-if-javascript-is-enabled.html). We will use window.location property along with <noscript> element to detect if JavaScript is enabled.   
  
Add a new HTML page to your project. Name it Default.htm. Copy and paste the following HTML and JavaScript.

<html>

<head>

    <script>

        // If JavaScript is enabled this code redirects the user to HTMLPag1.htm

        // If JavaScript is disable this code will not execute and the user reamains on this

        // page and he gets to the see the message that JavaScript is disabled.

        window.location = "/HTMLPage1.htm";

    </script>

</head>

<body>

    <h1>It seems that you have disabled JavaScript. Please enable JavaScript.</h1>

</body>

</html>

We don't need to make any modification to **HTMLPage1.htm**.  
  
**At this point**  
If you have JavaScript enabled, and if you visit Default.htm page, you will be redirected to HTMLPage1.htm  
If you have JavaScript disabled, and if you visit Default.htm page, you will reamin on Default.htm page and you will get to see the message that says JavaScript is disabled.  
  
**window.location** property is especially useful if you have 2 sites  
1. One for users with JavaScript   
2. Another for users without JavaScript  
  
Make the home-page of Non-JavaScript website the default page. In the default page include the following JavaScript code to redirect the user to the JavaScript-EnabledSite.com  
window.location = "http://www.JavaScript-EnabledSite.com";  
  
If you have JavaScript enabled, you will be redirected to JavaScript enabled website.  
If you have JavaScript disabled, you will not be redirected and stay with the website which works without JavaScript.   
  
**Some of the useful properties of the location object**

|  |  |
| --- | --- |
| **window.location.href** | Returns the URL of the current page |
| **window.location.hostname** | Returns the domain name |
| **window.location.protocol** | Returns the protocol (http or https) |
| **window.location.pathname** | Returns the path of the current page |

<script type="text/javascript">

    document.write("window.location.href = " + window.location.href + "<br/>");

    document.write("window.location.hostname = " + window.location.hostname + "<br/>");

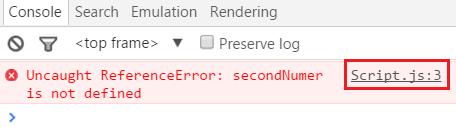
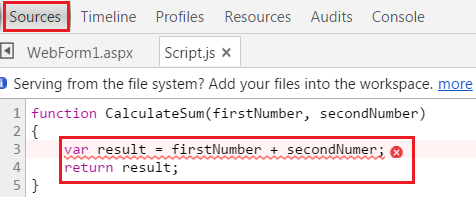
    document.write("window.location.pathname = " + window.location.pathname + "<br/>");

    document.write("window.location.protocol = " + window.location.protocol + "<br/>");

</script>

**Output :**  
window.location.href = http://localhost:57695/Default.htm  
window.location.hostname = localhost  
window.location.pathname = /Default.htm  
window.location.protocol = http:

75. Debugging JavaScript in chrome

To debug JavaScript in Google chrome use developer tools.   
  
   
  
**To launch Developer Tools in Google chrome**  
**1.** Click on **Customise and Control Google chrome** button on the top right corner of the browser window  
**2.** From the More tools option, select Developer tools   
  
Alternatively you can also use the keyboard shortcut **F12**  
  
You can find JavaScript errors if any in the developer tools.   
Debugging JavaScript in chrome   
  
Use the **Console tab** in the developer tools to find the source file of the JavaScript that caused the error. This will also tell you the line number of the error.   
   
  
Click on the JavaScript file name in the **Console tab**. This will open the JavaScript file in the Sources tab and underlines the line that caused the error with a red squiggly.   
   
  
**Setting breakpoints in Google Chrome :** To set a breakpoint, simply click on the grey margin where you see line numbers in the Sources tab.  
  
A blue tag appears indicating that a breakpoint is set. At this point if you reload the page, the breakpoint should be hit and you should be able to   
  
**Step thru the code using the following keyboard shortcuts**  
Step over - F10  
Step into - F11  
Step out - Shift + F11  
Continue - F8  
  
You can also use the following buttons in the Developer tools to step thru the code instead of using the keyboard shortcuts.   
step through code in google chrome   
  
To remove a breakpoint simply click on the grey margin again.  
  
**Setting a conditional breakpoint :** To set a conditional breakpoint, right click on the grey margin and select **"Add conditional breakpoint"**. You can then specify the condition that should be true for the breakpoint to be hit.  
  
**Call stack panel :** The Call Stack panel displays the complete execution path.   
