Web Programming

Lecture 3 – JavaScript Basics

Origins

- JavaScript was originally developed by Netscape as "LiveScript."
- In 1995, Sun and Netscape worked on it and its name became "JavaScript."
- It became standardized in the late 1990s (currently in version 3of ECMA 262).
- Netscape 7 and Internet Explorer 6 both support ECMA 262 v. 3.

The Three Parts of JavaScript

- JavaScript has three parts:
  - ***<u>Core</u>*** –the heart of the language, including operators, expressions, statements and subprograms.
  - ***Client side*** – collection of objects that support control of the browser and interactions with users (e.g., mouse clicks).
  - ***Server side*** – collections of objects that make the language useful on a Web server, e.g., supporting communication with a database management system.

JavaScript and Java

- Although their names are similar, Java and JavaScript are very different.
  - JavaScript's object model is very different from C++ and Java. JavaScript does NOT support the object-oriented software development paradigm.
  - Java is a strongly-typed language; JavaScript is dynamically typed and does not necessarily require declarations
  - Java objects are static (their collections of methods and properties are fixed at compile time); JavaScript's objects are dynamic.
  - Their biggest similarity is syntax.

Uses of JavaScript

- JavaScript was developed originally to provide programming capability for the Web.
- Client-side JavaScript allows the client to do a lot of tasks that would otherwise have to be done on a potentially overtaxed server.
- JavaScript is an alternative to Java applets.
- Interaction with form elements such as button and menus can be specified in JavaScript.
- The Document Object Model (DOM) allows JavaScript to access and modify CSS properties and XHTML document content, making XHTML documents dynamic.

Event-Driven Computation

- Event-driven computation – the user interacts with the elements of the display (e.g., command buttons, text boxes, etc.)
- One common use is to have the script check the validity of entries made in forms.
- Doing this on the client side saves execution time on the server.

Browsers and XHTML/JavaScript Documents

- If the browser reads an (X)HTML document, it displays the content using information provided by the accompanying tags.
- When a browser encounters a JavaScript script, it executes the script before returning to the tasks of displaying the document.
- Scripts producing content only when requested (or reacting to an "event") appear in the head of an XHTML document.
- Scripts that are interpreted just once (when loading) appear in the document's body.

Object Orientation and JavaScript

- JavaScript is an "object-based" language; i.e., it does objects and models of objects (but not classes).
- While there is prototyped–based inheritance, there is not real class-based inheritance.
- JavaScript does not support polymorphism.

JavaScript Objects

- JavaScript objects are collections of properties, which can be data properties or method properties.
- Data properties are either primitive values or references to other objects.
- All objects in a JavaScript program are accessed indirectly though variables.
- All other objects are specializations of the root object **`Object`**.

General Syntactic Characteristics

- JavaScript scripts are all indicated by the use of **`<script>`** tag.
- An internal script would indicated one way:

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

- An external script would be indicated another way:

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

- JavaScript identifiers are case-sensitive.

JavaScript Keywords

`break`	`delete`	`function`	`return`	`typeof`
`case`	`do`	`if`	`switch`	`var`
`catch`	`else`	`in`	`this`	`void`
`continue`	`finally`	`instanceof`	`throw`	`while`
`default`	`for`	`new`	`try`	`with`

NB – other words that are reserved for future use can be found at **http://www.ecma.ch**

JavaScript Comments

- Comments in JavaScript

```
// can be short
/* they can also be multiline
or even longer*/
```

- JavaScript scripts are usually embedded in XHTML comments to avoid problems with older browsers and XHTML validators:

```
<!--
   JavaScript script
// -->
```

JavaScript Syntax

- There are other problems that can come from placing JavaScript scripts in XHTML comments. For this reasons, it is usually a better idea to place them in separate files.
- While JavaScript does not require semi-colons (it normally assumes that the end-of-line ends the statements, using semi-colons can avoid some potential problems.

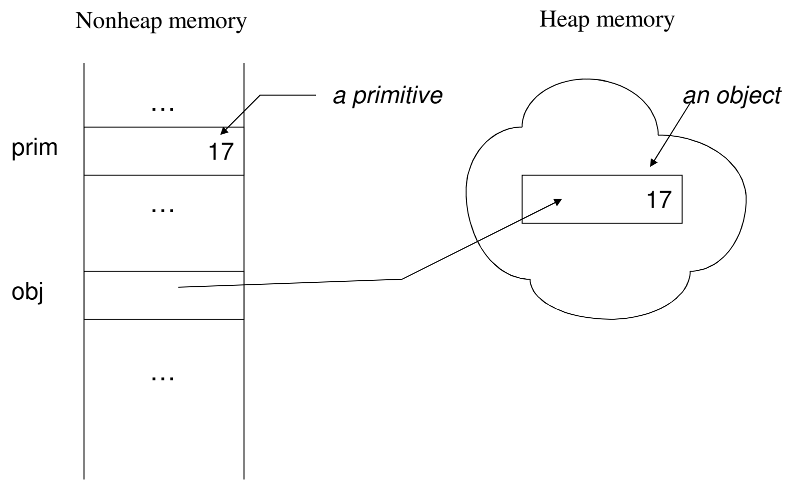
Primitive Types

- JavaScript has five primitive types:

Number	String
Boolean	Undefined
Null	

- The primitive types number, string and boolean have wrapper objects associated with them (called **`Number`**, **`String`** and **`Boolean`**).
- The wrapper objects provide properties and methods that can be convenient for a JavaScript programmer.
- Java Script will occasionally coerce between the Number data and String data items and their corresponding objects.

Primitives and Objects



Numeric Literals

- All numeric values are represented internally as double-precision floating point values.
- Literal numbers are ***integers*** or ***floating point values***.
  - Integers are strings of digits
  - Integers can be written in hexadecimal by preceding them with `0x`
  - Floating point values can have decimal points and/or exponents:

    72 7.2 .72 72. 7E2 7e2

    .7e2 7.e2 7.2e-2

String Literals

- String literals can be enclosed in single quotes (`'`) or double quotes (`"`) and can contain 0 or more characters.
- They can include escape sequences such as **`\n`** and **`\t`**
- A backslash can be used to enclose quotes inside a string:

```
"\"This isn\'t the first time,\", he said."
```

- 2 backslashes place a backslash in the string:

```
"D:\\bookfiles"
```

Booleans

- Boolean variables have two potential values: true and false.
- These can be computed by evaluating a relational expression or a Boolean expression.

`null` And `undefined`

- The only value that type Null can take is **`null`**, which means that it does not refer to an object.
  - As a Boolean, **`null`** is **`false`**.
  - As a Number, **`null`** is **`0`**.
- Undefined's only value is undefined. It is assigned when a variable has not been assigned a value or does not exist.
  - As a Boolean, **`undefined`** is **`false`**.
  - As a Number, **`undefined`** is **`0`**.

Declaring Variables

- Since JavaScript variables are dynamically typed, any variable can be used for anything.
- The interpreter will usually convert the variable's type to whatever it needs.
- A declaration has the keyword var followed by a list of variable name (that may or may not include an initial value):

```
var counter, index,
    pi = 3.14159265,
    quarterback = "John Elway",
    stop_flag = true;
```

- While explicit declarations are not required,the are recommended.

Numeric Operators

- JavaScript has the usual operators:
  - Binary operators such as addition (+), subtraction (-), multiplication (*), division (/) and modulus (%)
  - Unary positive (+) and negative (-) signs
  - Increment and decrement operators that can be prefix or postfix.
- Example (assume a is set to 7)

```
(++a) * 3 // Expression = 24, a is now 8
(a++) * 3 // Expression = 21, a is now 8
```

Precedence and Associativity

- Precedence rule specify which operator is evaluated first when two or more operators are in an expressions.

  `a * b + 1` **`*`** has precedence over **`+`**

- Associativity rules specify which operator is evaluated first when two operators share the same precedence.

  `a + b + c` we add **`a + b`** and then add **`c`** to the sum

Precedence and Associativity in JavaScript

Operators	**Associativity**
`++, --, unary -`	Right to left (though it is irrelevant)
`*, /, %`	Left to right
`Binary +, Binary -`	Left to right

Parentheses can be used to force any desired precedence:

```
(a + b) * c
```

Operators in JavaScript: Example

```
var a = 2,
    b = 4,
    c,
    d;
c = 3 + a * b;  // c is now 11

d = b / a / 2;   // d is now 1
```

The Math Object

- The Math object contains a collection of properties of Number objects and methods that operate on Number objects

`Math.abs()`	Absolute Value
`Math.ceil()`	Rounded up to the nearest integer
`Math.cos()`	Cosine
`Math.exp()`	e^x
`Math.pow()`	x^n
`Math.random()`	Random number
`Math.round()`	Rounded to the nearest integer

The Number Object

Property	Meaning
`MAX_VALUE`	Largest representable number
`MIN_VALUE`	Smallest representable number
`NaN`	Not a number
`POSITIVE_INFINITY`	Special value to represent infinity
`NEGATIVE_INFINITY`	Special value to represent negative infinity
`PI`	The value of π
`isNaN`	True if not a number; false if it is a number
`toString`	Returns a string representation of the value

Example
```
var x = Number.MIN_VALUE;
```

Example
```
var price = 427, str_price;
str_price = price.toString
```

The String Concatenation Object

- Strings are not arrays in JavaScript; they are considered scalar values.
- Concatenation is indicated by the plus (+) operator.
- Example

```
var first = "Freddy";
first = first + " Freeloader"
```

Implicit Type Conversions

- Coercions – implicit type conversions.
- When a value of one type is used in a situation where a value of another type is needed, JavaScript attempts to convert the value to the needed type.
- Example

```
"August " + 1977 → "August 1977"
7 * "August " → NaN
7 * "3" → 21
```

Explicit Type Conversions

There are several ways of forcing conversions:

- Using the **`String`** constructor

```
var str_value = String(value);
```

- Using the **`toString`** method:

```
var num = 6;
var str_value = num.string(); // = 6
var stValueBinary = num.toString(2);//110
```

- By concatenating to an empty string:

```
num_string = num + ""
```

Converting String to Number

- Subtracting a value from a string containing a number:

```
var aString = "4"
var number = aString - 0
```

- This has a severe limitation – the number within the string cannot be followed by anything but a blank.
- Using **`parseInt`** and **`parseFloat`** get around this limitation, however, the number is expected at the beginning of the string (otherwise the result is **`NaN`**).

String Properties and Methods

Method	Parameters	Result
`length`*	None – property	The length of the string
`charAt()`	A number	The character in the String object that is at the specified position
`indexOf()`	One-character string	The position in the String object of the parameter
`substring()`	Two number	The substring of the String object from the first parameter position to the second
`toLowerCase()`	None	Convert any uppercase letters in the string to lowercase
`toUpperCase()`	None	Convert any lowercase letters in the string to uppercase

String Methods – An Example

`var str = "George"`

`str.charAt(2)` is '`o`'

`str.index.Of('r')` is `3`

`str.substring(2, 4)` is '`org`'

`str.toLowerCase()` is '`george`'

The **`typeof`** Operator

- The typeof operator returns the type of the operand.
  - If the operand is Number, String or Boolean, it returns `"number"`, `"string"` or `"boolean"` respectively.
  - If the operand is an object or null, it returns `"object"`.
  - If the operand is undefined, it returns `"undefined"`.
- Example

  `var x = 5;`

  `typeof(x)` is `"number"`

Assignment Statements

- JavaScript has the same range of assignment operators as C, e.g., **`+=`**, **`-=`**, **`*=`**, **`/=`** , etc.
- Example

  `a += 7;`

  is equivalent to

  `a = a + 7;`

Methods For The Date Object

Method	Returns
`toLocaleString()`	A string of the `Date` information
`getDate()`	The day of the month, from 1 to 31
`getMonth()`	The month of the year, from 1 to 12
`getDay()`	The day of the week from 0 to 6
`getFullYear()`	The year
`getTime()`	The number of milliseconds since January 1, 1970
`getHours()`	The number of the hour, from 0 to 23
`getMinutes()`	The number of the minute, from 0 to 59
`getSeconds()`	The number of the second, from 0 to 59
`getMilliseconds()`	The number of the millisecond from 0 to 999

The Date Object - An Example

```
var today = new Date()
var thisMonth = today.getMonth()
var thisDay = today.getDay()
var thisYear = today.getFullYear()
```

Screen Output and Keyboard Input

- The normal output screen for a JavaScript script is the browser window containing the XHTML document in which the script is embedded.
- This window is an object in JavaScript, which has a constituent object called **`document`**, of which **`write()`** is the most interesting method.
- The output produced by **`write()`** will usually be punctuated with XHTML tags.

`document.write()` – An Example

```
document.write("The result is: ", x, "<br />");
```

appears in the browser window:

```
The result is:  42

_
```

alert()

- **`alert()`** opens a dialogue window and displays its parameter as a message, along with an "**OK**" button.
- The parameter string is plain text (not XHTML); it may include **\n** but not **`<br />`**.
- Example

```
alert("The sum is" + sum + "\n");
```

confirm()

- **`confirm()`** opens a dialog window in which it displays its string parameter, along with an **`OK`** button and a **`Cancel`** button.
- confirm returns **`true`** (for **`OK`**) or **`false`** (for **`Cancel`**).

```
var question =
  confirm("Do you want to continue this download?");
```

prompt()

- **prompt()** creates a dialogue window that contains a text box, which collects an input string which it returns as its value.
- **prompt()** also has **OK** and **Cancel** button. It returns a default string (often empty) if either button is pressed without entering a string.
- **prompts()** has 2 parameters: a prompt for the user and the default string.
- Example

```
name = prompt("What is your name", "");
```

firstjs.html

```
<!DOCTYPE html>

<!-- roots.html
   Compute the real roots of a given quadratic
   equation.  If the roots are imaginary, this
   script displays NaN, because that is what results
   from taking the square root of a negative number.
  -->

<html lang = "en">
  <head> <title> Real roots of a quadratic equation
  </title>
    <meta charset = "utp-8">
  </head>
  <body>
    <script type = "text/javascript">
      <!--

  // Get the coefficients of the equation from the
  user

      var a = prompt("What is the value of 'a'?\n",
                "");
      var b = prompt("What is the value of 'b'?\n",
                "");
      var c = prompt("What is the value of 'c'?\n",
                "");

  // Compute the square root and denominator
  // of the result
      var rootpart = Math.sqrt(b * b - 4.0 * a * c);
      var denom = 2.0 * a;
```

```
  // Compute and display the two roots
      var root1 = (-b + rootpart) / denom;
      var root2 = (-b - rootpart) / denom;

      document.write("The first root is : ", root1,
                  "<br />");
      document.write("The second root is : ", root2,
           "<br />");
  // -->
    </script>

  </body>
</html>
```

Control Statements

- Control statements are handled in JavaScript in a manner similar to C/C++/Java, including the use of braces to build compound statements.
- There are control expressions which are essentially true or false and determine the course of action to be taken
- There are selection statements (**`if`**, **`if-else`** and **`switch`**).
- There are loop statements (**`while`**, **`for`** and **`do..while`**)that allow an action or a series of action to occur.

Control Expressions

- Control expressions determine which action is to be performed and they are interpreted as **`true`** or **`false`**. A string is **`true`** unless it is empty(`""`) or zero (`"0"`). A non-zero numeric value is **`true`**; zero is **`false`**.
- If two operands in a relational expression are not of the same type, and the operator isn't === or !==, JavaScript will try to convert them to a single type.
  - If the operands are a string and a number, it will try to convert the string to a number.
  - If the only one of the operands is boolean, it will convert the boolean value to a number (`1` for true, `0` for false).

Relational Operators

Operations	**Operator**
Is equal to	`==`
Is not equal to	`!=`
Is less than	`<`
Is greater than	`>`
Is less than or equal to	`<=`
Is greater than or equal to	`>=`
Is strictly equal to	`===`
Is not strictly equal to	`!==`

`if-else` Statements

- `if` and `if-else` statements in JavaScript look like those in most modern languages.

```
if (a > b)
  document.write("a is greater than b.");
else
  document.write("a is greater than b.");
```

Operator Precedence Associativity

Operators	**Associativity**
`++`, `--`, unary -	Right to left
`*`, `/`, `%`	Left to right
`+`, `-`	Left to right
`>`, `<`, `>=`, `<=`	Left to right
`==`, `!=`	Left to right
`===`, `!===`	Left to right
`&&`	Left to right
`||`	Left to right
`=`, `+=`, `-=`, `*=`, `/=`, `&&=`, `||=`, `%=`	Right to Left

The **`switch`** Statement

- The **`switch`** statement works similar to the one in C.

```
switch(expression)  {
  case value1:
   //Statement(s);
case value2:
  // Statement(s);
default:
  //Statement(s);
}
```

- The expression is evaluated and the program jumps to the appropriate case, falling through from one case to another if there is no **`break`** statement.

borders2.html

```
<!DOCTYPE html>

<!-- borders2.html
     An example of a switch statrement for table
  border
     size selection
     -->

<html lang = "en">
  <head> <title> A switch statement </title>
    <meta charset = "utp-8">
  </head>
```

```
  <body>
    <script type = "text/javascript">
      <!--
        var bordersize;
        bordersize
          = prompt("Select a table border size \n" +
                         "0  (no border) \n" +
                         "1 (1 pixel border) \n" +
                         "4 (4 pixel border) \n" +
                         "8 (8 pixel border) \n");
```

```
switch (bordersize)  {
  case "0": document.write("<table>");
            break;
  case "1": document.write
                ("<table border = \'1\'>");
            break;
  case "4": document.write
                ("<table border = \'4\'>");
            break;
  case "8": document.write
                ("<table border = \'8\'>");
            break;
  default:  document.write
                ("Error - invalid choice: "
          +     bordersize + "<br />");
}
```

```
document.write
    ("<caption> 2003 NFL Divisional"+
                "Winners </caption>");

document.write("<tr>",
         "<th />",
         "<th> American Conference </th>",
         "<th> National Conference </th>",
         "</tr>",
         "<tr>",
         "<th> East </th>",
         "<td> New England Patriots </td>",
         "<td> Philadelphia Eagles </td>",
         "</tr>",
         "<tr>",
         "<th> North </th>",
         "<td> Baltimore Ravens </td>",
         "<td> Green Bay Packers </td>",
         "</tr>",
                    "<tr>",
                    "<th> West </th>",
                    "<td> Kansas City Chiefs </td>",
                    "<td> St. Louis Rams </td>",
                    "</tr>",
                    "<tr>",
                    "<th> South </th>",
                    "<td> Indianapolis Colts </td>",
                    "<td> Carolina Panthers </td>",
                    "</tr>",
                    "</table>");

          // -->
       </script>
    </body>
</html>
```

Loop Statements

- JavaScript has **`while`**, **`for`** and **`do`**..**`while`** statements that are similar to those in C/C++/Java.

  `while (`*control expression***`)`**

      statement or compound statement
- JavaScript also has a **`do`**..**`while`** statement where the test is at the end:

```
  do  {
      count ++;
      sum = sum + (sum * count);
  } while count <= 50;
```

`for` Statements

- The general form of the for statement is:

  `for` `(`*initExpr*`;` *cntrlExpr*`;` *incrExpr*`)`

  statement or compound statement
- Both the initial expression and the increment expression can be two expressions separated by a comma
- Example

```
var sum =0, count;
for (count = 0;  count <= 10; count++)
  sum += count;
```

`date.html`

```
<!DOCTYPE html>

<!-- date.html
     Illustrates the use of the Date object by
     displaying the parts of a current date and
     using two Date objects to time a calculation
     -->

<html lang = "en">
  <head> <title> Illustrates Date </title>
    <meta charset = "utp-8">
  </head>
```

```
<body>
  <script type = "text/javascript">
    <!--
     // Get the current date

     var today = new Date();

     // Fetch the various parts of the date

     var dateString = today.toLocaleString();
     var day = today.getDay();
     var month = today.getMonth();
     var year = today.getFullYear();
     var timeMilliseconds = today.getTime();
     var hour = today.getHours();
     var minute = today.getMinutes();
     var second = today.getSeconds();
     var millisecond = today.getMilliseconds();
```

```
     // Display the parts
     document.write(
       "Date: " + dateString + "<br />",
       "Day: " + day + "<br />",
       "Month: " + month + "<br />",
       "Year: " + year + "<br />",
       "Time in milliseconds: "
               + timeMilliseconds + "<br />",
       "Hour: " + hour + "<br />",
       "Minute: " + minute + "<br />",
       "Second: " + second + "<br />",
       "Year: " + year + "<br />",
       "Millisecond: " + millisecond + "<br />");

      // Time a loop
      var dum1 = 1.00149265, product = 1;
      var start = new Date();
```

```
        for (var count = 0;  count < 10000; count++)
          product = product
                   + 1.000002 * dum1 / 1.00001;

        var end = new Date();
        var diff = end.getTime() - start.getTime();
        document.write("<br /> The loop took "
                   + diff + " milliseconds <br />");
      // -->
    </script>
  </body>
</html>
```

Object Creation and Modification

- Objects are created with a new expression which includes a call to the constructor. In JavaScript, constructors creates and initializes the object's properties.

  ```
   var myObject = new Object();
  ```

  `myObject` does not have properties yet.
- JavaScript objects can have properties added or deleted at any time.

  ```
  //Create an Object object
  var myCar = new Object();
  // Create and initialize the make property
  myCar.make = "Ford";
  // Create and initialize model
  myCar.model = "Contour SVT":
  ```

Objects and Their Properties

- Properties of object can become objects themselves:

```
myCar.engine = new Object();
myCar.engine.config = "V6";
myCar.engine.hp = 200;
```

- Object properties can be accessed using the dot operator or as a subscript of the object to which they belong:

```
var prop1 = myCar.make;
var prop2 = myCar["make"];
```

- Objects can be deleted using **`delete`**:

```
delete myCar.model;
```

`for`..**`in`** Statement

- JavaScript has a loop statement that allows the listing of properties in an object.
- The syntax:

  `for (`identifier **`in`** object**`)`** statement(s)

- Example

```
for (var prop  in myCar)
   document.write("Name: ", prop,
      "; Value: ", myCar[prop], "<br />");
```

Arrays

- Arrays in JavaScript, like in Java, are special cases of objects.
- JavaScript arrays have dynamic length.

Array Object Creation

- JavaScript arrays can be created in a few ways.
- If the new statement has a single parameter, it is assumed to be the size of the array:

```
var myList = new Array(100); //uninitialized
```

- If there is more than one parameter they are taken to be initial values:

```
var yourList = new Array(1, 2, "three", "four");
```

- You can initialize an array without even using new:

```
var myOtherList = [1, 2, "three", "four"];
```

Characteristics of Array Objects

- All arrays in JavaScript have indices in the range of `0` to *n-1,* where *n* is the number of items in the array.
- The array will have m elements if m-1 is the highest index for which there is a value assigned, i. e., if your script includes:

  ```
  myList[47] = 2222;
  ```

  there are at least 48 elements in the array, although they may not all be assigned.
- The length of an array is a user-readable and writeable property. Adding the statement

  ```
  myList.length = 1002;
  ```

  set the length at 1002,whether all these values are defined or not.
- Only array elements to which values are assigned have storage allocated for them.

insert_names.html

```
<!DOCTYPE html>

<!-- insert_names.html
     The script in this docuument has an array of
     names, nameList, whose values are in
     alphabetical order.  New names are input
     through prompt.  Each new name is inserted
     into the name array, after which the new
     list is displayed.
     -->

<html lang = "en">
  <head> <title> Font properties </title>
  </head>
```

```
<body>
  <script type = "text/javascript">
  <!--
  // the original list of names

    var nameList = new Array("Al", "Betty",
                   "Kasper", "Michael", "Roberto",
                   "Zimbo");
    var newName, index, last;

  // Loop to get a new name and insert it

    while (newName
        = prompt("Please type a new name", "")) {

     // Loop to find the place for the new name
       last = nameList.length - 1;
```

```
       while (last >= 0
                 && nameList[last] > newName)  {
         nameList[last + 1] = nameList[last];
         last--;
       }

       //  Insert the new name into its spot in
       //  the array
      nameList[last + 1] = newName;

      // Display the new array
      document.write("<p> <b> The new name list",
                " is: </b> <br />");
```

```
        for (index = 0;  index < nameList.length;
                                            index++)
          document.write(nameList[index], "<br />");
        document.write("</p>");
      }

   // -->
      </script>
  </body>
</html>
```

Array Methods

- JavaScript arrays have several methods that can be useful, including:

```
join()              reverse()
sort()              concat()
slice()             toString()
push()              pop()
unshift()           shift()
```

`join()`

- **`join`** converts all of the elements of an array into string and concatenates them into a single string.
- If there are no parameters, the values in the string are separated by a comma. If there is a parameter, that string is used as a separator.
- Example

```
var names = new Array("Mary", "Murray", "Murphy",
  "Max");
var nameString = names.join(":");
```

produces "**`Mary : Murray : Murphy : Max`**"

`reverse()`

- **`reverse`** reverses the order in which elements appear in the array.
- Example

```
var names = new Array("Mary", "Murray",
  "Murphy", "Max");
names = names.reverse();
```

produces "**`Max"`**, **`"Murphy"`**, **`"Murray"`**, **`"Mary`**"

`sort()`

- sort converts all the elements of the sarray into strings and places them in alphabetical order.

```
var names = new Array("Mary", "Murray",
  "Murphy", "Max");
names = names.sort();
```

produces **"Mary"**, **"Max"**, **"Murphy"**, **"Murray"**

`concat()`

- **`concat`** concatenates its parameters to the end of the array.
- Example

```
var names = new Array("Mary", "Murray",
  "Murphy", "Max");
names = names.concat("Moo", "Meow");
```

produces **"Mary"**, **"Max"**, **"Murphy"**, **"Murray"**, **"Moo"**, **"Meow"**

`slice()`

- **`slice`** returns the part of the array specified by its subscripts.
- If there are two subscripts **`i`** and **`j`**, the new array includes from **`array [i]`** up to but not including **`array[j]`**
- If there is one subscript **`i`**, the new array includes from **`array[i]`** to the end of the array.
- Example

```
var list = [2, 4, 6, 8, 10];
list2 = list.slice(1, 3);
```

produces [4, 6]

`toString()`

- toString converts the elements of a array into one string where commas separates the elements.
- Example

```
<script type="text/javascript">
  var arr = new Array(3);
  arr[0] = "Jani";
  arr[1] = "Hege";
  arr[2] = "Stale";
  document.write(arr.toString());
</script>
```

produces Jani,Hege,Stale

push() and **pop()**

- **push** and **pop** treat the array as if it were a stack with the higher indices being nearer the top of the stack.
- Example

```
var list = ["Dasher", "Dancer", "Donner",
"Blitzen"]
var deer = list.pop(); // deer = Blitzen

// restores the lists as it was.
list.push("Blitzen");
```

unshift()

- **shift** and **unshift** remove and insert an item from or to the beginning of the list.
- Example

```
var list = ["Dasher", "Dancer", "Donner",
"Blitzen"]
var deer = list.shift(); // deer = Dasher

// restores the lists as it was.
list.unshift("Dasher");
```

nested_arrays.html

```
<!DOCTYPE html>

<!-- nested_arrays.html
     An example to illustrate an array of arrays
     displayed.
     -->

<html lang = "en">
  <head> <title> Font properties </title>
    <meta charset = "utp-8">
  </head>
```

```
  <body>
    <script type = "text/javascript">
    <!--
    // Create an array object with three arrays
    // as its elements

      var nestedArray = [[2, 4, 6],
                         [1, 3, 5],
                         [10, 20, 30]
                        ];
    // Display the elements of nestedList

    for (var row = 0; row <= 2; row++)  {
      document.write("Row ", row, ": ");

      for (var col = 0; col <= 2;  col++)
        document.write(nestedArray[row][col], " ");

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

   // -->
      </script>
  </body>
</html>
```

Functions

- Functions in JavaScript are similar to those in C/C++ and PHP.
- A function definition includes a header and a compound statement that specifies what the function does.
- A **`return`** statement return control to whatever called the function and passes back a value, if there is one specified in the **`return`** statement. If none is specified the function's result is **`undefined`**.

Function Fundamentals

- Functions in JavaScript are objects so variables that reference them can be handled like other object references:

```
function fun() {
  document.write("This is fun! <br />");
}
refFun = fun;
// Now, refFun refers to the fun object
fun(); // A call to fun
refFun();  // Also a call to fun
```

Local Variables

- Implicit variables (those not declared with a **`var`** statement) have global scope (over the entire XHTML document).
- Variables declared outside of a function definition also have global scope.
- Variables explicitly defined (using **`var`**) inside a function have only local scope.
- When there is a variable has been defined globally and locally, local scope has precedence.
- Functions in JavaScript can be nested but it is not considered a good idea.

Parameters

- JavaScript passes parameters by value. It passes object references by value, which still allows them to be changed. This provides one form of passing values by reference.
- There is no type checking of parameters and although the called function can use **`typeof`**, it cannot distinguish between different types of objects.

The **`arguments`** Array

- JavaScript does not check the number of parameters either. Excess actual parameters are ignored and excess formal parameters are set to **`undefined`**.
- All parameters are passed through the array arguments, which has a property length. This can be used to access all the parameters passed to the function.

parameters.html

```
<!DOCTYPE html>

<!-- parameters.html
     The params functrion and a test driver for it.
     This example illustates function parameters.
     -->
<html lang = "en">
  <head> <title> Parameters </title>
     <meta charset = "utp-8">
     <script type = "text/javascript">
     <!--
     // function params
     // Parameters: two named parameters and one
     //             unnamed parameter, all numbers
     // Returns:    Nothing
```

```
     function params(a, b)  {
       document.write("Function params was passed ",
          arguments.length, " parameter(s) <br />");
       document.write
             ("Parameter values are: <br />");

       for (var arg = 0;  arg < arguments.length;
                                             arg++)
         document.write(arguments[arg], "<br />");

       document.write("<br />");
     }
   // -->
    </script>
  </head>
  <body>
    <script type = "text/javascript">
    <!--

    // A text driver for params
      params("Mozart");
      params("Mozart", "Beethoven");
      params("Mozart", "Beethoven", "Tchaikowsky");


   // -->
      </script>
  </body>
</html>
```

Passing By Reference

- There is no elegant way to pass parameters in JavaScript. One way to do this is by passing the parameter as an object (e.g., an array).

```
function by10(a)  {
  a[0] *= 10;
}
…
var x
var listX = new Array(1);
listx[0] = x;
by10(list);
x = listx[0];
```

Passing By Reference (continued)

- Another way to pass the value back as the return value of the function:

```
function by10_2(a)  {
  return 10*a;
}
…
var x;
…
x = by10_2(x);
```

The **`sort`** Method, Revisited

- If you want to sort an array containing anything other than string, you must supply a comparison function that indicates which is greater:
  - A negative result indicates that the two values are in the correct order.
  - A zero result means that they are equal.
  - A positive result means that they must be switched.

Sorting, Revisited – An Example

```
// Function numOrder - 2 parameters a and b
// Returns a negative value if a and b are in
// order
// Returns 0 if a= b
// Returns a positive values if a and b need to
// be switched.
function numorder(a, b)  { return a - b; }

// Sort the array of numbers, list, into ascending
// order
numList.sort(numOrder);
```

An Example – **`medians.html`**

```
<!DOCTYPE html>

<!-- medians.html
     A function and a function tester
     Illustrates array operations
     -->

<html lang = "en">
  <head> <title> Median Computation </title>
     <meta charset = "utp-8">
     <script type = "text/javascript">
     <!--
     // function median
     // Parameter:       An array of numbers
     // Result:          The median of the array
     // Return value:    None
```

```
    function median(list)  {
      list.sort(function (a, b) { return a - b;});
       var listLen = list.length;

    //  Use the modulus operator to determine
    //  whether the array's length is odd or even
    //  Use the Math.floor to truncate numbers
    //  Use Math.round to round numbers

       if ((listLen % 2 ) == 1)
         return list[Math.floor(listLen / 2)];
       else
         return Math.round((list[listLen / 2 - 1]
                       + list [listLen / 2]) / 2);
    }

  // -->
   </script>
  </head>
```

```
  <body>
    <script type = "text/javascript">
    <!--

      var myList1 = [8, 3, 9, 1, 4, 7];
      var myList2 = [10, -2, 0, 5, 3, 1, 7];

      var med = median(myList1);
      document.write("Median of [", myList1,
                           "] is: ", med, "<br />");

      med = median(myList2);
      document.write("Median of [", myList2,
                           "] is: ", med, "<br />");
   // -->
      </script>
  </body>
</html>
```

Constructors

- Constructors are special methods that create and initialize properties of newly created objects. Calling a constructor is necessary for any new object.
- Constructor must be able to reference the object on which it is working. The reserved word this allows us to do that.
- If you wish to pass a reference to an object's method, the method must first be defined.

Constructors – An Example

```
<body>
  <script type = "text/javascript">
  function car (newMake, newModel, newYear)  {
    this.make = newMake;
    this.model = newModel;
    this.year = newYear;
    this.display = displayCar;
  }
  function displayCar()  {
    document.write("Car make: ", this.make,
                  "<br />");
    document.write("Car model: ", this.model,
                  "<br />");
    document.write("Car year: ", this.year,
                  "<br />");
  }

    myCar = new car("Ford", "Contour SVT", "2000");
    myCar.display();
  </script>
</body>
```

Pattern Matching Using Regular Expressions

- JavaScript has two approaches to pattern matching, one based on the **`RegExp`** object and another one based on the **`String`** object. We will use the latter.
- Patterns are based on the notation used for regular expressions.

Character and Character-Class Patterns

- Metacharacters have special meaning within patterns. They are
  `\ | ( ) [ ] { } ^ $ * + ? .`
- "Normal" characters are not metacharacters and match themselves within a pattern.
- The simplest technique for matching a pattern is **`search`**, which takes a pattern as a parameter.

Pattern Matching – An Example

```
<html>
<head> <title> Let's try this out </title>
</head>

<body>
  <script type = "text/javascript">
    var str = "Rabbits are furry";
    var position = str.search(/bits/);
    if (position > 0)
      document.write
           ("'bits' appears in position ",
                position, "<br />");
  </script>
</body>
</html>
```

Output **`'bits' appears in position 3`**

Pattern Matching – Some Other Examples

• `/snow./`	matches `snowy`, `snowe`, and `snowd` (among others).
• `/3\.4/`	matches `3.4`
• `[abc]`	matches `a`, `b` or `c`
• `[a-h]`	matches `a`, `b`, `c`, `d`, `e`, `f`, `g`, or `h`
• `[^aeiou]`	matches any character except `a`, `e`, `i`, `o` and `u`

Predefined Character Classes

Name	Equivalent Pattern	Matches
`\d`	`[0-9]`	A digit
`\D`	`[^0-9]`	Not a digit
`\w`	`[A-Za-z_0-9]`	A word character (alphanumeric)
`\W`	`[^A-Za-z_0-9]`	Not a word character
`\s`	`[ \r\t\n\f]`	A whitespace character
`\S`	`[^ \r\t\n\f]`	Not a whitespace character

Pattern Matching –Even More Examples

- `/\d.\d\d/` matches a digit, followed by a period, followed by 2 digits.
- `/\D\d\D/` matches a single digit (with non-digits on either side).
- `/\w\w\w/` matches three adjacent word characters.
- `/xy{4}z/` matches `xyyyyz`
- `/x*y+z?/` matches zero or more x's followed by one or more y's and possible by z.
- `/[A-Za-z]\w*/` matches identifier in most common programming languages.
- `/\bis\b/` matches `"A tulip is a flower"` but not `"A frog isn't"` (`\b` matches a border)

Anchors

- `^` anchors a pattern to the beginning of a string.
- `$` anchors a pattern to the end of a string.
- Examples
  - `/^pearl/` - matches `"pearls are pretty"` but not `"My pearls are pretty"`
  - `/gold$/` - matches `"I like gold"` but not `"golden"`

Pattern Modifiers

- Modifiers can appear at the end of a pattern to increase their flexibility
- i allows the pattern to match either lower or upper case
  - `/Apple/i` matches `"APPLE" "Apple" or "apple"`
- x allows white space to be added to the pattern.

```
/\d+          # The street number
\s            # The space before the street name
[A-Za-z]+     # The street name
/x
```

  is equivalent to `/\d+\s[A-Za-z]+/`

Other Pattern Matching Methods of String

- **`replace`** replaces substrings of the String object that match the given pattern.
  - `replace` takes 2 parameters: the pattern that it seeks to replace and the string that replaces it.
  - The matched substrings are assigned to the predefined variables `$1`, `$2`, `$3`, etc.
- **`match`** takes one parameter (the pattern to be matched) and returns and array of the strings that match it.
- **`split`** splits the object string into substring based on the pattern given as its parameter.

`replace` – An Example

```
var str =
    "Fred, Freddie, and Frederica were siblings";
str.replace("/Fre/g, "Boy");
```

changes the string to

```
"Boyd, Boyddie, and Boyderica were siblings"
```

`match` – An Example

```
var str =
  "Having 4 apples is better than having 3 oranges";
var matches = str.match(/\d/g);
```

matches will be set to `[4, 3]`

```
var str = "I have 428 dollars, but I need 500";
var matches = str.match(/(\d+)([^\d]+)(\d+)/
document.write(matches, "<br />");
```

matches will be set to

```
["428 dollars, but I need 500", "428",
  "dollars, but I need ", "500"]
```

`split` – An Example

```
var str = "grapes: apples: oranges";
var fruit = str.split(":")
```

`fruit` is set to **`["grapes", "apples", "oranges"]`**

`formschecks.html`

```
<!DOCTYPE html>

<!-- formscheck.html
     A function tstPhoneNum is defined and tested.
     This function checks the validity of phone
     number input from a form
     -->

<html lang = "en">
  <head> <title> Median Computation </title>
     <meta charset = "utp-8">
     <script type = "text/javascript">
     <!--
```

```
    /* Function testPhoneNum
       Parameters; A string
       Result:  Returns true if the parameters has
                the form of a legal seven-digit
                phone number 3 digits, a dash, 4
                digits)
     */
      function tstPhoneNum(num)  {
        // Use a simple pattern to check the
        // number of digits and the dash
        var ok = num.search(/\d{3}-\d{4}/);

        if (ok == 0)
          return true;
        else
          return false;
      }
  // -->
    </script>
  </head>
```

```
  <body>
    <script type = "text/javascript">
    <!--
      // A script to test txtPhoneNum
      var tst = tstPhoneNum("444-5432");
      if (tst)
        document.write("444-5432 is a ",
                  "legal phone number <br />");
      else
        document.write("Error in tstPhoneNum",
                       " <br />");

      tst = tstPhoneNum("444-r432");
      if (tst)
        document.write("Error in tstPhoneNum",
                                       " <br />");
      else
        document.write("444-r432 is not",
                 " a legal phone number <br />");
      tst = tstPhoneNum("44-5432");
      if (tst)
        document.write("Error in tstPhoneNum",
                                      " <br />");
      else
        document.write("44-5432 is not ",
                    "a legal phone number <br />");
   // -->
      </script>
  </body>
</html>
```

Errors in Scripts

- The Default setting for Internet Explorer does not provide debugging for JavaScript.
  - This can be changed by going to the Tools menu (select) Internet options. Uncheck the box that reads "Disable JavaScript debugging."
- JavaScript can be debugged in Firefox by going to the Tools menu and selecting "Error Console."