JAVA SCRIPT

Scripting Language

- Scripting languages are mostly associated with the World Wide Web where they have been used extensively to create dynamic web pages

- There are two ways to carry out the processing needed to generate the dynamic web page.

  - **client-side scripting**
  - **server-side scripting**

Client Side Scripting

- The client is the system on which the Web browser is running

- Client-side scripts are interpreted by the browser. The process with client-side scripting is:
  - The user requests a Web page from the server
  - The server finds the page and sends it to the user
  - The page is displayed on the browser with any scripts running during or after display

- Client-side scripting is used to make Web pages change after they arrive at the browser

Client Side Scripting

- user's web browser should understands the scripting language in which client-side scripts are written
- Client-side scripts rely on the user's computer and the code which makes up the script is there in the HTML for the user to look at
- Examples of Client side scripting languages : Javascript, VB script, etc.
- The biggest drawback of client-side scripting is the security implications
- The code could order the browser to carry out all sorts of actions, including accessing data on the computer itself

Server Side Scripting

- Server-side scripting is about "programming" the behavior of the server.
- In server side scripting the script is executed at the server end and produce output in a format understandable by web browsers which is then sent over to the user end.
- The user cannot see the script's source code and may not even be aware that a script was executed

Server Side Scripting

The process is:

- the user requests a Web page from the server
- the script in the page is interpreted by the server creating or changing the page content to suit the user and the occasion.
- the page in its final form is sent to the user and then cannot be changed using server-side scripting

- server-side scripts have greater access to the information and functions available on the server

- Server-side scripts require that their language's interpreter be installed on the server, and produce the same output regardless of the client's browser, operating system, or other system details

Server Side Scripting

Server Scripts are used for:

- customize the responses based on user's requirements, access rights etc
- Respond to user queries or data submitted from HTML forms
- Access any data or databases and return the result to a browser
- Provide security since your server code cannot be viewed from a browser

- Because the scripts are executed on the server, the browser that displays the file does not need to support scripting at all.

- Examples of Server side scripting languages : PHP, JSP, ASP

JAVA SCRIPT

- JavaScript is a Client Side Scripting Language designed for creating dynamic Web pages.
- JavaScript Syntax are similar to C and Java Language.
- JavaScript code is typically embedded in the HTML, to be interpreted and run by the client's browser.

What can we do with JAVASCRIPT

- To create interactive user interface in a web page (e.g., menu, pop-up alert, windows, etc.)

- Manipulating web content dynamically
  - Change the content and style of an element
  - Replace images on a page without page reload
  - Hide/Show contents

- JavaScript can produce pages that are customized to the user. Create pages dynamically Based on the user's choices, the date, or other external data

- It can do some processing of forms and can validate user input when the user submits the form

JavaScript is not Java

- Java and JavaScript are two completely different languages in both concept and design!
- JavaScript has some features that resemble features in Java:
  - JavaScript has Objects and primitive data types
  - JavaScript has qualified names; for example, document.write("Hello World");
  - JavaScript has Events and event handlers
  - Exception handling in JavaScript is almost the same as in Java

JavaScript is not Java

JavaScript has some features unlike anything in Java:

- Variable names are untyped
- Objects and arrays are defined in quite a different way
- JavaScript is an interpreted language but java is both interpreted and compiled

<SCRIPT> tag

- The <SCRIPT> tag alerts a browser that JavaScript code follows. It is typically embedded in the HTML.

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

- **Example**

```
    <SCRIPT language = "JavaScript">
    alert("Welcome to the script tag test page.")
    </SCRIPT>
```

Where Do You Place Scripts

- JavaScript can be put in the **<head>** or in the **<body>** of an HTML document
  - JavaScript *functions* should be defined in the **<head>**
    - This ensures that the function is loaded before it is needed
  - JavaScript in the **<body>** will be executed as the page loads

- JavaScript can be put in a separate .js file
  - **<script src="myJavaScriptFile.js"></script>**
  - Put this HTML wherever you would put the actual JavaScript code
  - An external .js file lets you use the same JavaScript on multiple HTML pages
  - The external .js file cannot itself contain a **<script>** tag

- JavaScript can be put in HTML *form object*, such as a button
  - This JavaScript will be executed when the form object is used

JavaScript Variables

- JavaScript variables can be used to hold values (x=5) or expressions (z=x+y).
- The keyword **var** is used to declare a variable

```
Eg;    var num = “1”;
       var name = “Kaushal”;
       var phone = “123-456-7890”;
```

- The keyword **var** is optional (but it’s good style to use it)
- Variables names must begin with a letter or underscore
- Variable names are case-sensitive
- Variables are untyped (they can hold values of any type)
- Variables declared within a function are local to that function

Data Types

- Primitive data types
  - Number: integer & floating-point numbers
  - Boolean: true or false
  - String: a sequence of alphanumeric characters

- Composite data types (or Complex data types)
  - Object: a named collection of data
  - Array: a sequence of values (an array is actually a predefined object)

- Special data types
  - Null: the only value is "null" – to represent nothing.
  - Undefined: the only value is "undefined" – to represent the value of an un intialized variable

Literals

- JavaScript Literals are the notation for representing a fixed data value that appears directly in a JavaScript program.

- JavaScript Literals helps us to assign values (initialize) to various JavaScript Variables; such as numbers, strings, and Booleans; It can also be used to initialize arrays and objects.

- **var** intNum = 82 *// 82 is a integer literal*
  var PI = 3.14 *// 3.14 is a float literal*
  var strInfo="This is a string" *// this is a string literal*
  var isMale=**true** *// A Boolean value literal*
  var arrAnimal={"cat","dog"} *// array literals*

Arrays

- An array is represented by the `Array` object. To create an array of N elements, you can write

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

where index of array runs from 0 to N-1.

- An array can store values of different types
- Property "length" tells the no.of elements in the array.
- Consists of various methods to manipulate its elements. e.g., **reverse(), push(), concat(),** etc

Array Examples

```
var Car = new Array(3);
Car[0] = "Ford";
Car[1] = "Toyota";
Car[2] = "Honda";

// Create an array of three elements with initial values
   var Car2 = new Array("Ford", "Toyota", "Honda");

// Create an array of three elements with initial values
   var Car3 = ["Ford", "Toyota", "Honda"];

// An array of 3 elements with initial values of different types
   var tmp3 = new Array(1, "a", true);

// Makes tmp3 an array of 100 elements. tmp[3] to
     tmp[98] are undefined.
   tmp3[99] = "Something";
```

Operators

- Most JavaScript syntax is borrowed from C. So java Script supports the following operators
  - **Arithmetic operators:**
  - **Comparison operators:**
  - **Logical operators:**
  - **Bitwise operators:**
  - **Assignment operators:**
  - **Special operators:**

Comparison Opeartors

- Let var var1 = 3, var2 = 4;

Operator	Description	Examples returning true
Equal (==)	Returns true if the operands are equal.	3 == var1 "3" == var1
Not equal (!=)	Returns true if the operands are not equal.	var1 != 4 var2 != "3"
Strict equal (===)	Returns true if the operands are equal and of the same type.	3 === var1
Strict not equal (!==)	Returns true if the operands are not equal and/or not of the same type.	var1 !== "3" 3 !== '3'

Special operators

- **Conditional operator**

  condition ? value_if_true : value_if_false

- **delete**

  deletes an object or an element at a specified index in an array

  delete objectName;

  delete ArrayName[index];

- **instanceof**

  returns true if the specified object is of the specified object type

  Syntax: objectName instanceof objectType

Ex:
```
var theDay = new Date(1995, 12, 17);
if (theDay instanceof Date) {
  // statements to execute
}
```

Special operators

- **new**

  creates an instance of a user-defined object type or of one of the predefined object types

- **typeof**

  Syntax: typeof operand

  OR

  typeof (operand)

  The typeof operator returns a string indicating the type of the unevaluated operand

```
var shape = "round";          typeof shape; // returns "string"
var size = 1;                 typeof size; // returns "number"
var today = new Date();       typeof today; // returns "object"
```

EXPRESSION

- An expression is any valid set of literals, variables, operators that evaluates to a single value. The value may be a number, a string, or a logical value.

- There are two types of expressions:
  - those that assign a value to a variable

    x = 7
  - those that simply have a value

    3 + 4 -simply evaluates to 7; it does not perform an assignment

JavaScript has the following kinds of expressions:

- **Arithmetic**: evaluates to a number
- **String**: evaluates to a character string, for example
- **Logical**: evaluates to true or false

JavaScript Conditional Statements

- In JavaScript we have the following conditional statements:
  - **if statement**
  - **if...else statement**
  - **if...else if....else statement**
  - **switch statement**

JavaScript Looping Statements

- **while**
  - Syntax:

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

- **do...while**
  - Syntax:

    do {
      code to be executed } while (*condition*)

- **for**
  - Syntax:
    - for (*initialization*; *condition*; *increment*)
    - { *code to be executed* }

Functions

- JavaScript functions are created by the help of "function" keyword
- Syntax:

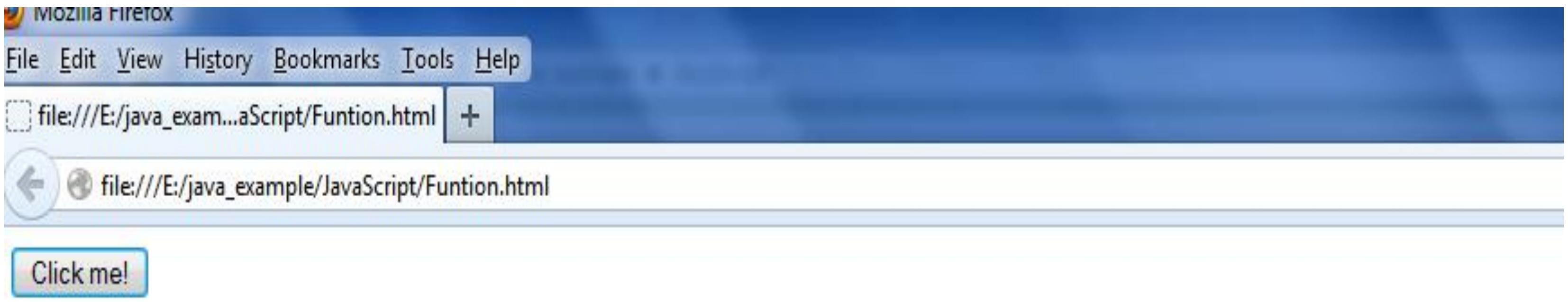
```
function function_name(arguments){
    statement here
}
```

- A JavaScript function contains some code that will be executed only by an event or by a call to that function
- You may call a function from anywhere within the page (or even from other pages if the function is embedded in an external .js file).
- Functions can be defined either <head> or <body> section. As a convention, they are typically defined in the <head> section

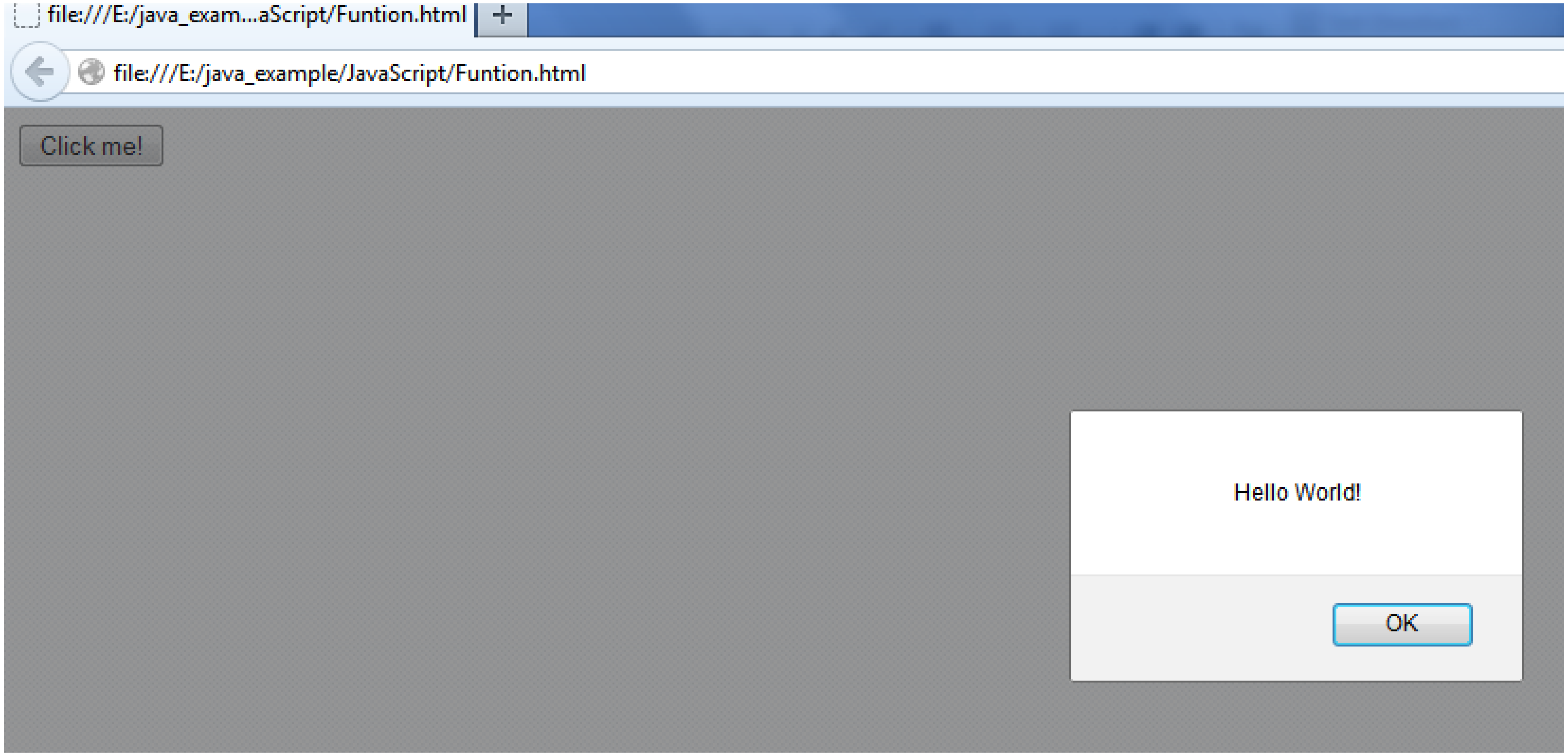
Functions

```
<html>
<head>
    <script type="text/javascript">
        function displaymessage() {
            alert("Hello World!")
         }
    </script>
</head>
<body>
<form>
<input type="button" value="Click me!“
        onclick="displaymessage()" >
</form>
</body>
</html>
```

Functions


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///E:/java_exam...aScript/Funtion.html
+
file:///E:/java_example/JavaScript/Funtion.html
Click me!

Functions


file:///E:/java_exam...aScript/Funtion.html
file:///E:/java_example/JavaScript/Funtion.html
Click me!
OK

Hello World!

Built-in Functions

- In addition to the functions created by user, java script has got a set of built-in functions
- **`eval(String)`**
  - evaluates an expression contained within strings

    eval("3 + 4"); // Returns 7 (Number)

`isFinite(x)`

- Returns true if the given argument is a finite, legal number

  var v1==isFinite(”123”);//true

  var v2==isFinite(”25/12/86”);//false

`isNaN(x)`

- Determines whether given argument is “Not a Number”. Returns true if the given argument is not a number

  var ssn=“123-45-6789”;

  var check=isNaN(ssn);//true

Built-in Functions

- **parseInt(s)**
  - Converts string literals to integers
    - parseInt("3 chances")   // returns 3
    - parseInt("   5 alive")   // returns 5
    - parseInt("How are you") // returns NaN

- **parseFloat(s)**
  - Finds a floating-point value at the beginning of a string.
    - parseFloat("3e-1 xyz")   // returns 0.3
    - parseFloat("13.5 abc")   // returns 13.5

OBJECTS

- JavaScript supports programming with objects and every object has its own properties and methods
- Properties define the characteristics of an object.

  Examples: color, length, name, height, width
- To refer to a property of an object

  objectName.propertyName
- Methods are the actions that the object can perform or that can be performed on the object.
- *Examples: alert, confirm, write, open, close()*
- To refer to a method of an object use:

  objectName.methodName()

OBJECTS

Some of the built-in objects of JavaScript are:

Object	Description
Array	Creates new array objects
Boolean	Creates new Boolean objects
Date	Retrieves and manipulates dates and times
Math	Contains methods and properties for performing mathematical calculations
Number	Contains methods and properties for manipulating numbers.
String	Contains methods and properties for manipulating text strings

User Defined OBJECTS

There are 2 different ways to create a new object:

- Define and create a direct instance of an **object**.
- Use a function to define an object, then create new object instances.

Creating direct instances

- An object can be created by invoking the built-in constructor for the **Object** class
- **var objectname=new Object();**

This example creates a new instance of an object, and adds four properties to it:

```
var person=new Object();
person.firstname="John";
person.lastname="Doe";
person.age=50;
person.eyecolor="blue";
```

Using an Constructor function

- You can define your own custom objects using a **constructor function.**
- Java script objects inherits all the variables and statements of the constructor function on which they are based
- Any javascript function can serve as a constrcutor

```
function person(firstname,lastname,age,eyecolor)
{
      this.firstname=firstname;
      this.lastname=lastname;
     this.age=age;
    this.eyecolor=eyecolor;
}
```

- Once you have a constructor function, you can create new instances of the object, like this:

```
var m1=new person("John","Doe",50,"blue");
var m2=new person("Sally","Rally",48,"green");
```

Using an Constructor function

- To delete a specific property in custom object use delete operator

  SYNTAX

  delete objectname.property
- It is possible to create functions that will be used as a method of object
- Defining methods to an object is done inside the constructor function

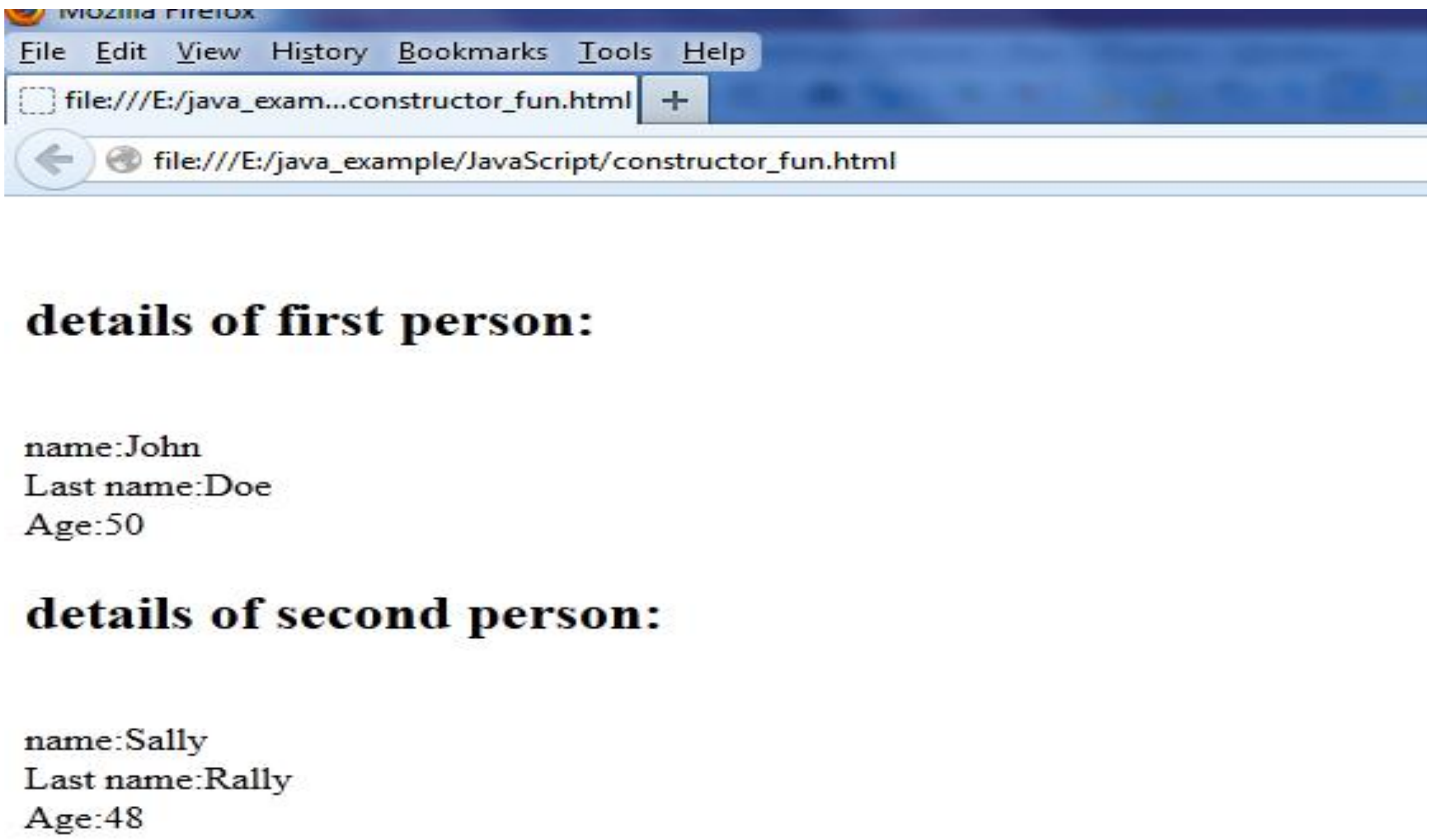
EXAMPLE:

```
<html>
<body>
<script>
function display()
{    document.write("<br/>name:"+this.firstname);
     document.write("<br/>Last name:"+this.lastname);
     document.write("<br/>Age:"+this.age);
}
```

Adding Methods to JavaScript Objects

```
function person(firstname,lastname,age,eyecolor)
{
        this.firstname=firstname;
        this.lastname=lastname;
        this.age=age;
        this.eyecolor=eyecolor;
        this.display_details=display;
}
var m1=new person("John","Doe",50,"blue");
 document.write("<br/><h2>details of first person:</h2>");
m1.display_details();
var m2=new person("Sally","Rally",48,"green");
  document.write("<br/><h2>details of second person:</h2>");
m2.display_details();
</script>
</body>
</html>
```

Adding Methods to JavaScript Objects


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///E:/java_exam...constructor_fun.html
file:///E:/java_example/JavaScript/constructor_fun.html

```
details of first person:

name:John
Last name:Doe
Age:50

details of second person:

name:Sally
Last name:Rally
Age:48
```

Looping Statements-**for…in**

Syntax:

for (variable in object | array])
{ statements }

- It enables to iterate through the properties of an object, or each element of an array.

- **Object**, and **array** represents the object or array over which to iterate.

- **Statement** represents one or more statements to be executed for each property of object or each element of array.

- Eg:

```
var myArray = ["aa", "bb"];
   // for-in loop
 for (var i in myArray) {
 document.write("\t"+myArray[i]);
 }
```

```
function person(firstname,lastname,age,eyecolor)
{
        this.firstname=firstname;
        this.lastname=lastname;
        this.age=age;
        this.eyecolor=eyecolor;
}
var m1=new person("John","Doe",50,"blue");
 for (var i in m1) {
        document.write(m1[i]);
        document.write("<br />");
 }
```

Events

- JavaScript supports an event handling system.
- Every element on a web page is capable of creating certain events.
- Events include such activities as mouse clicks, mouse movement, pressing keyboard keys, the opening and closing of windows and frames, and the resizing of windows.
- Code that executes in response to a specific event is called an event handler
- The event handler code is included as an attribute of the element that initiates the event.
- The synatx of an event handler within an element is:

  <event_handler="javascript code">

Events

- For example, we can add to a **button** element a click attribute that is assigned some java script code which will run when a user clicks on the button.

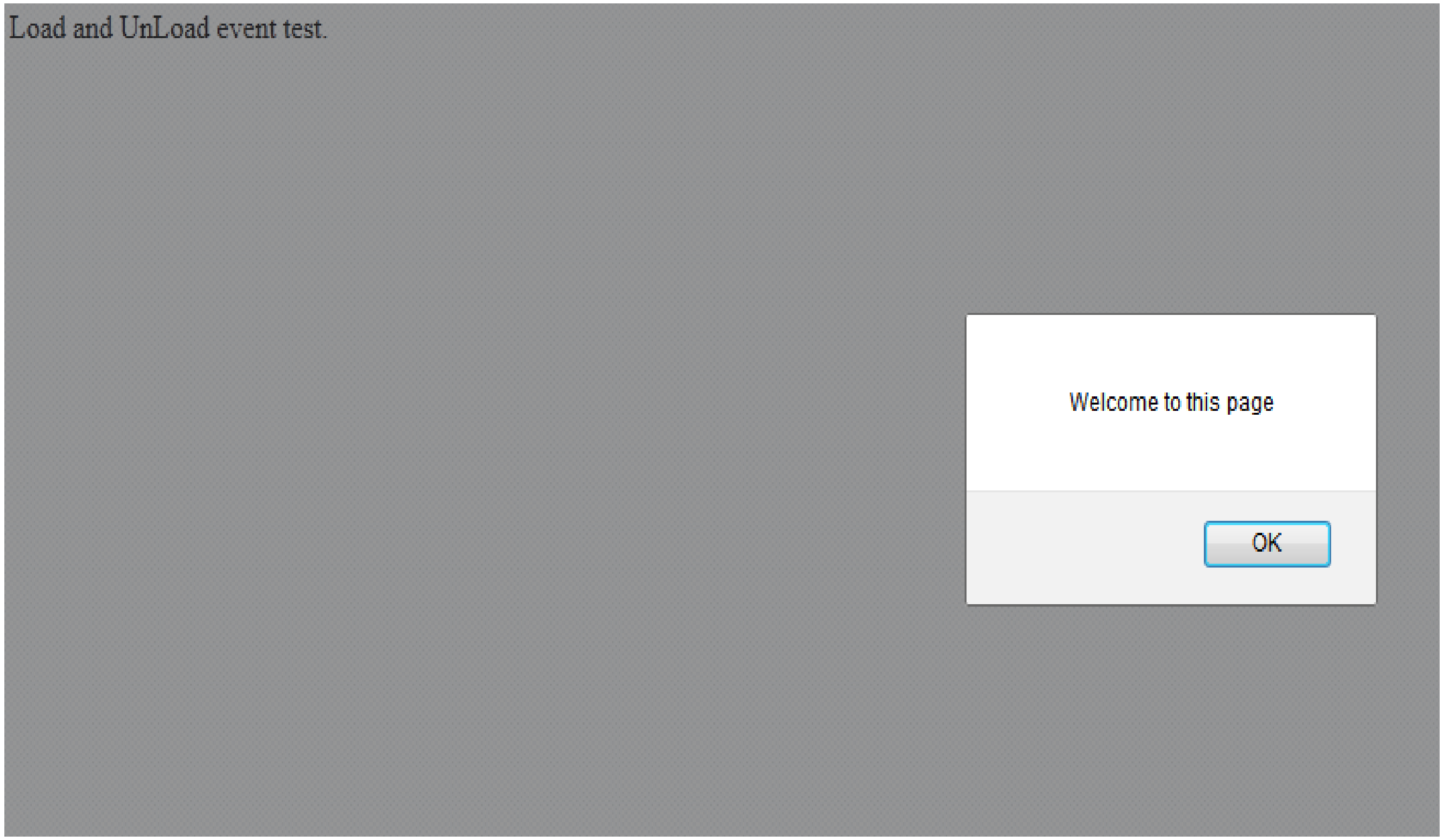
  <input type="button" onclick="alert("you clicked a button")">

Event handler	**Triggered when**
onChange	The value of the text field, text area, or a drop down list is modified
onClick	A link, an image or a form element is clicked once
onDblClick	The element is double-clicked
onMouseDown	The user presses the mouse button
onLoad	A document or an image is loaded
onSubmit	A user submits a form
onUnLoad	The user closes a document or a frame
onResize	A form is resized by the user

Events:Example

```
<html><head>
<title>onLoad and onUnload Event Handler
  Example</title>
</head>
<body
  onLoad="alert('Welcome to this page')"
>
Load event and Unload test.
</body>
</html>
```

Events

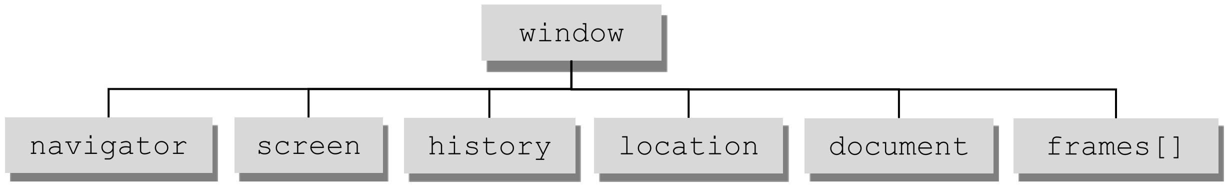


Load and UnLoad event test.

Welcome to this page

Browser Object Model

- The Browser Object Model (BOM) defines a hierarchy of objects and their relationships, which are used to manipulate or obtain information about the browser.
- The objects in the browser-object-model are automatically created when a browser opens a web page
- The top level of the hierarchy is the ***window object***, which contains the information about the window displaying the document



- Underneath the **Window Object** there are lots of other objects, which allows you do things like, redirecting the user to a different web page, get the size of the browser window, access all the HTML elements on the page etc.

Browser Object Model

- The **Document** object: represents the document currently displayed
- The **Frames** collection: represents the frames in the current window
- The **screen** object: represents properties of the screen on which the current window is being rendered.
- The **History** object: The history object maintains a list of URLs that the browser has visited
- The **Location** object: represents the current URL that the browser is displaying
- The **Navigator** object: represents information about the browser itself, like the version number

Window Object

- The window object represents the browser window the document is being displayed in
- It is created for each window that appears on the screen. A window can be the main window, or a new window
- It contains information about both the display window and the browser displaying the document.
- It also contains methods for generating such things as error messages and prompt boxes and have information on things that have been moved to the Windows clipboard from that page.

Window Object

- The Window object is the *global object* within the browser, ie, all other accessible programmatic elements within the browser hang off it
- This includes the scripts you write and the variables you create within those scripts.
- As the global object, it is not technically necessary to write "window" before properties and methods of the window object.

PROPERTIES

- **closed-**Returns a Boolean value indicating whether a window has been closed or not
- **document-**Returns the Document object for the window
- **innerHeight** -Sets or returns the inner height of a window's content area

Window Object

- **name-** sets or returns the name of a window
- **Frames[]-**returns an array listing the Frame objects in a window
- **innerWidth**-Sets or returns the inner width of a window's content area
- **Opener-** refers to window that opened another window

METHODS

- **alert()-**Displays an alert box with a message and an OK button
- **confirm()-**Displays a dialog box with a message and an OK and a Cancel button
- **open()-**Opens a new browser window
- **prompt()-**Displays a dialog box that prompts the visitor for input
- **resizeTo()-**

Document Object

- Document object represents the Web page that is loaded in the browser window, and the content displayed on that page.
- All elements on a web page are contained within document object, and each element is represented in java script by its own objects

PROPERTIES

- **bgColor-** Gets/sets the background color of the current document.
- **Forms-**Returns a list of the FORM elements within the current -document.
- **lastModified**-Returns the date on which the document was last modified.
- **Links**-Returns a list of all the hyperlinks in the document.
- **Title**-Returns the title of the current document

Document Object

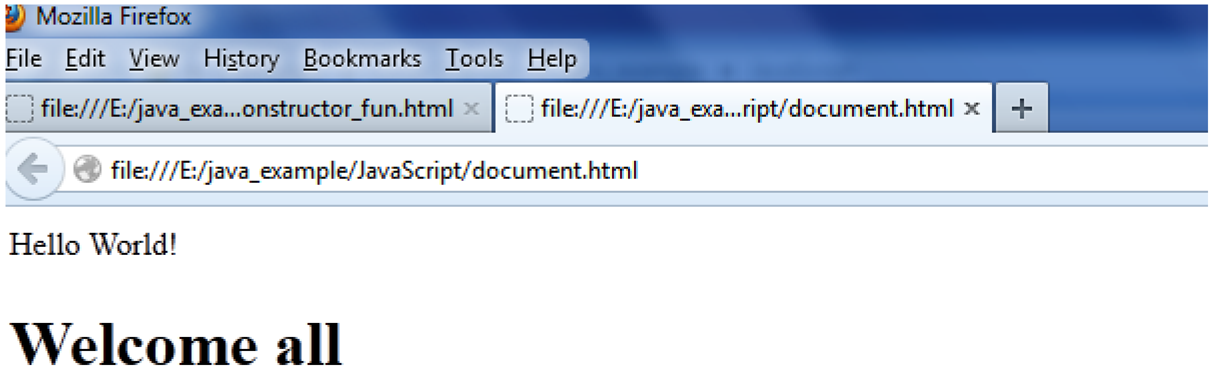
METHODS

- **getElementsByTagName()-** Returns a list of elements with the given name.
- **getElementById()-** Returns an object reference to the identified element
- **write()**-Writes one or more HTML expressions to a document in the specified window
- **captureEvents()**-Sets the document to capture all events of the specified type.
- **open()** - opens a new document
- **close()** - closes the document

Document Object

```
<html>
     <body>
     <script type="text/javascript">
          document.write("Hello World!")
           document.write("<H1>Welcome all </H1><BR>")
     </script>
     </body>
     </html>
```

Document Object


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///E:/java_exa...onstructor_fun.html
file:///E:/java_exa...ript/document.html
file:///E:/java_example/JavaScript/document.html

Hello World!

Welcome all

FORMS

- Forms are one of the most common web page elements which is used to collect information from users and transmit that information to a server for processing.
- There are 4 primary elements used within **<form>** element to create form controls:**<input>,<button>,<select>** and **<textarea>**
- The <input> element is the most commonly used form element used to create the following types of form controls(interface elements):

Text boxes, Password boxes, radio buttons, check boxes, push buttons, submit buttons etc..

<form>

input elements

< /form>

- Two important attributes associated with form tag are action and method

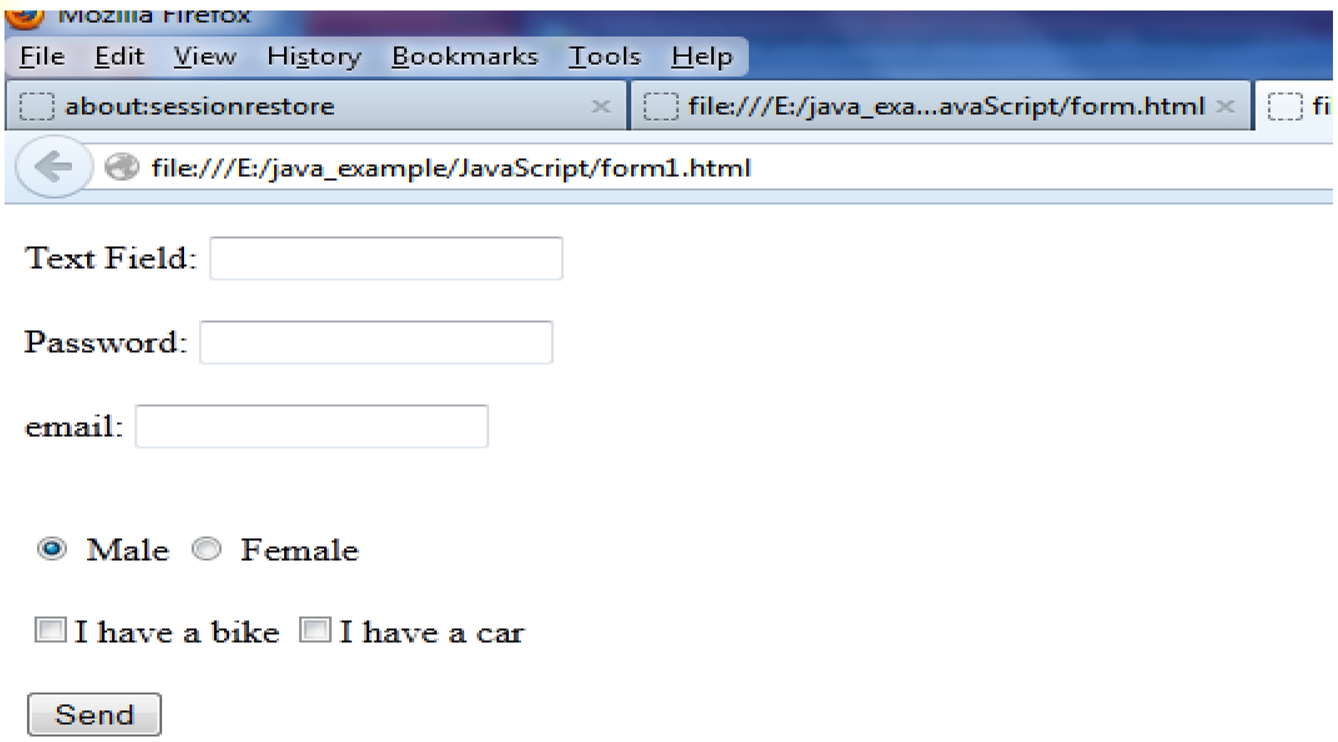
<form action="formprocessor.html" method="get">

</form>

FORMS

```
<html>
<body>
<FORM action="Formprocessor.html" method="post">
<P>Text Field: <INPUT type="text" name="firstname"><BR><BR>
     Password: <input type="password" name="pwd"><BR><BR>
    email: <INPUT type="text" name="email"><BR><BR><BR>
   <INPUT type="radio" name="sex" value="Male"> Male
   <INPUT type="radio" name="sex" value="Female"> Female<BR><BR>
    <input type="checkbox" name="vehicle" value="Bike">I have a bike
   <input   type="checkbox"   name="vehicle"   value="Car">I   have   a
car<br><BR>
  <INPUT type="submit" value="Send">
  </P>
</form> </body>
</html>
```

FORMS


Mozilla Firefox
File Edit View History Bookmarks Tools Help
about:sessionrestore
file:///E:/java_exa...avaScript/form.html
fi
file:///E:/java_example/JavaScript/form1.html

Text Field:

Password:

email:

[✓ Male] Female

I have a bike I have a car

Send

FORMS

- JavaScript can be used to validate data in HTML forms before sending off the content to a server.

  Form data that typically are checked by a JavaScript could be:

  has the user left required fields empty?

  has the user entered a valid e-mail address?

  has the user entered a valid date?

  has the user entered text in a numeric field?

- Each time you add a set of <form> and </form> tags to an HTML document, a form object is created.

- The **attributes** of form element are represented by properties of **Form** object.

- The **Document** object has a **forms[]** array which contains all the forms on a web page.

FORMS

- **Form** object has an **elements[]** array which can be used to access each element in a form

- To access forms using Java Script, you can use any one of the following options.
  - Use forms array of document object
  - Name the form in the opening form tag and use that name to access form
  - Give the form an id in opening form tag and access from using the **document.getElementById()** method

- Java script uses **Form** object to access form controls and verify form information, which represents a form on a web page.

- The form object contains properties, methods and events that you can use to manipulate form and form controls

FORMS

Form object properties:

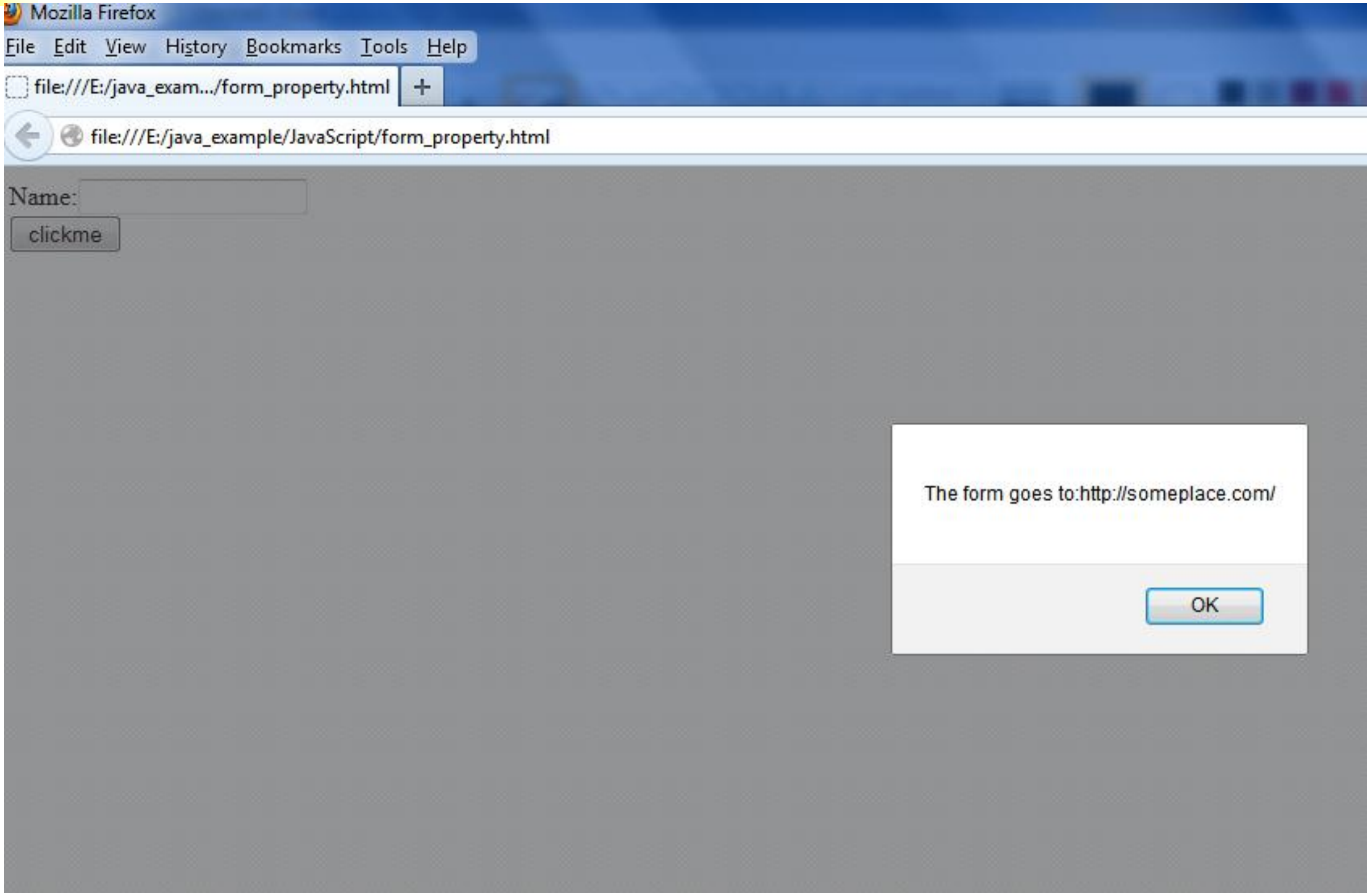
- **action:** returns the value that represents the value of action attribute in the form tag
- **method:** returns a string that represents the value of method attribute in form tag
- **length:** the value of total number of elements in an html form
- **name**: value of name attribute in html form tag
- **elements**: an array that includes an array element for each form element in an html form

- With the help of hierarchy of objects available in **browser object model** it is possible to access the various interface elements available in an HTML page and validate them using java Script.

FORMS

```
<html>
<head>
<body>
<script type="text/javascript">
function prop()
{  alert("The form goes to:"+info_form.action);
}
</script>
</head>
<body>
 <form action="http://someplace.com" name="info_form">
   Name:<input type="text"/></br>
   <input type="button" value="clickme" onclick="prop()"/>
 </form>
</body>
</html>
```

FORMS



The form goes to:http://someplace.com/

Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///E:/java_exam.../form_property.html
file:///E:/java_example/JavaScript/form_property.html
Name:
clickme
OK

FORMS

The syntax for accessing text entries:

<INPUT TYPE = TEXT> is

formName.elementName.value

Where **value** is a keyword, **formName** and **elementName** are the names you gave your form and text input element

So – to get the value a user entered into a text field:

```
<form action="/cgi-bin/test.cgi" name="myForm“>
    Name:<input type="text" name="Name" />
 </form>
    myTextVariable = myForm.name.value
```

Form Validation

To validate the contents of the form before sending it to server you need to know when the user tries to submit form.

When user clicks the submit button, a submit event occurs, which can be captured by ‘**onsubmit**’ event handler.

The javascript code which validates the form elements can be given to the **onsubmit** event handler so that the form will be submitted only if the validation process is sucessful

Form Validation: Example

```
<html><head>
<title>Form Validation</title>
<script type="text/javascript">
function validate()
{
   if( document.myForm.Name.value == "" ){
         alert( "Please provide your name!" );
         document.myForm.Name.focus() ;
         return false;
   }
   if( document.myForm.EMail.value == "" ){
     alert( "Please provide your Email!" );
     document.myForm.EMail.focus() ;
     return false;
   }
```

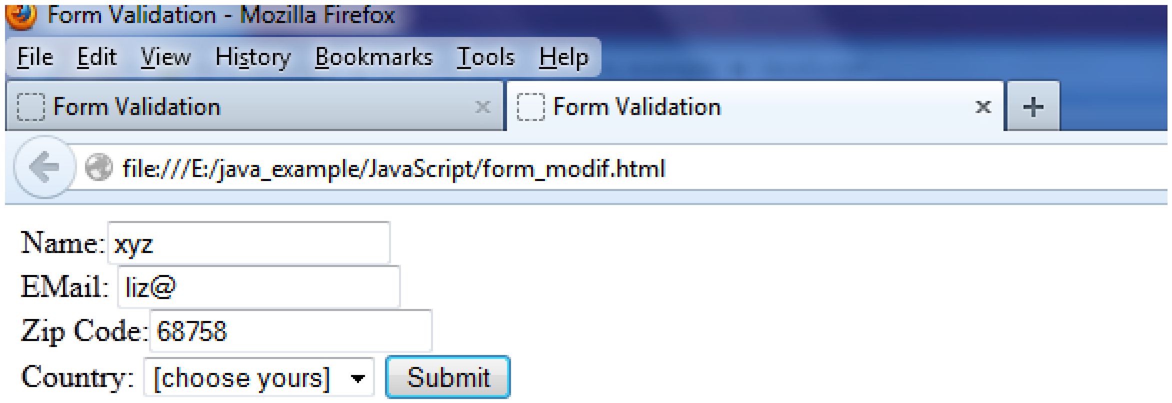
Form Validation: Example

```
if( document.myForm.Zip.value == "" ||
        isNaN( document.myForm.Zip.value ) ||
        document.myForm.Zip.value.length != 5 )
  {
     alert( "Please provide a zip in the format #####." );
     document.myForm.Zip.focus() ;
     return false;
  }
  if( document.myForm.Country.value == "-1" )
  {
    alert( "Please provide your country!" );
    return false;
  }
  return( true );
}
</script>
</head><body>
```

Form Validation: Example

```
<form action="/cgi-bin/test.cgi" name="myForm"
        onsubmit="return(validate());">
 Name:<input type="text" name="Name"/><br>
 EMail: <input type="text" name="EMail"/><br>
 Zip Code:<input type="text" name="Zip"/><br>
 Country:
 <select name="Country">
   <option value="-1" selected>[choose yours]</option>
   <option value="1">USA</option>
   <option value="2">UK</option>
   <option value="3">INDIA</option>
 </select>
   <td><input type="submit" value="Submit" />
 </form>
 </body>
 </html>
```

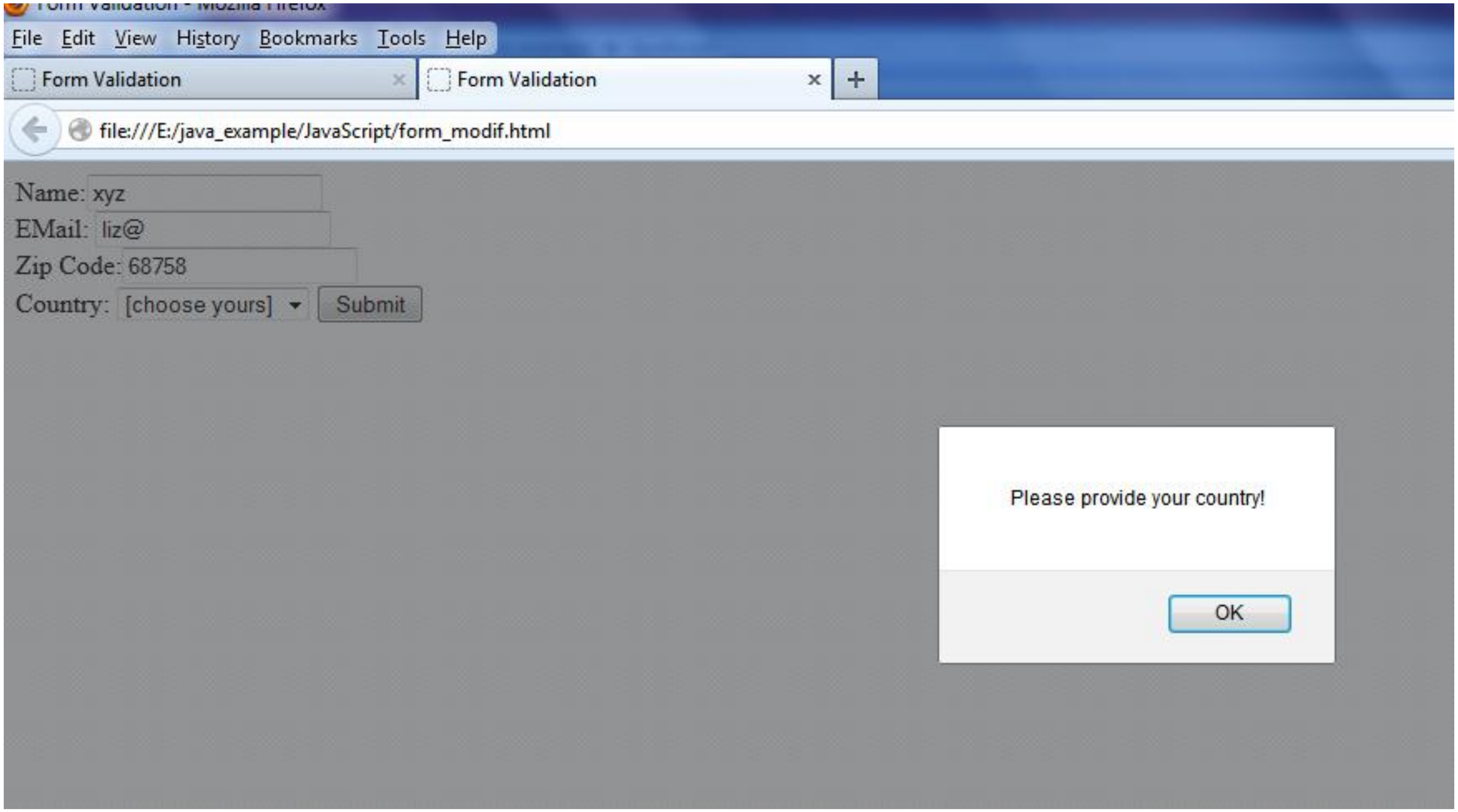
Form Validation: Example


Form Validation - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Form Validation
Form Validation
Submit

file:///E:/java_example/JavaScript/form_modif.html

Name: xyz
EMail: liz@
Zip Code: 68758
Country: [choose yours]

Form Validation: Example

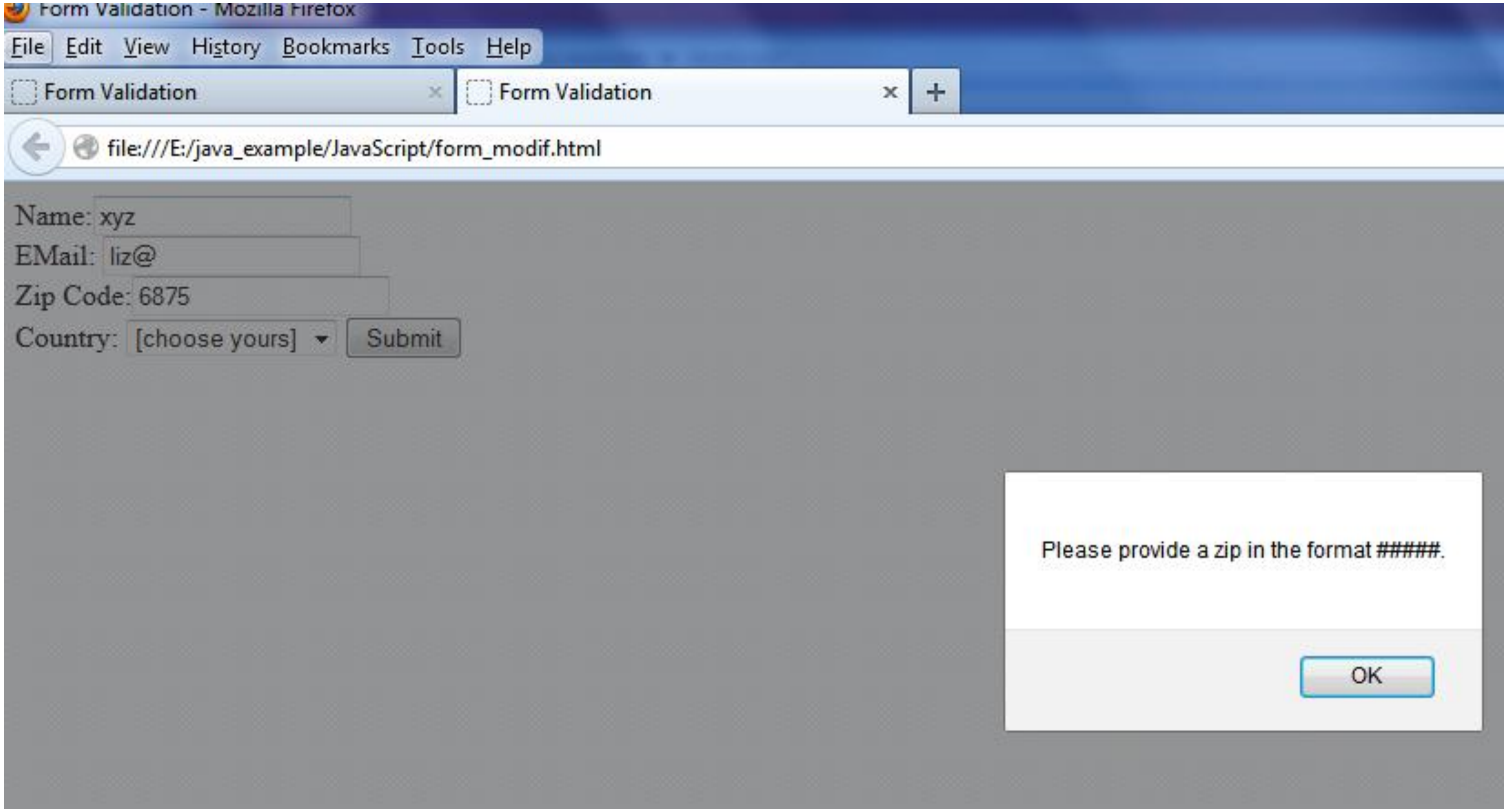

File Edit View History Bookmarks Tools Help
Form Validation
Form Validation
file:///E:/java_example/JavaScript/form_modif.html

Name: xyz
EMail: liz@
Zip Code: 68758
Country: [choose yours] Submit

Please provide your country!

OK

Form Validation: Example


Form Validation - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Form Validation
Form Validation
file:///E:/java_example/JavaScript/form_modif.html
Submit
OK

Name: xyz
EMail: liz@
Zip Code: 6875
Country: [choose yours]

Please provide a zip in the format #####.

Frames

- Frame divides a browser window into different blocks or areas.
- Each frame is a separate HTML document.
- The **<frameset>** tag defines how the frames are to be laid out on the screen.
- It has two attributes to do this, **cols**, which specifies how many columns to split the screen into, and **rows**, which specifies how many rows to split the screen into
- EXAMPLE

Frame1.html

```
<html>
<body>
    I am frame1.html
</body>
</html>
```

Frames

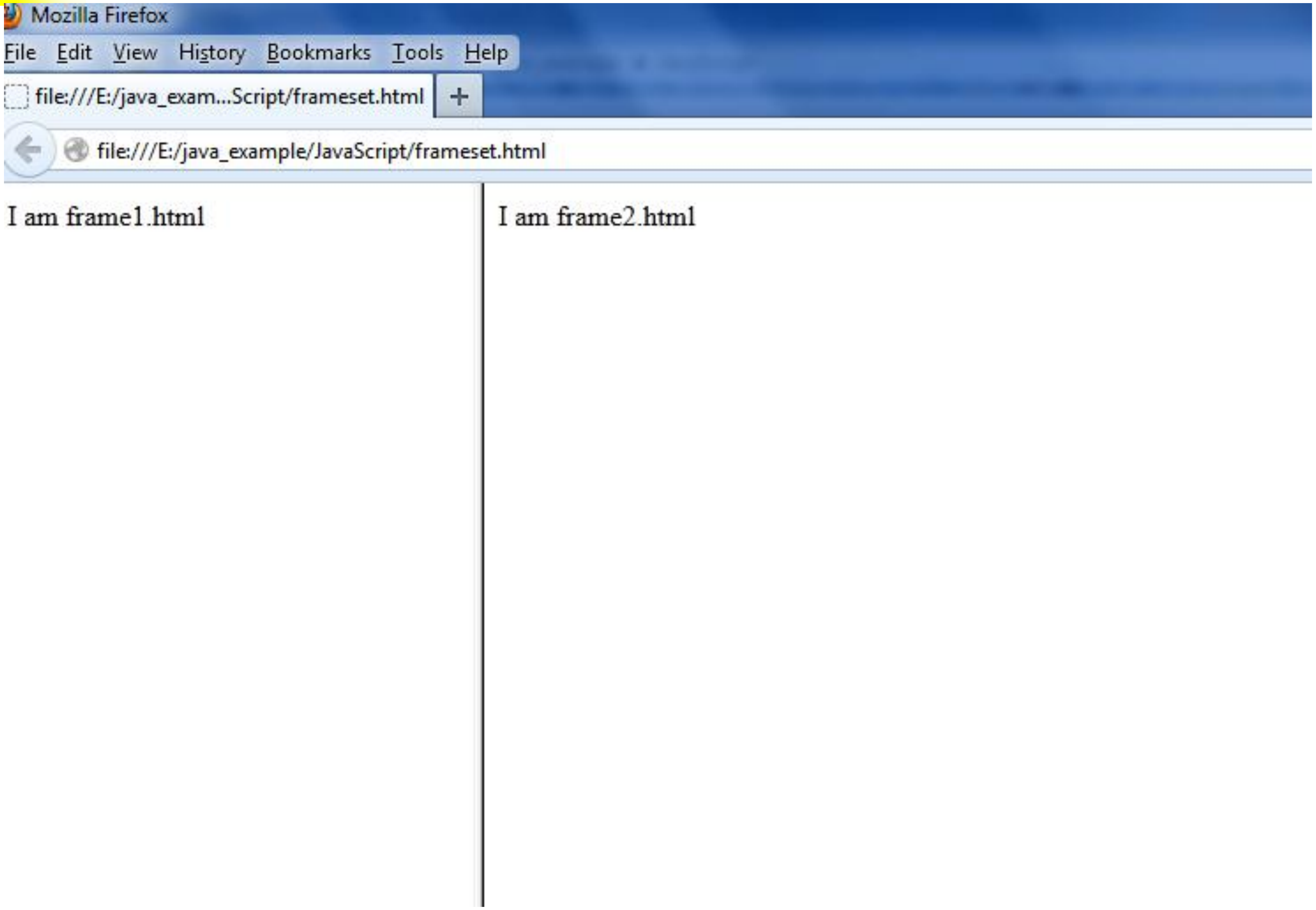
Frame2.html

```
<html>
<body>
        I am frame2.html
</body>
</html>
```

Frameset.html

```
<html>
<frameset cols="20%,80%">
    <frame src="frame1.html"></frame>
    <frame src="frame2.html"></frame>
</frameset>
</html>
```

Frames


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///E:/java_exam...Script/frameset.html
+

file:///E:/java_example/JavaScript/frameset.html

I am frame1.html

I am frame2.html

Nested Frames

- Framesets may be nested to any level.

Example

<FRAMESET rows="33%, 33%, 34%">

 ...contents of first frame...

  <FRAMESET cols="40%, 50%">

  ...contents of second frame, first row... ...contents of second frame, second row...

  </FRAMESET> *...contents of third frame...*

</FRAMESET>

Frames

- As far as JavaScript is concerned, each frame can be treated as a separate window. In fact, each frame has its own **Window** object.

- Each frame is stored as an element in the **frames[]** array of the **Window** object of the parent window for that frame

- If a web page doesn’t contain a frame its frames[] array is empty

To access frames using Java Script, you can use any one of the following options:

- **Use frames array of window object**
- **Name the frame in frame tag and use that name to access frame**

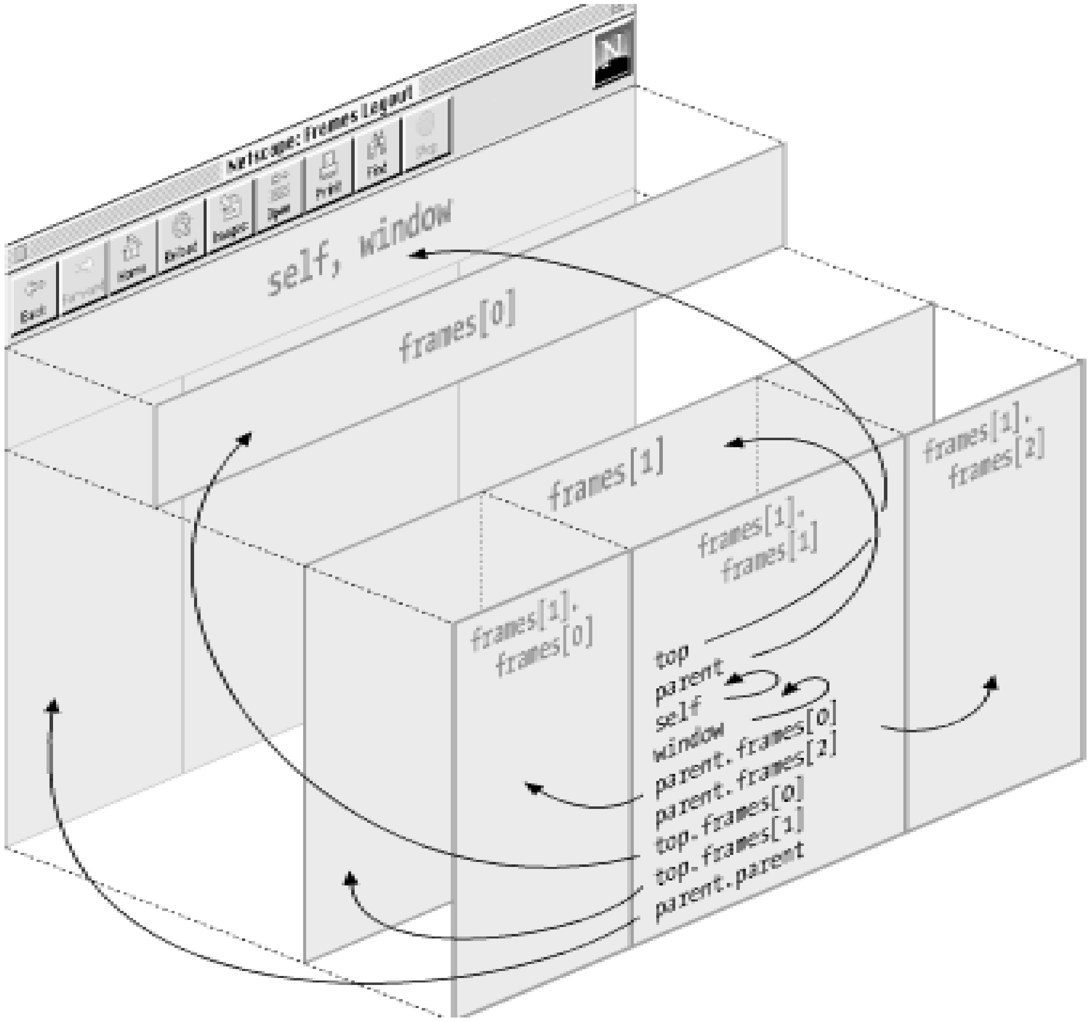
Frames

parent and top

- The Window object has properties **top** and **parent** which is specifically designed to address the situation of having a hierarchical arrangement of windows
- **window.parent** refers to the Window object of the parent window.

  Example: window.parent.frame2.*someElement*
- **window.top** is a reference to the **Window** object of the top window element in the window hierarchy
- If you have multiple levels of nested frames, you can get to the top of the frameset hierarchy

Frames


Netscape: Frames Layout
Back
Forward
Home
Reload
Images
Open
Print
Find
Stop
self, window
frames[0]
frames[1]
frames[1].frames[0]
frames[1].frames[1]
frames[1].frames[2]
top
parent
self
window
parent.frames[0]
parent.frames[2]
parent.frames[0]
top.frames[0]
top.frames[1]
top.parent
parent.parent

Java Script & Frames

We can use java script to pass values from one frame to another.

The following example changes the value of a text box in a frame on the right from a frame on the left side. First, set up a little frameset.

This will require 3 separate html pages:

frm1.html **frm2.html** **frm_script.html**

frm1.html

```
<HTML><HEAD>
<TITLE>JavaScript & Frames</TITLE>
</HEAD>
<BODY>
<FORM>
<INPUT type="button" value="What is cool?"
   onClick="parent.right_frame.document.form1.text1.value='Me!'">
</FORM></BODY></HTML>
```

Java Script & Frames

frm2.html

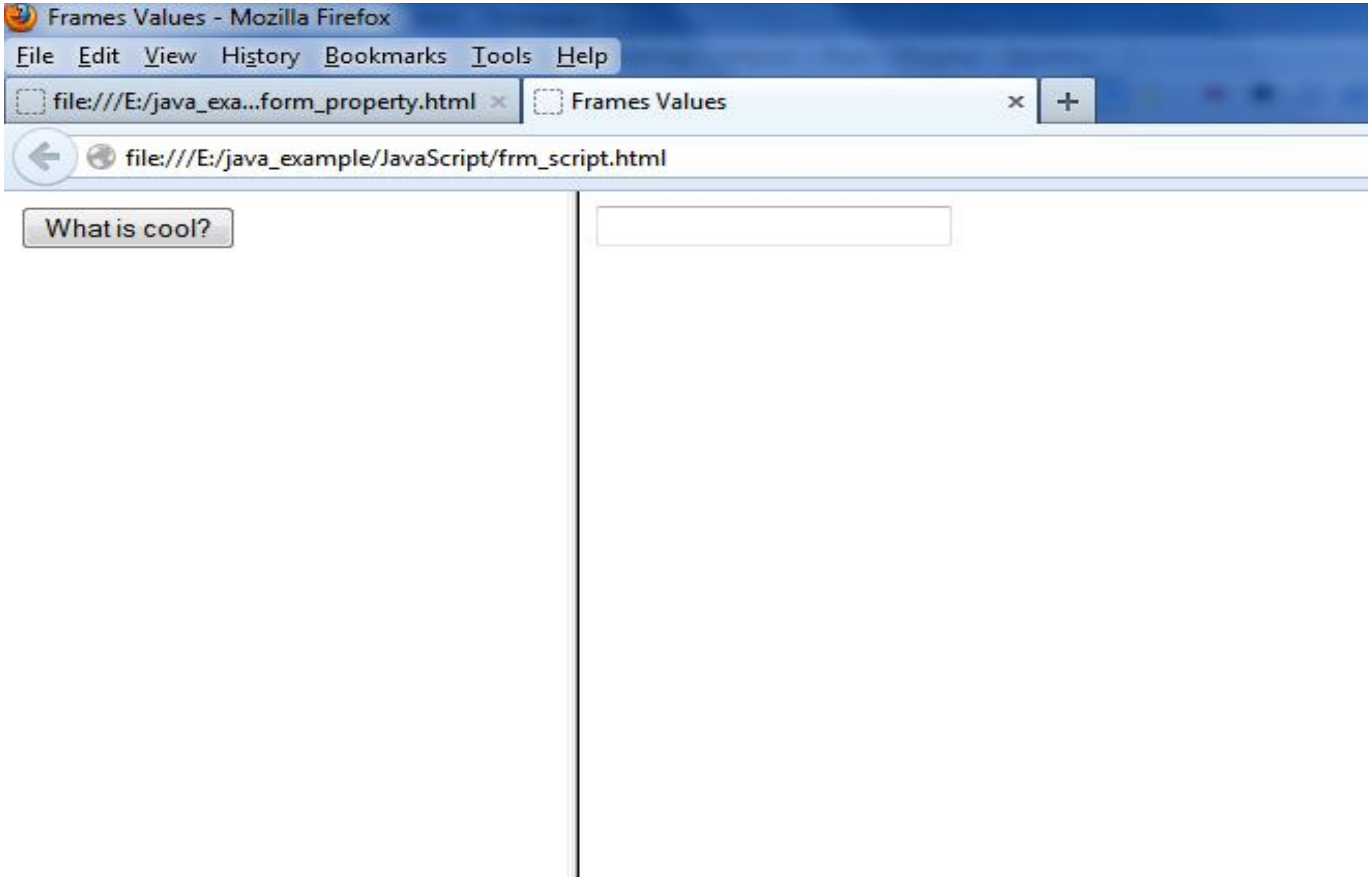
```
<HTML>
<HEAD>
<TITLE>JavaScript Example</TITLE>
</HEAD>
<BODY>
<FORM name="form1">
<INPUT type="text" name="text1" size="25" value="">
</FORM>
</BODY>
</HTML>
```

Java Script & Frames

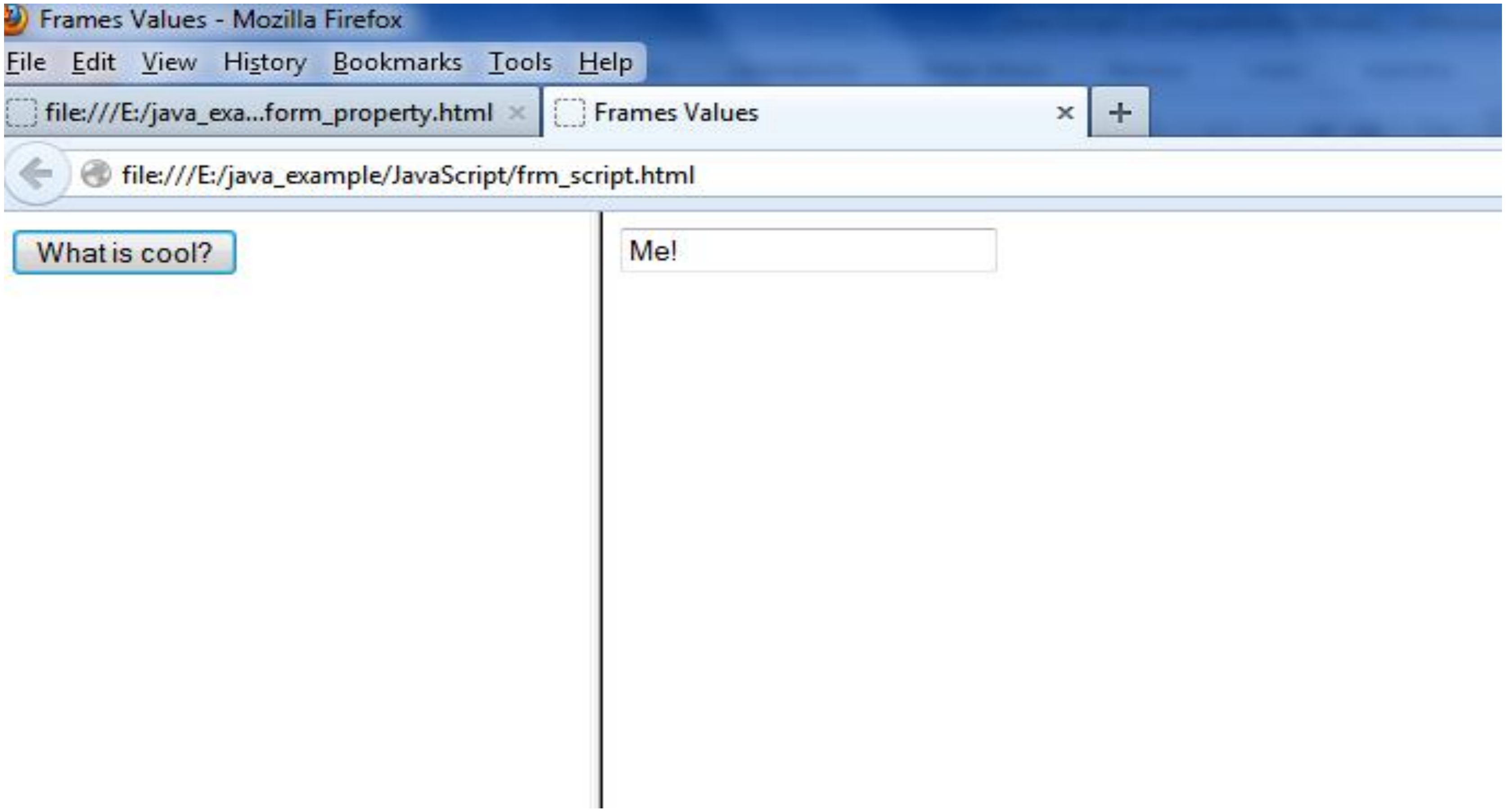
frm_script.html

```
<HTML>
<HEAD>
<TITLE>Frames Values</TITLE>
</HEAD>
<FRAMESET cols="20%,80%">
<FRAME SRC="frm1.html" name="left_frame">
<FRAME SRC="frm2.html" name="right_frame">
</FRAMESET>
</HTML>
```

Java Script & Frames


Frames Values - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///E:/java_exa...form_property.html
Frames Values
file:///E:/java_example/JavaScript/frm_script.html
What is cool?

Java Script & Frames


Frames Values - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///E:/java_exa...form_property.html
Frames Values
file:///E:/java_example/JavaScript/frm_script.html
What is cool?
Me!