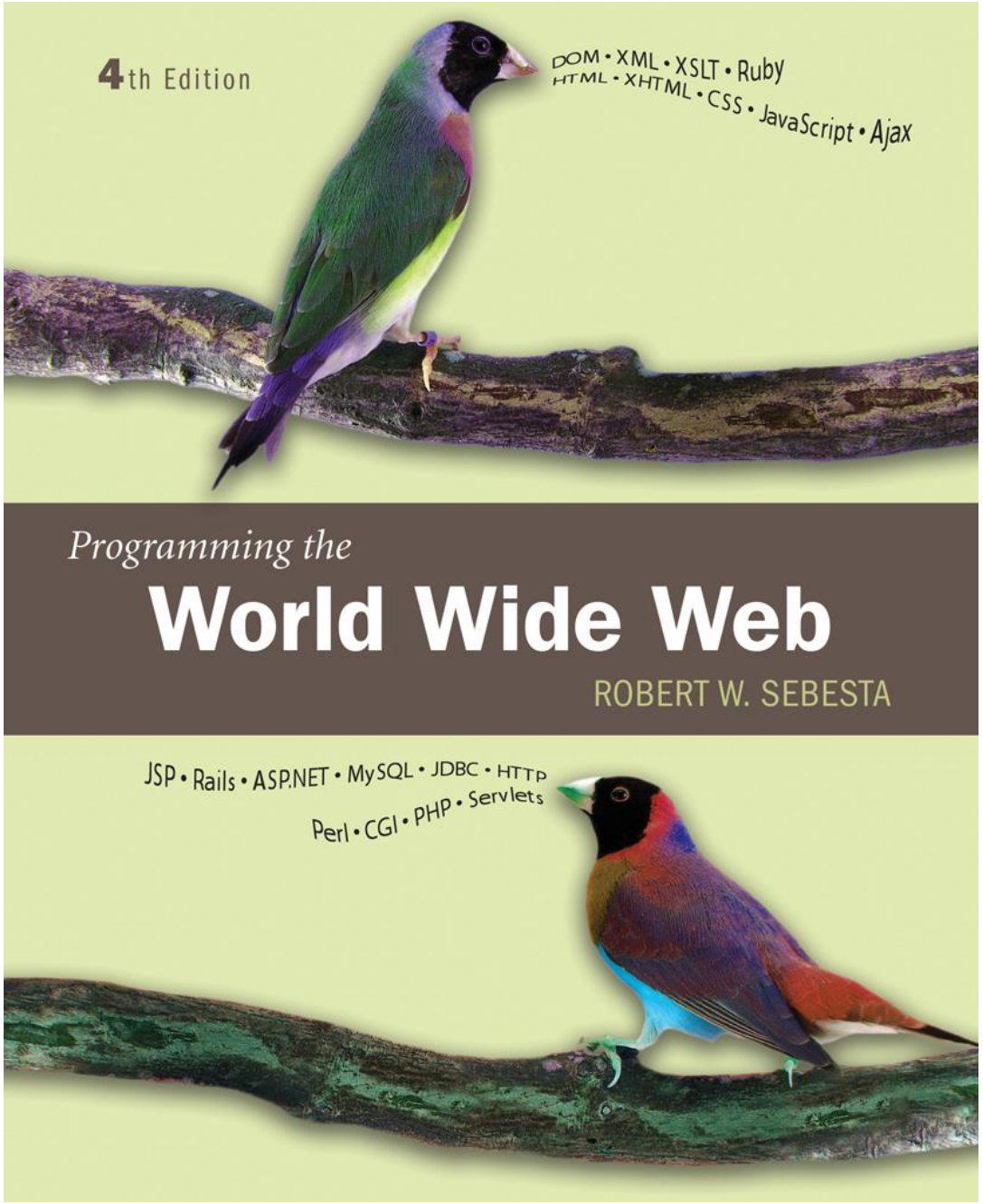


Chapter 4
Basics of JavaScript

Overview

- Overview of Java Script
- General syntactic characteristics
- Primitive, Operations & Expression
- Control Statements
- Screen output & Keyboard Input
- Functions
- Object Creation and Modifications
- Arrays
- Pattern Matching (RegExp)
- Constructors

Overview of JavaScript: Origins

- LiveScript, Originally developed by Netscape.
- Became a joint venture of Netscape and Sun in 1995, renamed JavaScript.

- Now standardized by the European Computer Manufacturers Association as ECMA-262
- An XHTML-embedded scripting language.

- We'll call collections of JavaScript code *scripts*, not programs.

- ECMA-262 edition 3 is the current standard.
  - Edition 4 is under development.
- Supported by Netscape, Mozilla, Internet Explorer.

Overview of Java Script

JavaScript is the most popular scripting language on the internet, and works in all major browsers, such as Internet Explorer, Firefox, Chrome, Opera, and Safari.

Scripts are Simple & Interpreted programming Language.

- Scripts are embedded as plain text, interpreted by application.
- Simpler execution model: Don't need compiler or IDE,
- Saves bandwidth: Source code is downloaded, not compiled executable.
- Platform-independence: code interpreted by any Script-enabled browsers.

JavaScript is the first web scripting language, developed by Netscape (Brendan Eich) in 1995. Syntactic similarities to Java/C++, but simpler and more flexible. Some silent features include:

a. Loose typing.
b. Dynamic variables.
c. Simple Objects.
d. Event driven.

4

What Java Script can do

Adding dynamic features to Web pages
- validation of form data
- image rollovers
- time-sensitive or random page elements
- handling cookies

Defining programs with Web interfaces
- utilize buttons, text boxes, clickable images, prompts, frames

limitations of client-side scripting
- since script code is embedded in the page, viewable to the world
- for security reasons, scripts are limited in what they can do

e.g., can't access the client's hard drive
- since designed to run on any machine platform, scripts do not contain platform specific commands
- script languages are not full-featured

e.g., JavaScript objects are crude, not good for large project development

- JavaScript can be divided into three parts:

  The core, client side and server side.

- The core is the heart of the language, including its operators, expressions, statements and subprograms.

- Client-side Javascript is a collection of objects that support control of a browser and interactions with users.

  Example: With JavaScript, an XHTML document can be made responsive to user inputs such as mouse clicks and keyboard use.

- Server-side JavaScript is a collection of objects that make the language useful on a web server.

  Example: To support communication with a database management system.

- Client-side JavaScript is an XHTML embedded scripting language, referred to as script.

JavaScript and Java

- JavaScript and Java are only related through syntax of their expressions, assignment statements and control statements.

- Java is strongly typed language, JavaScript is dynamically typed.

- JavaScript has a different object model from Java. JavaScript's support for objects is very different.
- Objects in Java are static, collection of data members and methods are fixed at compile time.
- JavaScript objects are dynamic-the number of data members and methods of an object can change during execution.

- JavaScript is interpreted.
  - Source code is embedded inside XHTML doc, there is no compilation.

Uses of JavaScript

- Can be used to replace some of what is done with CGI (but no file operations or networking).
- Can be used to replace some of what is typically done with applets (except graphics).
- User interactions through forms are easy
  - Events easily detected with JavaScript
  - E.g. validate user input
- The Document Object Model makes it possible to support dynamic HTML documents with JavaScript.

Uses of JavaScript

- Provide alternative to server-side programming.
  - Servers are often overloaded.
  - Client processing has quicker reaction time.

- JavaScript can work with forms.

- JavaScript can interact with the internal model of the web page (Document Object Model).

- JavaScript is used to provide more complex user interface than plain forms with HTML/CSS can provide
  - http://www.protopage.com/ is an interesting example
  - A number of toolkits are available. Dojo, found at http://dojotoolkit.org/, is one example

Event-Driven Computation

- Users actions, such as mouse clicks and key presses, are referred to as *events*.

- The main task of most JavaScript programs is to respond to events.
- For example, a JavaScript program could validate data in a form before it is submitted to a server
  - *Caution:* It is important that crucial validation be done by the server. It is relatively easy to bypass client-side controls
  - For example, a user might create a copy of a web page but remove all the validation code.

- JavaScript scripts are executed entirely by the browser.
- Once downloaded there is no exchange of information with the server.
  - NB JavaScript programs can issue HTTP requests and load other pages.

- JavaScript scripts do not require the Java VM to be loaded.
- Thus JavaScript scripts tend to be fast.

XHTML/JavaScript Documents

- When JavaScript is embedded in an XHTML document, the browser must interpret it.

- Two locations for JavaScript server different purposes
  - JavaScript in the head element will react to user input and be called from other locations.
  - JavaScript in the body element will be executed once as the page is loaded.

- Various strategies must be used to ‘protect’ the JavaScript from the browser
  - For example, comparisons present a problem since < and > are used to mark tags in XHTML.
  - JavaScript code can be enclosed in XHTML comments.
  - JavaScript code can be enclosed in a CDATA section.

Object Orientation

- JavaScript is NOT an object-oriented programming language
  - Rather object-based .
- Does not support class-based inheritance
  - Cannot support polymorphism.
- JavaScript objects are collections of properties, which are like the members of classes in Java
  - Data and method properties .
- JavaScript has primitives for simple types
- The root object in JavaScript is Object – all objects are derived from Object

Object Orientation and JavaScript

- Objects are collections of *properties.*
- Properties are either *data properties* or *method properties.*

- Data properties are either primitive values or references to other objects.
- Primitive values are often implemented directly in hardware.
- The Object object is the ancestor of all objects in a JavaScript program
  - Object has no data properties, but several method properties

- JavaScript is *object-based*
  - JavaScript defines objects that encapsulate both data and processing
  - However, JavaScript does not have true inheritance nor subtyping
- JavaScript provides *prototype-based inheritance*

Embedding in XHTML docs

- **Directly embedded**

```
<script type="text/javascript">
    <!--
        …Javascript here…
    -->
</script>
```

  - **However, note that** a-- **will not be allowed here!**

- **Indirect reference**

  Or indirectly, as a file specified in the src attribute of <script>, as in

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

- **This is the preferred approach**

Overview

- Overview of Java Script
- General syntactic characteristics
- Primitive, Operations & Expression
- Control Statements
- Screen output & Keyboard Input
- Functions
- Object Creation and Modifications
- Arrays
- Pattern Matching (RegExp)
- Constructors

JavaScript Objects

- An Object in JavaScript is a collections of *properties* and *methods. (member of classes in Java and C++).*
- Properties are either *data properties* or *method properties.*

- Data properties are either primitive values or references to other objects.
- Primitive values are often implemented directly in hardware.
- Method properties are referred to as methods or functions.

- All objects in JavaScript program are indirectly accessed through variables.
- Properties of an object are referenced by attaching the name of the property to the variable that references the object.

- The Object (root) object is the ancestor of all objects in a JavaScript program.
  - Object has no data properties, but several method properties.
  - Properties are names, values are data values or functions.

General Syntactic characteristics

- Syntax based on C, C++ and Java
  Ex: var val = “This is string assignment”;

- Variables are Case Sensitive
  Ex: a. var x not same as var X

- Semicolon (;) at the end.
  - Semicolon are automatically inserted to code.
  - Better to write semicolon (;) always.

- Block of Code defined by { …. }
  Ex: function add() { a =10; b=20; c = a + b; return c; }

- Single line comments is prefixed by //
- Multiline comment start with ‘/*’ and Ends with ‘*/’

- Variable declaration is not necessary
  Ex: var x = 10 same as x=10
  - Variables can be declared by using keyword ‘var’ .
  - Variables can contain letters, numbers and _ (underscore) .
  - Variable name should begin with Letter, _ (underscore), dollar sign($).
  - Initial value of any variable is ‘undefined’ .

Java script code can be embedded in a XHTML document using <script> tag.

a. Java Scripts in the body section will be executed while the page loads.

b. Java Scripts in the head section will be executed when CALLED.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- xhtml1-transitional.dtd-->
<!-- xhtml1-frameset.dtd -->

<html>
  <head>
            <title> Simple JS </title>
    <script language="javascript">
            <!--document.write("When called ");-->
    </script>
  </head>
  <body>
            <script language="javascript">
        document.write(“ <h2> Document Load ... </h2> ");
            </script>
  </body>
</html>
```

Document.write Displays text in page

Text to be displayed can include html tags

Tags are interpreted by browser when text is displayed

Java script as file inclusion

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html>
  <head>
            <title> External JS file inclusion </title>

            <script language="javascript" src="one.js">
</script>
  </head>

  <body>
            Hi
  </body>
</html>
```

```
// External JS File
document.write("External Java script inclusion !!!!
<br/>");
```

To use the external script, point to the .js file in the "src" attribute of the <script> tag .

Save the external JavaScript file with a .js file extension .

.js file should not contain <script> tag again.

Statement Syntax

- Statements can be terminated with a semicolon.
- However, the interpreter will insert the semicolon if missing at the end of a line and the statement seems to be complete.
- Can be a problem:

```
return
x;
```

- If a statement must be continued to a new line, make sure that the first line does not make a complete statement by itself.

Declaring Variables

- JavaScript is dynamically typed – any variable can be used for anything (can have a primitive value or it can be a reference to any object).
- The interpreter determines (and converts) the type of a particular occurrence of a variable.
- Variables can be either implicitly declared by assigning it a value or explicitly declared by listing it in a declaration statement that begins with reserved word var.

```
var sum = 0,
    today = "Monday",
    flag = false;
```

Primitive Types

- Five primitive types
  - Number
  - String
  - Boolean
  - Undefined
  - Null
- Number, String, and Boolean have wrapper objects.
- Each contains a property that stores a value of the corresponding primitive type.

- Purpose of wrapper objects is to provide properties and methods convenient for use with values of primitive types.

- In the cases of Number and String, primitive values and objects are coerced back and forth so that primitive values can be treated essentially as if they were objects.

- All numeric values are stored in double-precision floating point.
- String literals are delimited by either ' or "
- Boolean values are true and false.
- The only Null value is null.
- The only Undefined value is undefined.

Primitives and Object Storage:

Suppose that prim is a primitive variable with the value 17 and obj is a Number object whose property value is 17.

Fig shows how prim and obj are stored.

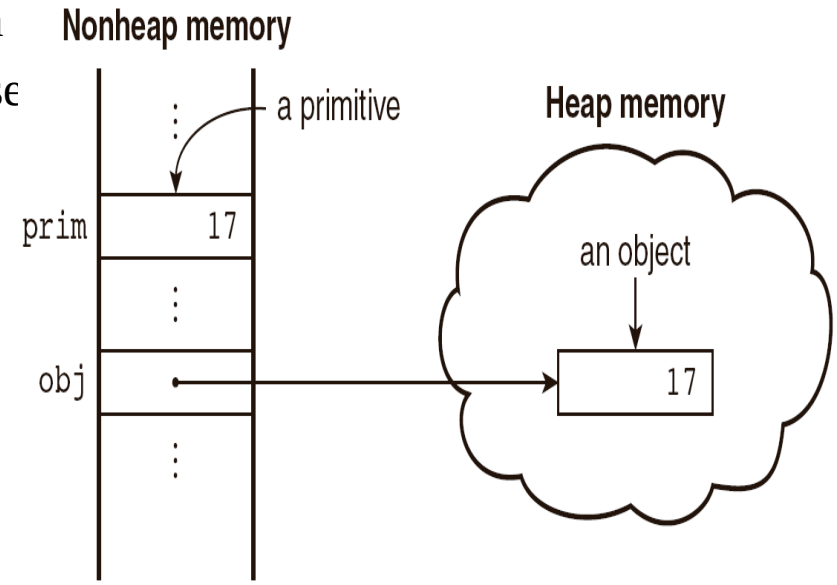


Figure 4.1 Primitives and objects

Numeric and String Literals

- All numeric literals are values of type Number.
- Number values are represented internally as double-precision floating-point values
  - Number literals can be either integer or float-point values.
  - Integer literals are strings of digits.
  - Float values may have a decimal points and/or and exponents or both.

  Example of legal numeric literals are:

  72 7.2 .72 72. 7E2 7e2 .7e2 7.e2 7.2E-2

- A String literal is sequence of zero or more characters delimited by either single or double quotes.
  - There is no difference between single and double quotes.
  - Certain characters may be *escaped* in strings.
    - \’ or \” to use a quote in a string delimited by the same quotes.
    - \\ to use a literal backspace.
  - The empty string ‘’ or “” has no characters.

Other Primitive Types

- Null
  - A single value of type Null, reserved word null.
  - A variable that is used but has not been declared nor been assigned a value has a null value.
  - Using a null value usually causes a runtime error.

- Undefined
  - A single value, undefined
  - However, undefined is not, itself, a reserved word
  - The value of a variable that is declared but not assigned a value, if displayed word “undefined” is displayed.

- Boolean
  - Two values: true and false.
  - Computed as a result of evaluating a relational or boolean expression.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html>
<head>
<title>Data Types and Variables</title>
</head>
<body>
<script type="text/javascript">
x = 1024;
document.write("<p>x = " + x + "</p>");
x = "foobar";
document.write("<p>x = " + x + "</p>");
</script>
</body>
</html>
```

you don't have to declare variables, will be created the first time used

Variables are loosely typed, can assign different types of values

Assignments are as in C++/Java

Message = “Hello”;
PI = 3.14159;

isBoolean = true;

Operators:

- Assignment operator '=' used to assign values to JavaScript variables
- Arithmetic operator '+' used to add values together
- '+' Also used with String for Concatenation
- Number with String on '+' will give String as a result

Arithmetic operator y=5

+	Addition	x=y+2	x=7
-	Subtraction	x=y-2	x=3
*	Multiplication	x=y*2	x=10
/	Division	x=y/2	x=2.5
%	Modular	x=y%2	x=2.5
++	Increment	x=++y	x=6
--	Decrement	x=--y	x=4

Assignment operator y=5 and x=10

=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that x=5, the table below explains the comparison operators:

Operator	Description	Example
==	is equal to	x==8 is false
===	is exactly equal to (value and type)	x===5 is true x==="5" is false
!=	is not equal	x!=8 is true
>	is greater than	x>8 is false
<	is less than	x<8 is true
>=	is greater than or equal to	x>=8 is false
<=	is less than or equal to	x<=8 is true

Logical Operators

Logical operators are used to determine the logic between variables or values.

Given that x=6 and y=3, the table below explains the logical operators:

```
&&  and  (x < 10 && y > 1) is true
||  or   (x==5 || y==5) is false
!   not  !(x==y) is true
```

Numeric Operators

Standard arithmetic

+ * - / %

Unary operators: + - Increment and decrement -- ++

Increment and decrement differ in effect when used before and after a variable.
Assume that a has the value 7, initially
(++a) * 3 has the value 24
(a++) * 3 has the value 21
a has the final value 8 in either case

Precedence rules of a language- specify which operator is evaluated first when 2 operators with different precedence are adjacent in an expression.
Associativity rules of a language- which operator is evaluated first when two operators with same precedence are adjacent in an expression.

Operator	Associativity
++, --, unary –	Right (though it is irrelevant)
*, /, %	Left
Binary +, binary –	Left

The first operators listed have the highest precedence.

Precedence of Operators

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

Example of Precedence

```
var a = 2,
b = 4,
c,
d;
c = 3 + a * b;
// * is first, so c is now 11 (not 24)
d = b / a / 2;
// / associates left, so d is now 1 (not 4)
```

The Math Object

- The Math object provides a collection of properties of Number objects and methods that operate on Number objects.
- The Math Object has methods for the trigonometric functions such as sin, cos.
- Also it provides floor (to truncate a number), round (to round a number), max (return largest of two numbers), min, etc.
- All of the Math methods are referenced through Math object, as
  - e.g., Math.cos(x)

The Number Object

- Properties of Number

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

- Operations resulting in errors return NaN, values that cannot be represented as a double-precision floating-point number, one that is too large(overflow), returns NaN.
  - Use isNaN(a) to test if a is NaN.
  - To determine if a variable has the NaN value, predefined predicate function isNaN() must be used.

- The Number object has a method toString which converts the number through which it is called to a string.

  Example:

```
var price = 427,
        str_price;
. . .
str_price = price.toString();
```

String Catenation operator

- JavaScript strings are stored and treated as unit scalar values.
- The operation + is the string catenation operation.

  Example:

  If the value of first is “Freddie”, the value of the following expression is “Freddie Freeloader” :

```
first + “ Freeloader”
```

Implicit Type Conversion

- JavaScript interpreter performs several different implicit type conversions, called coercions.

- When a value of one type is used in a position that requires a value of a different type, JavaScript attempts to convert values in order to be able to perform operations.

- If either operand of a + operator is a string, the operator is interpreted as a string catenation operator.

  "August" + 1977 causes the number to be converted to string and a concatenation to be performed.

- Expression hence evaluates to "August 1977".

- If operator is the one used only with numbers, forces numeric context on the right operand as follows:

  7 * “3” causes the string to be converted to a number and a multiplication to be performed.

- If the second operand were a string that could not be converted to a number, such as “August” , the conversion would produce a NaN.

- null is converted to 0 in a numeric context, undefined to NaN.
- 0 is interpreted as a Boolean false, all other numbers are interpreted a true.

- The empty string is interpreted as a Boolean false, all other strings (including “0”!) as Boolean true.
- undefined, Nan and null are all interpreted as Boolean false.

Explicit Type Conversion

- Used to force type conversions between strings and numbers.

- Strings that contain numbers can be converted to numbers with the **String** constructor as
  var str_value = String(value);

- Conversion is also possible with the **toString** method:
  var num = 6;
  var str_value = num.toString();
  var str_value_binary = num.toString(2);
  Result is “6” for first conversion and “110” for second.

- A number can be converted to string by **catenating it with the empty string**

- Strings can explicitly be converted to numbers using **Number constructor** as follows:
  var number = Number(aString);

- Explicit conversion of string to number.

  var number = aString – 0

  - Number must begin the string and be followed by space or end of string (causes problem if any other character is present).

- 2 predefined string functions which do not have this problem:

  parseInt and **parseFloat** convert the beginning of a string but do not cause an error if a non-space follows the numeric part.

- The **parseInt** function searches the string for an Integer literal.
- If one is found at the beginning of the string, it is converted to number and returned.
- Otherwise NaN is returned.

- The parseFloat function searches for a floating-point literal.

String Properties and Methods

- The String object includes one property length, and a large collection of methods.
- Character positions in strings begin at index 0.

- The number of characters in a string is stored in the length property as follows:

  var str = “George”;

  var len = str.length;
- len is set to the number of characters in str, 6.

String methods:

Method	Parameters	Result
`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 numbers	The substring of the `String` object from the first parameter position to the second
`toLowerCase`	None	Converts any uppercase letters in the string to lowercase
`toUpperCase`	None	Converts any lowercase letters in the string to uppercase

- For the String methods, character position starts at zero.

- For example, suppose str has been defined as follows:

  var str = “George”;

The following expressions have the values as follows:

str.charAt(2) is ‘o’
str.indexOf(‘r’) is 3
str.substring(2, 4) is ‘org’
str.toLowerCase() is ‘george’

The typeof Operator

- The typeof operator returns the type of its single operand.

- Returns “number” or “string” or “boolean” for primitive types.

- If the operand is an object or null, typeof evaluates to “object”.
- Returns undefined, if the operand is a variable that has not been assigned a value.

- Two syntactic forms
  - typeof x
  - typeof(x)

  - typeof operator always returns a string.

Assignment Statements

- Simple assignment operator is denoted by =

- Compound assignment with
  - += -= /= *= %= …

  Example:

  a += 7 means the same as a = a + 7.

- A variable in JavaScript can refer to a primitive value such as 17, or an object.
- Objects are allocated on heap, and variables that refer to them are essentially reference variables.
- When used to refer to an object, a variable stores an address only.
- Therefore assigning the address of an object to a variable is fundamentally different from assigning a primitive value to a variable.

The Date Object

- A Date object represents a *time stamp*, that is, a point in time.

- A Date object is created with the new operator and the Date constructor.
- The simplest Date constructor, takes no parameter and builds an object with current date and time for its properties.

```
var today= new Date();
```

The date and time properties of a Date object are in two forms: local and UTC (Coordinated Universal time or Greenwich Mean Time).

The Date Object: Methods

toLocaleString	A string of the Date information
getDate	The day of the month
getMonth	The month of the year, as a number in the range of 0 to 11
getDay	The day of the week, as a number in the range of 0 to 6
getFullYear	The year
getTime	The number of milliseconds since January 1, 1970
getHours	The number of the hour, as a number in the range of 0 to 23
getMinutes	The number of the minute, as a number in the range of 0 to 59
getSeconds	The number of the second, as a number in the range of 0 to 59
getMilliseconds	The number of the millisecond, as a number in the range of 0 to 999

Screen output & Keyboard Input

- The JavaScript model for the XHTML document is the Document Object model.
- The model for the browser display window is the Window object.
- The Window object has two properties, document and window, which refer to the Document and Window objects, respectively.
- The document property refers to the Document object.
- The window property refers to the Window object.
- The Document object has a method, **write,** used to create script output, which dynamically creates XHTML document content.
- The parameter is a string, often catenated from parts, some of which are variables.

  e.g., document.write("Answer: " + result + "
");

- The parameter is sent to the browser, so it can be anything that can appear in an XHTML document (
, but not \n).
- If a line break is needed in the output, interpolate
 into the output.

- The parameter to write can include any XHTML tags and content.
- Write can take any number of parameters, multiple parameters are concatenated and placed in the output.

Window creates three methods that creates dialog boxes for three specific kinds of user interactions.

Alert Box:

An alert box is often used if you want to make sure information comes through to the user. When an alert box pops up, the user will have to click "OK" to proceed.

Confirm Box:

A confirm box is often used if you want the user to verify or accept something. When confirm box pops up, the user will have to click either "OK" or "CANCEL" to proceed. If the user clicks "OK", the box returns true. If user clicks "CANCEL", the box returns false.

Prompt Box:

A prompt box is often used if you want the user to input a value before entering the page. When a prompt box pops up, the user will have to click either "OK" or "CANCEL" to proceed after entering an input value. If the user clicks "OK" the box returns the input value. If the user clicks "CANCEL" the box returns null.

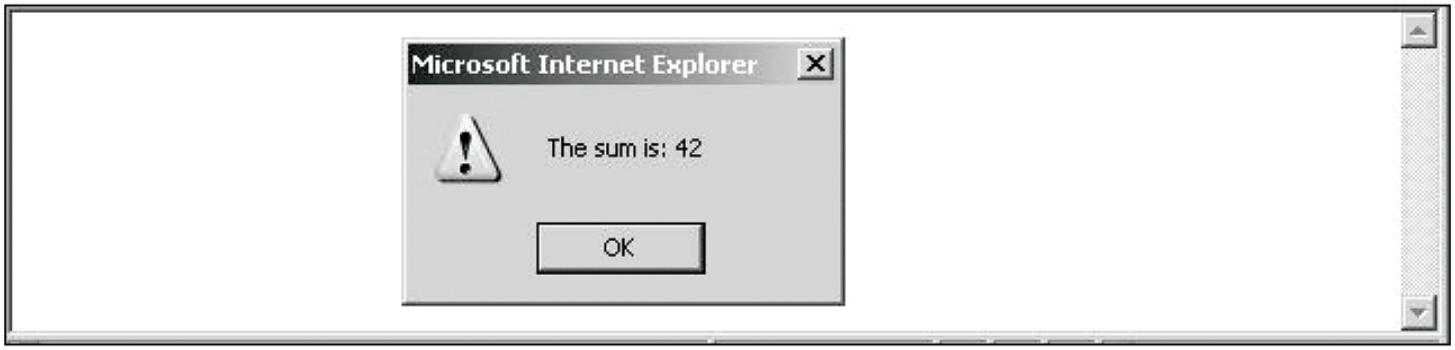
Screen Output

The Window object has three methods for creating dialog boxes.

1. Alert

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

- Parameter is plain text, not XHTML.
- Opens a dialog box which displays the parameter string and an OK button.
  - It waits for the user to press the OK button .



Screen Output

2. Confirm

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

- Opens a dialog box and displays the parameter and two buttons, OK and Cancel
- Returns a Boolean value, depending on which button was pressed (it waits for one).
- true for OK and false for cancel.

Screen Output

3. Prompt

```
prompt("What is your name?", “  ”);
```

- Opens a dialog box and displays its string parameter, along with a text box and two buttons, OK and Cancel.
- The text box is used to collect a string of input from the user, which prompt returns as its value.
- The second parameter is for a default response if the user presses OK without typing a response in the text box (waits for OK).

Example of Input and Output

```
<xml version="1.0">
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1//EN"
                http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd>
<!– roots.html
      A document for roots.js  -->
<html xmlns = http://www.w3.org/1999/xhtml>
<head>
      <title>roots.html</title>
</head>
<body>
      <script type = "text/javascript" src = "roots.js">
      </script>
</body>
</html>
```

```
// roots.js
// 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

// 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 root_part = Math.sqrt(b * b – 4.0 * a * c);
   var denom = 2.0 * a;

// Compute and display the two roots
   var root1 = (-b + root_part ) / denom;
   var root2 = (-b – root_part ) / denom;
document.write("The first root is:  ",root1, "<br />");
document.write("The second root is:  ",root2, "<br />");
```

Control Statements

- Control expressions provide a basis for controlling the order of execution of statements.

- A *compound* is a sequence of statements enclosed in curly braces. Compound statemes*tatement* in JavaScript nts can be used as components of control statements allowing multiple statements to be used.

- A *control construct* is a control statement including the statements or compound statements whose execution it controls.

- A variable declared in a compound statement, access is not confined to that compound statement but is visible for the whole XHTML document.

  But an exception to this rule is if the variable is declared in a function.

Control Expressions

- A result of evaluating a control expression is one of the Boolean value true or false.

- If the value of a control expression is a string, it is interpreted as true unless it is either the empty string(“ ”) or a zero string(“0”).

- If the value is a number, it is true unless it is zero (0).
  - An expression with a non-Boolean value used in a control statement will have its value converted to Boolean automatically.

Relational Operators

Relational operators are used in logical statements to determine equality or difference between variables or values.
Given that x=5, the table below explains the comparison operators:

Operator	**Description**	**Example**
= =	is equal to	x= =8 is false
= = =	is exactly or strictly equal to (value and type)	x= = =5 is true x= = ="5" is false
!=	is not equal	x!=8 is true
!= =	Is strictly not equal to	5!= =“8” is false
>	is greater than	x>8 is false
<	is less than	x<8 is true
>=	is greater than or equal to	x>=8 is false
<=	is less than or equal to	x<=8 is true

Comparison operators

== != < <= > >=

=== compares identity of values or objects

3 == ‘3’ is true due to automatic conversion

3 === ‘3’ is false (no type conversion allowed)

54

Logical Operators

- Logical operators are used to determine the logic between variables or values.
- Given that x=6 and y=3, the table below explains the logical operators:

&& and (x < 10 && y > 1) is true

|| or (x==5 || y==5) is false

! not !(x==y) is true

Conditionals

Selection statements – “if” and “if…else“

```
if (a > b)
  document.write(“a is greater than b <br />”);
else {
  a = b;
  document.write(“a was not greater than b, now they are equal <br />”);
}
```

The switch statement semantics

The expression is evaluated.

The value of the expressions is compared to the value in each case in turn.

If no case matches, execution begins at the default case.

Otherwise, execution continues with the statement following the case.

Execution continues until either the end of the switch is encountered or a break statement is executed.

Conditional statements are used to perform different actions based on different conditions.

if Statement: We use if statement to execute some code only if a specified condition is met
Syntax:

```
if(condition){
  code to be executed when condition is true;
}
```

if…else Statement: We use if-else statement to execute some code when condition is true and another code when condition is not true
Syntax:

```
if(condition){
  code to be executed when condition is true;
}else{
  code to be executed when condition is false;
}
```

If…else if…else Statement: We use this type of statements to select one of several blocks of code to be executed
Syntax:

```
If(condition1){
  code to be executed when condition 1is true
}else if(condition2){
  code to be executed when condition2 is true
}else{
  code to be executed when both condition1 & condition2 are false
}
```

switch Statement: We use switch statement to execute one or many blocks of code to be executed.
Syntax:

```
switch(n){
  case 1:
            execute code block 1;
            break;
  case 2:
            execute code block 2;
            break;
  default:
            code to be executed if n is different from case 1 and 2
}
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
</head>
<body>
<script type="text/javascript">
  var d = new Date();
  day = d.getDay();
  switch(day){
    case “5”: document.write(“Friday”); break;
    case “6”: document.write(“Saturday”); break;
    default: document.write(“Invalid day”);
  }
</script>
</body>
</html>
```

new Date() will get system date
getDay() – day of week
getHours() – Hour of day
getYear() – Year
getMonth() – Month of year
getMinutes() – of hours
getSeconds() – of Minutes
getMilliseconds() – of Seconds

Loop Statements

- Loop statements in JavaScript are similar to those in C/C++/Java.

- **While**

  while (*control expression*)

  statement or compound statement

- **For**

  for (*initial expression; control expression; increment expression*)

  statement or compound statement

- **do/while**

  do *statement or compound statement*

  while (*control expression*)

while Statement Semantics

- The control expression is evaluated.
- If the control expression is true, then the statement is executed.
- These two steps are repeated until the control expression becomes false.
- At that point the while statement is finished.

do/while Statement Semantics

- The statement is executed.
- The control expression is evaluated.
- If the control expression is true, the previous steps are repeated.
- This continues until the control expression becomes false.
- At that point, the statement execution is finished.

for Statement Semantics

- The initial expression is evaluated.
- The control expression is evaluated.
- If the control expression is true, the statement is executed.
- Then the increment expression is evaluated.
- The previous three steps are repeated as long as the control expression remains true.
- When the control expression becomes false, the statement is finished executing.

for loop: loops through a block of code
A specified number of times

Syntax:

```
for(start;loop end;increment){
  code to be executed;
}
```

Example:

```
<html>
  <head>

  </head>
  <body>
    <script
language="javascript">
        for(i=0;i<5;i++){

document.write("<br/>NUM"+i);
        }
    </script>
  </body>
</html>
```

while: loops through a block of code
While a specified condition is true

Syntax:

```
while(var <= endvalue){
 code to be executed;
}
```

Example:

```
<html>
  <head>

  </head>
  <body>
    <script
language="javascript">
        var i=0;
        while(i<5){

document.write("<br/>NUM"+i)
;
            i++;
        }
    </script>
  </body>
</html>
```

do-while: Execute a block of code ONCE,
and then it will repeat the loop as long as
The specified condition is true.

Syntax:

```
do{
 code to be executed;
} while(var <= endvalue);
```

Example:

```
<html>
  <head>

  </head>
  <body>
    <script
language="javascript">
        var i=0;
        do{

document.write("<br/>NUM"+i)
;
            i++;
        }while(i<5);
    </script>
  </body>
</html>
```

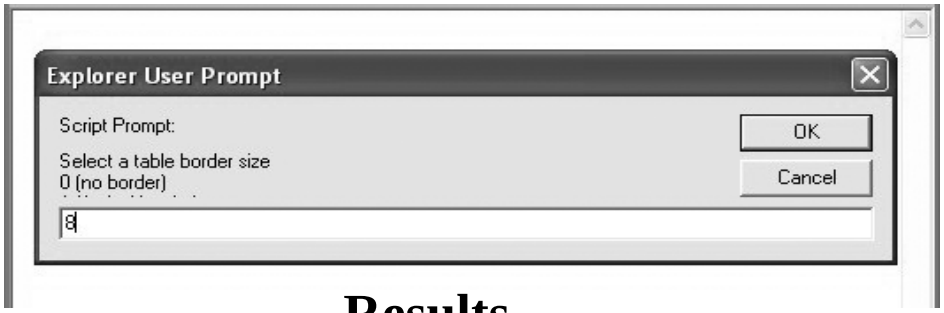
```
// border2.js
//An example of a switch statement for table border size selection

var 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 />");
               break;
    document.write("<caption> 2006 NFL Divisional", "Winners </caption>");
```

```
document.write("<table>"
                "<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> Chicago Bears </td>",
               "</tr>",
               "<tr>",
               "<th > West </th>",
               "<td> San Deigo Chargers</td>",
               "<td> Seattle SeaHawks </td>",
               "</tr>",
```

```
        "<tr>",
          "<th > South </th>",
          "<td> IndianPolis Colts </td>",
          "<td> New Orleans Saints </td>",
        "</tr>",
"</table>");
```

User Input Prompt



Results

2006 NFL Divisional Winners

	American Conference	National Conference
East	New England Patriots	Philadelphia Eagles
North	Baltimore Ravens	Chicago Bears
West	San Diego Chargers	Seattle Seahawks
South	Indianapolis Colts	New Orleans Saints

Object Creation and Modification

- **Java script is not-object oriented, but it is Object based scripting language.**
- **Objects can be created using ‘new’ keyword followed by a call to a constructor method.**
- **The constructor called in the *new* expression creates the properties that characterize the new object.**
- **In Java, the new operator creates a particular object, with a type and a specific collection of members.**
- **In JavaScript, new operator creates a blank object, or one with no properties.**
- **JavaScript objects do not have types.**
- **The constructor both creates and initializes the properties.**

The following statement creates an object that initially has no properties.

Syntax:

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

- **New object will not have any properties, and properties can be added to object any time during the execution.**
- **The variable my_object references the new object.**
- **Call to constructor must include parenthesis, even if there are no parameters.**

Constructors

- **JavaScript constructors are special methods that create and initialize the properties for newly created objects.**

- **Every new expression must include a call to a constructor whose name is same as the object being created.**

  Example: Constructor for arrays is Array().

- **Constructor must be able to reference the object on which it is to operate.**

- **JavaScript has a predefined reference variable for this purpose, named this.**

- **When the constructor is called, this is a reference to the newly created object.**

- **The this variable is used to construct and initialize the properties of the object.**

Syntax:

```
function person (myName, myAge){
    this.name=myName;
    this.age = myAge;
}
```

This constructor can be used as follows:

```
var somePerson = new Person('surya',27);
```

- **Creates object named somePerson with surya as name and 27 as age.**

- **We may access the object properties in two ways:**

1. **somePerson.name**
2. **somePerson["name"]**

- **If a method is to be included in the object, it is initialized the same way as if it were a data property.**

Example, Suppose you want for person objects that listed the property values.

- **A function to serve as such a method could be written as follows:**

```
function display_person() {
        document.write("Person name:  ", this.name, "<br />");
        document.write("Age:  ", this.age, "<br />");
}
```

The following line must be added to the car constructor:

```
this.display = display_person;
```

- **Now the code somePerson.display( ); would produce the following:**

```
Person name: Surya
Age: 27
```

Object functions: Functions are just properties like any other property of an object.

```
function displayName(){
  document.write("My name is"+this.name);
}

function person(myName, myAge){
  this.name = myName;
  this.age = myAge;
  this.displayMe = displayName;
}
```

Then to call the function of an Object:

```
var pObj = new person('surya',28);
pObj.displayMe();

var pObj = new person('xxxxxx',2222);
pObj.displayMe();
```

Alternatively we may declare the function inside the constructor

```
function person(myName, myAge){
  this.name = myName;
  this.age = myAge;
  this.displayMe = function() { document.write("My name is
```

- **Properties of an object are accessed using a dot notation: *object.property***

- **Properties are not variables, so they are not declared, they are just name of values.**
- **They are used with object variables to access property values.**

- **The number of properties of an object may vary dynamically in JavaScript.**

- **A property for an object is created by assigning a value to that property.**

```
// Create an Object object
var my_car = new Object( );
// Create and initialize the make property
my_car.make = “Ford”;
// Create and initialize model
my_car.model = “Contour SVT”;
```

- **This code creates a new object my_car, with two properties, make and model.**

- **An abbreviated way to create an object and its proeprties.**

Example:

The object referenced with my_car could be created with the following statement:

```
var my_car = {make: "Ford",  model: " Contour SVT"  };
```

- **Objects can also be nested, you can create a new object that is property of my_car with properties of its own as follows:**

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

- **Properties can be accessed in two ways.**
- **Using object-dot-property notation.** ***var prop1 = my_car.make;***
- **Property names of an object can be accessed as if they were elements of an array, using property name as asubscript.** ***var prop2 = my_car["make"];***

- **The delete operator can be used to delete a property from an object.**

```
delete my_car.model
```

The for-in Loop

- **Syntax**

  for (*identifier* in *object*)

  statement or compound statement

- **The loop lets the identifier take on each property in turn in the object.**

- **Printing the properties in my_car:**

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

- **Result:**

  Name: make; Value: Ford

  Name: model; Value: Contour SVT

Arrays

- Arrays are Objects with some special functionality.
- Array elements can be primitive values or reference to other Objects.
- Array length is dynamic 'length' property stores the length.

- Array objects can be created in 2 ways:

1. By using 'new' operator and a call to a constructor named Array.
   a. var myList = new Array(3);
      myList[0] = 1;
      myList[1] = 2;
      myList[2] = 'hello';

   b. var myList1 = new Array(24,'a',3.14);
   In (b) an Array object of length 3 is created and initialised.
   Elements need not be of the same type.

   c. var myList2 = new Array(100);
   A new Array object of length 100 is created, whenever a single parameter is passed to the Array constructor that is taken as number of elements not the initial value.

2. var myList3 = [24,'a',3.14];
   Second way of creating Array object is with a literal array value, i.e., list of values enclosed in brackets.

Array Object Creation - Summary

- Arrays can be created using the new Array method
  - new Array with one parameter creates an empty array of the specified number of elements
    - new Array(10);
  - new Array with two or more parameters creates an array with the specified parameters as elements
    - new Array(10, 20);

- Literal arrays can be specified using square brackets to include a list of elements
  - var alist = [1, “ii”, “gamma”, “4”];

- Elements of an array do not have to be of the same type

Characteristics of Array Objects

- Lowest index of every JavaScript array is Zero.
- The length of an array is the highest subscript to which an element has been assigned, plus 1.

var myList=new Array(31)

- Example, If myList is an array with 30 elements and the following statement is executed, the new length of myList will be 31.

  var myList[30] = “Index 30”;

- Length property is both read- and write-accessible through the length property, added to every array object by the Array constructor.

  now myList.length gives 31 as result.

- As the length property is writable, we can set it to make the array any length we like, as In myList.length=2.

- An array can be made to grow by adding new elements or by setting its length property can lengthen, shorten or can also be made to shrink by setting its length property to a smaller value.

Array Example

```
<html>
<head>
<title> 01 - Array Constructors </title>
<script language='javascript'>
            var array1 = new Array();
            var array2 = new Array(5);
            //array2[-2] = 'SomeValue';
            var array3 = new Array(10, 12, 32);
            function displayArray(name, data)
            {
             document.write('Length of ' + name + ': ' + data.length +
'\n');
             document.write('Displaying ' + name + '...\n');
             for(var index in data){
               document.write( name + '[' + index + '] = ' + data[index]
+ '\n');
             }
             document.write('\n');
            }
</script>
</head>
```

Some Array methods:
concat(elements)
join() – form string
array.pop()
Array.push(ele)
Array.reverse()

```
<body><pre><script language='javascript'>
        displayArray('array1', array1);
        displayArray('array2', array2);
        displayArray('array3', array3);

        //Now, add this code =>
        array3[42] = 74;
        displayArray('array3', array3);

        //Truncating an array
        array3.length = 2;
        displayArray('array3', array3);
 </script></pre>
</body>
</html>
```

- The example below shows the dynamic nature of arrays in JavaScript as follows:
  This script has an array of names, which are in alphabetical order.

  It uses prompt to get new names, one at a time, and inserts them into the existing array.

  Our approach is to move elements down one at a time, starting at the end of the array,

  until the correct position for the new name is found.

  Then the new name is inserted, and the new array is displayed.

```
// insert_names.js
// This script has an array of names, name_list, whose values are in alphabetical order.
// New names are input through a prompt. Each new name is inserted into the name_list
// array, after which the new list is displayed.

// The original list of names
var name_list = new Array ("Al", "Betty", "Kasper", "Michael", "Roberto", "Zimbo");
var new_name, index, last;

// Loop to get a new name and insert it
while (new_name = prompt("Please type a new name", " ")) {
        last = name_list.length – 1;
```

```
// Loop to find the place for the new name
  while (last >= 0 && name_list[last] > new_name) {
        name_list[last + 1] = name_list[last];
        last--;
   }

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

// Display the new array
   document.write("<p><b>The new name listis:</b>  " , "<br />" );
   for(index = 0; index < name_list.length; index++)
        document.write(name_list[index], "<br />");
   document.write("</p>");
} //** end of the outer while loop
```

Array Methods

- join method:

  Converts all of the elements of an array to strings and catenates them into a single string.

  If no parameter is provided to join, the values in the new string are seperated by commas.

  ```
  var names = new Array[“Mary”, “Murray”, “Murphy”, “Max”];
  ……
  var name_string = names.join(“ :  ”);
  ```

  The value of the name_string is now “Mary : Murray : Murphy : Max”.

- reverse method:

  It reverses the order of the elements of the Array object through which it is called.

- sort method:

  Sort method coerces the elements of the array to strings, if they are not already strings, and sorts them alphabetically:

  ```
  names.sort( );
  ```

  The values of names is now [“Mary”, “Max”, “Murphy”, “Murray”].

- concat method:

  Catenates its actual parameters to the end of the Array object on which it is called.

```
var names = new Array[“Mary”, “Murray”,  “Murphy”, “Max”];
……
 var new_names = names.concat(“Moo”, “Meow”);
```

  The new_names array now has length 6.

- slice method:

  Slice method does for arrays what the substring method does for strings.

  It returns the part of the Array object specified by its parameters, which are used as subscripts.

  The returned array has the elements of the array object through which it is called from the first parameter up to, but not including the second parameter.

  Example:

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

- Value of list2 is now [4, 6]

- If slice is given just one parameter, the returned array has all of the elements of the object , starting with the specified index.

Example:

```
var list = [2, 4, 6, 8, 10];
.....
var list3 = list.slice(2);
```

- Value of list3 is now [6, 8, 10].

- When toString method is called through an Array object, each of the object is converted if necessary to a string.
- These strings are catenated by commas.
- This method behaves much like join.

Dynamic List Operations

- push
  - Add an element to the high end of the array.
- pop
  - Remove an element from the high end of the array.

  Example:

```
var list = [“Dasher”, “Dancer”, “Donner”, “Blitzen”];
var deer = list.pop();        //deer is “Blitzen”
list.push(“Blitzen”);
// This puts “Blitzen” back on the list.
```

- shift
  - Remove an element from the beginning of an array.
- unshift
  - Add an element to the beginning of an array.

  Example:

```
var deer = list.shift();        //deer is now “Dasher”
list.unshift(“Dasher”);
// This puts “Dasher” back on the list.
```

Two-dimensional Arrays

- A two-dimensional array in JavaScript is an array of arrays.
  - This need not even be rectangular shaped: different rows could have different length.

- Example nested_arrays.js illustrates two-dimensional arrays.

```
// Create an array object with three arrays as its elements
var nested_array = [[2, 4, 6], [1, 3, 5], [10, 20, 30]];
// Display the elements of the nested list
for (var row = 0; row <= 2; row++) {
      document.write("Row  ", row, ":  ");

 for (var col = 0; col <= 2; col++) {
      document.write(nested_array[row][col], " ");
 document.write("<br />");
 }
```

Function Fundamentals Summary

- Function definition syntax
  - A function definition consist of a header followed by a compound statement
  - A function header:
    - function *function-name*(*optional-formal-parameters*)
- return statements
  - A return statement causes a function to cease execution and control to pass to the caller.
  - A return statement may include a value which is sent back to the caller.
    - This value may be used in an expression by the caller.
  - A return statement without a value implicitly returns undefined.
- Function call syntax
  - Function name followed by parentheses and any actual parameters.
  - Function call may be used as an expression or part of an expression.
- Functions must defined before use in the page header.

Functions are Objects

- Functions are objects in JavaScript.
- Functions may, therefore, be assigned to variables and to object properties.
  - Object properties that have function values are methods of the object.
- Example

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

JavaScript functions

- To keep the browser from executing a script when the page loads, you can put your script into a function.
- A function will be executed by an event or by a call to the function.
- You may call a function from anywhere within a page (or even from other pages if the function is embedded in an external .js file).
- Functions can be defined both in the <head> and in the <body> section of a document. However, to assure that a function is read/loaded by the browser before it is called, it could be wise to put functions in the <head> section.

How to Define a Function

```
function functionname(var1,var2,...,varX)
{
    some code
}
```

The parameters var1, var2, etc. are variables or values passed into the function. Var1, var2,etc. are the formal parameters.

- We place all function definitions in the head of the HTML document.
  - Calls to functions appear in the document body.
- A function with no parameters must include the parentheses () after the function name.

Example

```
<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>
```

The function displaymessage() will be executed if the input button is clicked.

The return Statement

- The return statement is used to specify the value that is returned from the function.
- So, functions that are going to return a value must use the return statement.

The example below returns the product of two numbers (a and b):

```
<html>
    <head>
      <script type="text/javascript">
        function product(a,b)
       {
            return a*b;
        }
     </script>
    </head>

    <body>
       <script type="text/javascript">
        document.write(product(4,3));
       </script>
    </body>
</html>
```

- JavaScript functions are objects, so variables that references them can be treated as object references-they can be passed as parameters, be assigned to other variables and be the elements of the array.

- Consider the following example:

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

The Lifetime of JavaScript Variables

- If you declare a variable, using "var", within a function, the variable can only be accessed within that function.
- Variables explicitly declared in a function are **local variables.**
- When you exit the function, the variable is destroyed.
- You can have local variables with the same name in different functions, because each is recognized only by the function in which it is declared.
- If you declare a variable outside a function, all the functions on your page can access it.

- A variable not declared using var has global scope, visible throughout the page, even if used inside a function definition.
- A variable declared with var outside a function definition has global scope.
- A variable declared with var inside a function definition has local scope, visible only inside the function definition.
    - If a global variable has the same name, it is hidden inside the function definition.

Functions – parameters

- Parameters are passed by value, but when a reference variable is passed, the semantics are pass-by-reference.

- There is no type checking of parameters, nor is the number of parameters checked
  - excess actual parameters are ignored, excess formal parameters are set to undefined.

- All parameters are sent through a property array, arguments, which has the l**ength** property.
- By accessing arguements.length , a function can determine the No. Of actual parameters passed.

- Parameters named in a function header are called *formal parameters.*
- Parameters used in a function call are called *actual parameters.*
- Parameters are passed by value.

How to pass a variable to a function, and use the variable in the function.

```
<html>
  <head>
    <script type="text/javascript">
      function myfunction(txt)
      {
          alert(txt);
      }
    </script>
  </head>
  <body>
    <form>
        <input type="button" onclick="myfunction('Hello')" value="Call function">
    </form>
    <p>By pressing the button above, a function will be called with "Hello" as a parameter.
    The function will alert the parameter.</p>
  </body>
</html>
```

```
<html>
    <head>
        <script type="text/javascript">
         function myFunction()
         {
              return ("Hello world!");
         }
        </script>
    </head>
<body>
<script type="text/javascript">
       document.write(myFunction())
</script>
</body>
</html>
```

Parameter Passing Example

```
function fun1(my_list) {
        var list2 = new Array(1, 3, 5);
        my_list[3] = 14;
        ...
        my_list = list2;
        ...
}
...
var list = new Array(2, 4, 6, 8)
fun1(list);
```

- The first assignment changes list in the caller
- The second assignment has no effect on the list object in the caller
- Pass by reference can be simulated by passing an array containing the value

The sort Method, Revisited

- A parameter can be passed to the sort method to specify how to sort elements in an array
  - The parameter is a function that takes two parameters
  - The function returns a negative value to indicate the first parameter should come before the second
  - The function returns a positive value to indicate the first parameter should come after the second
  - The function returns 0 to indicate the first parameter and the second parameter are equivalent as far as the ordering is concerned
- Example median.js illustrates the sort method

```
// Function num_order
// Parameter: Two numbers
// Returns: If the first parameter belongs before the
//            second in descending order, a negative number
//          If the two parameters are equal, 0
//          If the two parameters must be
//            interchanged, a positive number
function num_order(a, b) {return b - a;}
// Sort the array of numbers, list, into
// ascending order
  num_list.sort(num_order);
```

Pattern Matching (RegExp)

- Regular expression is a string of special values that programmers can use to explicitly match a specific string of text.

Few regular expression characters:

a. '.' matches any singular character

b. '?' matches one or none of the preceding character.

c. '+' matches at least one of the preceding character.

d. '*' matches none or all of the preceding character.

e. '^' matches the absolute beginning of the string.

f. '$' matches the absolute end of the string.

g. '\w+' matches a whole word.

h. '\w' matches a "word" character.

i. '\W' matches a whitespace.

j. x|y matches one or the other of x or y.

k. [0..9] matches ONE number, ranging from 0 to 9

l. [A-Za-z] matches any letter, uppercase or lowercase

m. \. matches a period

n. \? Matches a question mark

o. \[matches a left square bracket

p. \| matches a “pipe” character

q. /something/g global (matches all instances)

r. /something/i ignore case

s. /something/gi – combining both i and g

Pattern Matching (RegExp) cont…

Regular expression can be created using
Exp = /:+/; //matches one or more
colons
Exp = new RegExp(“:+”); // same thing

```
<html>
 <head>
    <script language="javascript">
     validate = function(){
          val = document.forms[0].input.value;
          alert(val);
          exp = /a|e|i|o|u/;
          alert("First index of Oval is ::: "+val.search(exp));
     }
    </script>
 </head>
 <body>
     <form id="form1">
           <input id="input" type="text">
           <input id="Validate" type="button"
                    onClick="validate()" value="validate">
           </form>
 </body>
</html>
```

RegExp object has 3 methods:
1. test(): searches a string for a specified value, return true Or false

2. exec(): searches a string for a specifed value. Returns the text of the found value. If no match is found, it returns null

3. compile(): The compile method is used to change the RegExp.

Ex: var pat = new RegExp(“e”);
document.write
(pat.test(“hello”));
// returns true
pat.compile(“d”);
document.write
(pat.test(“hello”));
// returns false

Using Regular Expressions

- Regular expressions are used to specify patterns in strings.

- JavaScript provides two methods to use regular expressions in pattern matching
  - String methods
  - RegExp objects (not covered in the text).

- A literal regular expression pattern is indicated by enclosing the pattern in slashes.

- The **search method** returns the position of a match, if found, or -1 if no match was found.

Example Using search

•The simplest pattern-matching method is search, which takes a pattern as a parameter.
•The search method returns the position in the String object (through which it is called) at which the pattern matched.
•If there is no match, search returns –1.

Ex:

```
var str = "Rabbits are furry";
var position = str.search(/bits/);
if (position > 0)
        document.write("'bits' appears in position",
                position, "<br />");
else
        document.write(
                "'bits' does not appear in str <br />");
```

- This uses a pattern that matches the string ‘bits’
- The output of this code is as follows:

  'bits' appears in position 3

Characters and Character-Classes

- *Metacharacters* have special meaning in regular expressions
  - \ | ( ) [ ] { } ^ $ * + ? .
  - These characters may be used literally by escaping them with \
- Other characters represent themselves.
- A period matches any single character.
  - /f.r/ matches for and far and fir but not fr
- A character class matches one of a specified set of characters.
  - [*character set*]
  - List characters individually: [abcdef]
  - Give a range of characters: [a-z]
  - ^ at the beginning negates the class

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

Repeated Matches

- A pattern can be repeated a fixed number of times by following it with a pair of curly braces enclosing a count.

- A pattern can be repeated by following it with one of the following special characters
  - * indicates zero or more repetitions of the previous pattern.
  - + indicates one or more of the previous pattern.
  - ? indicates zero or one of the previous pattern.

- Examples
  - /\(\d{3}\)\d{3}-\d{4}/ might represent a telephone number
  - /[$_a-zA-Z][$_a-zA-Z0-9]*/ matches identifiers

Anchors

- Anchors in regular expressions match positions rather than characters.
  - Anchors are 0 width and may not take multiplicity modifiers.

- Anchoring to the end of a string.
  - ❖ ^ at the beginning of a pattern matches the beginning of a string.
    - A pattern is tied to a string position with an anchor. A pattern can be specified to match only at the beginning of the string by preceding it with a circumflex (^) anchor.
    - For example, the following pattern matches “pearls are pretty” but does not match “My pearls are pretty”:

      /^pearl/
  - ❖ **$** at the end of a pattern matches the end of a string.
    - For example, the following pattern matches “I like gold” but does not match “golden”:

      /gold$/
    - The $ in /a$b/ matches a $ character.

- Anchoring at a word boundary
  - \b matches the position between a word character and a non-word character or the beginning or the end of a string.
  - /\bthe\b/ will match ‘the’ but not ‘theatre’ and will also match ‘the’ in the string ‘one of the best’.

Pattern Modifiers

- Pattern modifiers are specified by characters that follow the closing / of a pattern.

- Modifiers modify the way a pattern is interpreted or used.

- The modifiers are specified as letters just after the right delimiter of the pattern.
- The **i modifier** makes the letters in the pattern match either uppercase or lowercase letters in the string. For example, the pattern /Apple/i matches
- ‘APPLE’, ‘apple’, ‘APPle’, and any other combination of uppercase and lowercase spellings of the word “apple.”

- The **x modifier** allows white space to appear in the pattern. Because comments are considered white space, this provides a way to include explanatory comments in the pattern.

For example, the pattern

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

is equivalent to

```
/\d+\s[A-Z][a-z]+/
```

Other Pattern Matching Methods

1) The **replace method** takes a pattern parameter and a string parameter.

- The replace method is used to replace substrings of the String object that match the given pattern.
- The replace method takes two parameters: the pattern and the replacement string.
- The **g modifier** can be attached to the pattern if the replacement is to be global in the string, in which case the replacement is done for every match in the string.

The matched substrings of the string are made available through the predefined variables $1, $2, and so on. For example, consider the following statements:

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

- Parentheses can be used in patterns to mark sub-patterns.
  - The pattern matching machinery will remember the parts of a matched string that correspond to sub-patterns.

2) **The match method** is the most general of the String pattern-matching methods. The match method takes a single parameter: a pattern.

- It **returns an array** of the results of the pattern-matching operation. If the pattern has the **g modifier,** the returned array has all of the substrings of the string that matched.
- If the pattern does not include the g modifier, the returned array has the match as its first element, and the remainder of the array has the matches of parenthesized parts of the pattern if there are any:

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

In this example, `matches` is 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 />");
```

The following is the value of the `matches` array after this code is interpreted:

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

3**) The split method** splits the object string using the pattern to specify the split points.

Ex: var str = “grapes:apples:oranges”;

var fruit = str.split(“:”);

In this example, fruit is set to [grapes, apples, oranges].

Errors in Scripts

- JavaScript errors are detected by the browser.
- Different browsers report this differently.
  - Firefox uses a special console.
- Support for debugging is provided .
  - In IE 7, the debugger is part of the browser.
  - For Firefox 2, plug-ins are available.
    - These include Venkman and Firebug.