JavaScript Language - Client Side

Introduction

- **JavaScript** is Netscape's cross-platform, object-oriented scripting language
- Standarized by EMCA, extended by Microsoft, IBM, Mozilla
- **Core JavaScript** contains a core set of objects, such as *Array*, *Date*, and *Math*, and a core set of language elements such as operators, control structures, and statements; it can be extended for a variety of purposes by supplementing with additional objects; for example:
  - **Client-side JavaScript** extends the core language by supplying objects to control a browser and its Document Object Model (DOM – W3C); e.g. client-side extensions allow an application to place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation
  - **Server-side JavaScript** (ex. Rihno engine) extends the core language by supplying objects relevant to running JavaScript on a server; e.g. server-side extensions allow an application to communicate with a relational database, provide continuity of information from one invocation to another of the application, or perform file manipulations on a server.

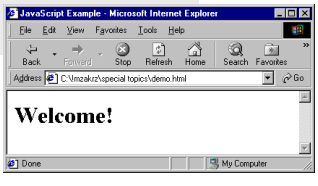
A Simple JavaScript Program

```
<HTML>
<HEAD>
<TITLE>JavaScript Example</TITLE>
<SCRIPT LANGUAGE="JavaScript">
  document.writeln("<H1>Welcome!</H1>");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

executed before <BODY> is displayed

document object represents the HTML document currently displayed in the browser

writeln method writes a line of HTML text in the HTML document being displayed



Displaying Messages

```
<HTML>
<HEAD>
<TITLE>JavaScript Example</TITLE>
<SCRIPT LANGUAGE="JavaScript">
  window.alert("Welcome to Wn JavaScript!");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

window object represents the browser's window

alert method displays an alert dialog box

Values and Data Types

- JavaScript recognizes the following types of values:
  - **numbers**, such as 42 or 3.14159
  - **logical** (Boolean) values, either true or false
  - **strings**, such as "hello!"
  - **null**, a special keyword denoting a null value; null is also a primitive value; since JavaScript is case sensitive, **null** is not the same as **Null**, **NULL**, or any other variant
  - **undefined**, a top-level property whose value is undefined; undefined is also a primitive value
- JavaScript is a dynamically typed language - no need to specify the data type of a variable while declaring it; data types are converted automatically as needed during script execution
  - x = "The answer is " + 42 // returns "The answer is 42"
  - "37" - 7 // returns 30

Variables

- You can declare a variable in two ways:
  - by simply **assigning** it a value; e.g. x = 42
  - with the keyword **var**; e.g. var x = 42
- A variable or array element that has not been assigned a value has the value **undefined**
- When you set a variable identifier by assignment outside of a function, it is called a **global** variable, because it is available everywhere in the current document; when you declare a variable within a function, it is called a **local** variable, because it is available only within the function
- Using **var** to declare a global variable is optional; however, you must use **var** to declare a variable inside a function

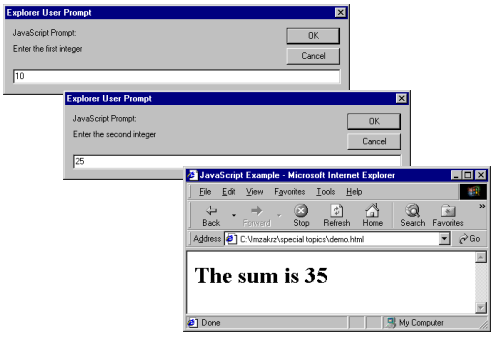
Using Variables (source)

prompt method displays a dialog to input a value

```
<HTML>
<HEAD>
<TITLE>JavaScript Example</TITLE>
<SCRIPT LANGUAGE="JavaScript">

 firstNumber = window.prompt("Enter the first integer","0");
 secondNumber = window.prompt("Enter the second integer","0");

 number1 = parseInt(firstNumber);
 number2 = parseInt(secondNumber);

 sum = number1 + number2;

 document.writeln("<H1>The sum is " + sum + "</H1>");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

explicitly convert character string datatype to integer

Using Variables (result)



Expressions and Operators

- Expressions and operators similar to C language; e.g. +, -, *, /, %, <<, >>, >>> (zero fill right shift), ++, --, +=, -=, *=, /=, %=, <<=, >>=, &, ^, |, ==, !=, === (string equal), !== (string not equal), >, <, >=, <=, &&, ||, !, condition ? val1:val2, op1, op2 (comma operator), delete (deletes an object), new (creates an object), this (referes to the current object), typeof (returns a string indicating the type of the operand), void

Statements

```
if (condition) {
statements1
}
[else {
statements2
} ]
```

```
switch (expression){
 case label :
   statement;
   break;
 case label :
   statement;
   break;
   ...
 default : statement;
}
```

```
with (object){
statements
}
```

```
do {
statement
} while (condition)
```

```
while (condition) {
statements
}
```

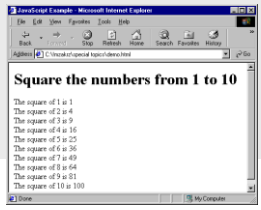
```
for (variable in object) {
statements }
```

```
label    break    continue    /* comments */    // comments
```

```
for ([initialExpression]; [condition]; [incrementExpression]) {
statements
```

Programmer-Defined Functions

```
<HTML>
<HEAD>
<TITLE>JavaScript Example</TITLE>
<SCRIPT LANGUAGE="JavaScript">
 document.writeln("<H1>Square the numbers from 1 to 10</H1>");

 for (x = 1; x <= 10; x++)
  document.writeln("The square of "+x+" is "+square(x)+ "</BR>");

 function square(y)
  {
    return y*y;
  }
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```



Objects

Accessing object attributes:

```
myCar.make = " Ford"
myCar.model = " Mustang"
myCar.year = 1969;
```

Object creation:

```
myNewCar = {make:" Honda" ,model: " Civic" , year: 1999}
```

Object creation with a constructor method:

```
myNextCar = new car(" Volvo" , " S70" , 1998)
```

Method definition:

```
function displayCar() {
  var result = "A Beautiful " + this.year + " " +
               this.make + " " + this.model
  pretty_print(result)
}

this.displayCar = displayCar;
```

Predefined Core Objects (1/2)

- **Array**, e.g.:

```
billingMethod = new Array(5);
billingMethod[1] = "cash";
myArray = new Array("Wind","Rain","Fire");
```

  - Array methods: concat, join, pop, push, reverse, shift, unshift, slice, splice, sort,
- **Boolean**, e.g.: `myBoleanObject = new Boolean(true)`
- **Date**, e.g.: `Xmas95 = new Date("December 25, 1995 13:30:00")`
  - Date methods: set*, get*, to, parse, UTC
- **Function**, e.g.:`var setBGColor = new Function( "document.bgColor='antiquewhite'")`
- **Math**, e.g.: `Math.sin(1.56)`
  - Math methods: abs, sin, cos, tan, acos, asin, atan, exp, log, ceil, floor, min, max, pow, random, round, sqrt

Predefined Core Objects (2/2)

- **String**, e.g.: `s1 = new String("foo")`
  - String methods:
    - HTML-oriented: anchor, big, blink, fixed, italics, small, strike, sub, sup, link
    - Character-oriented: charAt, charCodeAt, indexOf, lastIndexOf, concat, fromCharCode, split, slice, substring, substr, match, replace, search, toLowerCase, toUpperCase

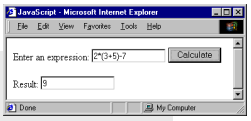
Embedding JavaScript in HTML

- You can embed JavaScript in an HTML document:
  - as statements and functions within a **<SCRIPT>…</SCRIPT>** tags in <HEAD> section
  - by specifying a file as the JavaScript source, e.g. **<SCRIPT SRC="filename.js"></SCRIPT>**
  - by specifying a JavaScript expression as the value of an HTML attribute, e.g. **<HR WIDTH="&{barWidth};%" ALIGN="LEFT">**
  - as event handlers within certain other HTML tags, e.g. **<INPUT TYPE ="button" VALUE="Press Me" onClick="myfunc('astring')">**
- JavaScript code is usually commented to hide it from old browsers
- HTML enclosed within a **<NOSCRIPT>** tag is displayed by browsers that do not support JavaScript

Handling Events

- Event handlers **names**: onAbort, onBlur, onChange, onClick, onDragDrop, onError, onFocus, onKeyDown, onKeyUp, onKeyPress, onLoad, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onMove, onReset, onResize, onSelect, onSubmit, onUnload
- Defining an **event handler**:
  - **<TAG eventHandler="JavaScript Code">**
  - example: `<INPUT TYPE="button" NAME="Button1" VALUE="Open Sesame!" onClick="window.open('mydoc.html', 'newWin')">`

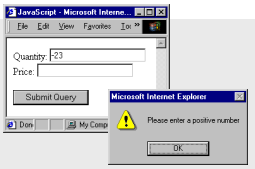
Handling Events - Example (1)




```
<HTML>
<HEAD>
<SCRIPT>
function compute() {
  myform.result.value = eval(myform.expr.value)
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="myform">
Enter an expression: <INPUT TYPE="text" NAME="expr" SIZE=15 >
<INPUT TYPE="button" VALUE="Calculate" onClick="compute()">
<BR><BR>
Result: <INPUT TYPE="text" NAME="result" SIZE=15 >
</FORM>
</BODY>
</HTML>
```

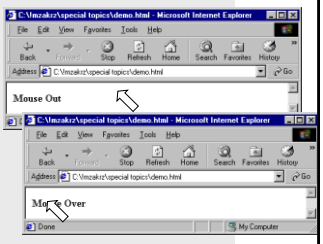
eval function evaluates the given arithmetic expression

Handling Events - Example (2)




```
<HTML>
<HEAD>
<SCRIPT>
function isaPosNum(s) {
  return (parseInt(s) > 0)
}
function value_check(item) {
 if (!isaPosNum(item.value))
   alert("Please enter a positive number");
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
Quantity: <INPUT TYPE="text" NAME="q" onChange="value_check(this)"><BR>
Price: <INPUT TYPE="text" NAME="p" onChange="value_check(this)"><BR><BR>
<INPUT TYPE="submit">
</FORM>
</BODY>
</HTML>
```

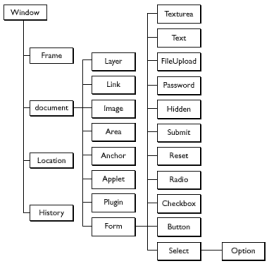
Handling Events – Example (3)




```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function m1()
{
  i1.innerText="Mouse Over";
}

function m2()
{
  i1.innerText="Mouse Out";
}

</SCRIPT>
</HEAD>
<BODY>
<B id="i1" onMouseOver="m1()" onMouseOut="m2()">Mouse Out</B>
</BODY>
</HTML>
```

Web Browser Objects Hierarchy

- When you load a document in web browser, it creates a number of JavaScript objects with property values based on the HTML in the document and other pertinent information; these objects exist in a hierarchy that reflects the structure of the HTML page itself

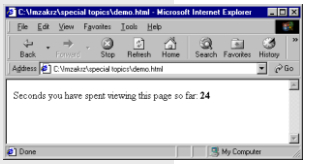


window and frame Objects

- Methods (selected):
  - **open** (opens browser window)
  - **close** (closes browser window)
  - **alert** (displays an Alert dialog box with a message)
  - **confirm** (displays a Confirm dialog box with OK and Cancel buttons)
  - **prompt** (displays a Prompt dialog box with a text field for entering a value)
  - **blur**, **focus** (removes focus from, or gives focus to a window)
  - **scrollTo** (scrolls a window to a specified coordinate)
  - **setInterval** (evaluates an expression or calls a function each time the specified period elapses)
  - **setTimeout** (evaluates an expression or calls a function once after the specified period elapses)
- Properties (selected):
  - **location** (to redirect the client to another URL)

window.setInterval() Example

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
var seconds = 0;

function startTimer(){
  window.setInterval("updateTime()", 1000);
}

function updateTime(){
  seconds++;
  soFar.innerText = seconds;
}

</SCRIPT>
</HEAD>
<BODY OnLoad="startTimer()">

<P>Seconds you have spent viewing this page so far:
<B ID="soFar">0</B></P>
</BODY>
</HTML>
```



document and form Objects

- Document Methods (selected):
  - **write**, **writeln**,
- Document Properties (selected):
  - **bgColor**, **fgColor**, **linkColor**, **alinkColor**, **vlinkColor**, **lastModified**, **referrer**, **cookie**
- Each form in a document creates a **form** object; a document can contain more than one form - **form** objects are stored in an array called **forms**; example: `document.forms[0]`
- The elements in a form, such as text fields, radio buttons, and so on, are stored in an **elements** array; example: `document.forms[0].elements[0]`

location and history Objects

- The **location** object has properties based on the current URL; it has two methods:
  - **reload** - forces a reload of the window's current document
  - **replace** - loads the specified URL over the current history entry
- The **history** object contains a list of strings representing the URLs the client has visited; properties: **current**, **next**, **previous**, methods: **back(), forward(), go()**; e.g. `history.go(-1)`

Working With Cookies

- The **document.cookie** property is a string that contains all the names and values of web browser cookies

```
function setCookie(name, value, expire) {
  document.cookie = name + "=" + escape(value)
  + ((expire == null) ? "" : ("; expires=" + expire.toGMTString()))
}
function getCookie(Name) {
  var search = Name + "="
  if (document.cookie.length > 0) { // if there are any cookies
    offset = document.cookie.indexOf(search)
  if (offset != -1) { // if cookie exists
    offset += search.length
  // set index of beginning of value
  end = document.cookie.indexOf(";", offset)
  // set index of end of cookie value
  if (end == -1)
  end = document.cookie.length
  return unescape(document.cookie.substring(offset, end))
}}}
```

Working With Cookies - Example

```
<BODY>
<H1>You have visited this page
<SCRIPT>
var counter = getCookie("Cnt");
if (counter != null) {
  document.write(counter," times!");
  setCookie("Cnt", parseInt(counter)+1);}
else {
  setCookie("Cnt", 1);
  document.write("for the very first time!");}
</SCRIPT>
</BODY>
```