MODULE IV

- Introduction to JavaScript and jQuery - The Basics of JavaScript: Overview of JavaScript, Object Orientation and JavaScript, General Syntactic Characteristics- Primitives, Operations, and Expressions, Screen Output and Keyboard Input, Control Statements, Object Creation and Modification, Arrays, Functions. Callback Functions, Java Script HTML DOM. Introduction to jQuery: Overview and Basics.

Overview of JavaScript: Origins

- Originally developed by Netscape
- JavaScript was invented by Brendan Eich in 1995
- Joint Development with Sun Microsystems in 1995
- can be **run on any operating systems** and almost all web browsers.
- ECMA-262 edition 3 is the current standard
  - Edition 4 is under development
- Supported by Netscape, Mozilla, Internet Explorer

INTRODUCTION

- JavaScript is one of the **3 languages** all web developers **must** learn:

  1. **HTML** to define the content of web pages
  2. **CSS** to specify the layout of web pages
  3. **JavaScript** to program the behavior of web pages

JAVASCRIPT FEATURES

- Initially called LIVESCRIPT
- Allows pages to become dynamic & interactive
- Can embedded in a web page or linked in as an external file
- Used to validate the data on the web page before submitting it to the server.
- Used to create cookies.

JavaScript Components

- JavaScript can be divided into 3 parts
  - ➢ Core
    - – heart of the language
    - – Including its operators, expressions statements & subprograms
  - ➢ Client-side
    - – Collection of objects supporting browser control and user interaction (mouse clicks, keyboard etc)
  - ➢ Server-side
    - – Collection of objects that make the language useful on a web servers
- Server-side JavaScript is used far less frequently than Client-side JavaScript

Java and JavaScript

Java	JavaScript
Supports Object-oriented Programming Language	Object based Programming Language, its object model is different from that of java & C++
Strongly typed language, types are all known at compile time & operand types are checked for compatibility	Weakly typed language, Variables need not be declared, dynamically typed
Objects in java are static in the sense that their collection of data members & methods are fixed at compile time	Objects are dynamic – the number of data members & methods of an object can change during execution
Java uses the concept of classes and objects that makes reuse of the code easier	there is no such thing in JavaScript.

Similarity between java & JavaScript

- Syntax of their expressions , assignment statements & control statements

Uses of JavaScript

- Provide **programming capability** at both server & the client ends of a web connection
- 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

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

XHTML/JavaScript Documents

- When JavaScript is embedded in an XHTML document, the browser must interpret it
- Two locations for JavaScript serve 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

Object Orientation and JavaScript

- Not an object-oriented programming language
- Object-based language
- Doest not have classes
- Cannot have class based inheritance
- Prototype – based inheritance
- Cannot supports polymorphism

JavaScript Objects (1)

- 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 value** is a value that has **no properties or methods**.
- Primitive values are often implemented directly in hardware resulting in faster operations on their values

- Root object in JavaScript is Object
- The Object object is the ancestor of all objects in a JavaScript program
  – Object has no data properties, but several method properties

JavaScript in XHTML

- Directly embedded

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

- Indirect reference

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

  – This is the preferred approach

JavaScript identifiers

- Identifiers or names are similar to those of other common programming languages
- Must begin with a letter, an underscore or a dollar sign
- No length limitations for identifiers
- Case sensitive

JavaScript reserved words

- 25 reserved words
- Break,case,catch,continue,delete,do,return,switch, for, new, while etc
- Another collection of words is reserved for future use in Javascript
- Has a large collection of predefined words, including alert,open ,java & self

JavaScript comments

1. two adjacent slashes(//) appear on a line ,the rest of the line is considered as comment
2. /*…………………*/ (single & multiple line comments)

Issues in embedding JavaScript in XHTML document

- There are some browsers still in use recognize the <script> tag but do not have JavaScript interpreters
  - Simply ignore the contents of the script element & cause no problems.
- Old browsers does not recognize the script tag, simply read as text
- So enclose the contents of all script elements in XHTML comments to avoid this problem

JavaScript in XHTML: CDATA (1)

- The <![CDATA[…]]> block is intended to hold data that should not be interpreted as XHTML
- Using this should allow any data (including special symbols and --) to be included in the script
- This, however does not work, at least in Firefox:

```
<script type="text/javascript">
   <![CDATA[
     …JavaScript here…
   ]]>
</script>
```

- The problem seems to be that the CDATA tag causes an internal JavaScript error

JavaScript in XHTML: CDATA (2)

- This does work in Firefox

```
<script type="text/javascript">
    /*<![CDATA[ */
      ...JavaScript here...
    /*]]> */
</script>
```

- The comment symbols do not bother the XML parser (only /* and */ are 'visible' to it)
- The comment symbols protect the CDATA markers from the JavaScript parser

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;
```

JavaScript Primitives (1)

- A **primitive value** is a value that has no properties or methods.
- A **primitive data type** is data that has a primitive value.
- JavaScript defines 5 types of primitive data types:
  - ➢ String
  - ➢ number
  - ➢ boolean
  - ➢ undefined
  - ➢ null

  Types.html

JavaScript Primitives (2)

- Primitive values are immutable (they are hardcoded and therefore cannot be changed).
- if x = 3.14, you can change the value of x. But you cannot change the value of 3.14.

JavaScript Objects

- In JavaScript, almost "everything" is an object.
- Booleans, Numbers, Strings etc can be objects (if defined with the **new** keyword)
- All JavaScript values, except primitives, are objects.

Object Properties

- The named values, in JavaScript objects, are called **properties**.
-

Property	**value**
firstName	John
lastName	Doe

Object Methods (1)

- Methods are **actions** that can be performed on objects.
- Object properties can be both primitive values, other objects, and functions.
- An **object method** is an object property containing a **function definition**.

Object Methods (2)

Property	**value**
firstName	John
lastName	Doe
fullName	function() {return this.firstName + " " + this.lastName;}

- JavaScript objects are containers for named values, called properties and methods.

Wrapper Objects (1)

- JavaScript includes predefined objects
- that are closely related to the Number, String, and Boolean types, named Number, String, and Boolean,
- These objects are called wrapper objects.

Wrapper Objects (2)

- Each contains a property that stores a value of the corresponding primitive type.
- purpose of the wrapper objects is to provide
    - ➢ properties and methods that are convenient for use with values of the primitive types.
    - ➢ In the case of Number, the properties are more useful; in the case of String, methods are more useful
- Because JavaScript coerces values between the Number type primitive values and Number objects and between the String type primitive values and String objects
- methods of Number and String can be used on variables of the corresponding primitive types.

The difference between primitives and objects is shown in the following example.

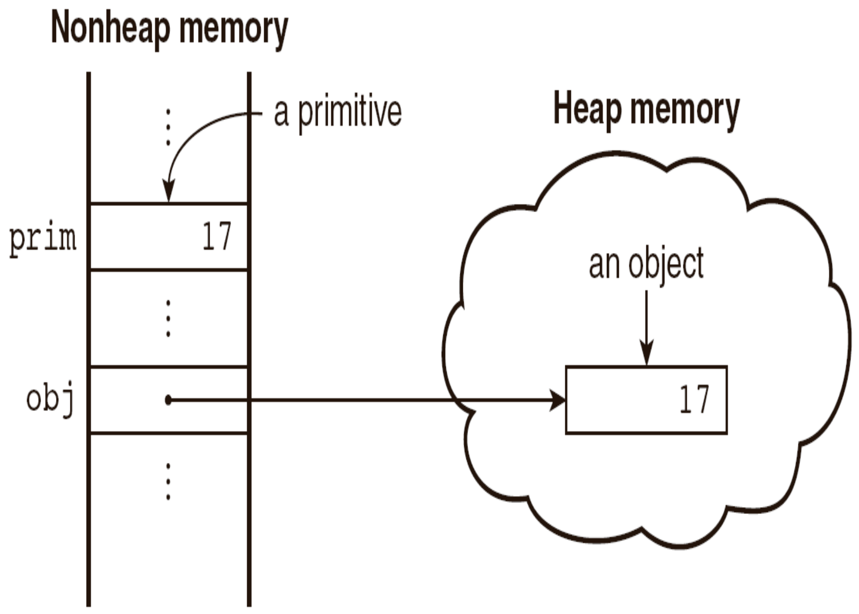


Figure 4.1 Primitives and objects

- prim is a primitive variable with the value 17
- obj is a Number object whose property value is 17.
- Figure shows how prim and obj are stored.

Numeric and String Literals

- Number values are represented internally as double-precision floating-point values
  - Number literals can be either integer or float
  - Float values may have a decimal and/or and exponent
- A String literal is 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 “ ” has no characters

Other Primitive Types

- Null
  - A single value, null
  - `null` is a reserved word
  - 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 an 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
- Boolean
  - Two values: true and false

Declaring Variables

- JavaScript is *dynamically typed*, that is, variables do not have declared types
  - A variable can hold different types of values at different times during program execution
- A variable is declared using the keyword var

```
var counter,index,pi = 3.14159265;
```

Numeric Operators

- Standard arithmetic

  ```
  +    *    -    /    %
  ```

- Increment and decrement

  ```
  --      ++
  ```

  Increment and decrement differ in effect when used before and after a variable

Precedence of Operators (1)

- Precedence rules
  - ➢ specify which operator is evaluated first when two operators **with different precedence** are adjacent in an expression.
- Associativity rules
  - ➢ specify which operator is evaluated first when two operators **with same precedence** are adjacent in an expression.

Precedence of Operators (2)

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

The `Math` Object

- Provides a collection of properties and methods useful for Number values
- This includes the trigonometric functions such as `sin` and `cos`
- When used, the methods must be qualified, as in `Math.sin(x)`

The `Number` Object

- Properties
  - `MAX_VALUE`
  - `MIN_VALUE`
  - `NaN`
  - `POSITIVE_INFINITY (special value to represent infinity)`
  - `NEGATIVE_INFINITY`
  - `PI`
- Operations resulting in errors return `NaN` `(not a number)`
  - Use `isNaN(a)` to test if a is `NaN`
- toString method converts a number to string

String Catenation Operator

- The operator + is the string catenation operator
- Strings are not stored or treated as array of characters, rather they are unit scalar values.
- In many cases, other types are automatically converted to string

Implicit Type Conversion (1)

- JavaScript attempts to convert values in order to be able to perform operations
- “August “ + 1977 causes the number to be converted to string and a concatenation to be performed
- 7 * “3” causes the string to be converted to a number and a multiplication to be performed

Implicit Type Conversion (2)

- null is converted to 0 in a numeric context, undefined to NaN
- 0 is interpreted as a Boolean false, all other numbers are interpreted as true
- The empty string is interpreted as a Boolean false, all other strings as Boolean true
- undefined, Nan and null are all interpreted as Boolean false

Explicit Type Conversion

- Explicit conversion of string to number
  - Number(aString)   eg:- var number=Number(aString);
- parseInt and parseFloat convert the beginning of a string to integer literal & floating point literal

`String` Properties and Methods

- One property: length
  - ➢ Number of characters in a string is stored in the length property

    var str = “George”;

    var len = str.length;

    Here len= 6

- Character positions in strings begin at index 0

String Methods

Method	Parameters	Result
charAt	A number	Returns the character in the String object that is at the specified position
indexOf	One-character string	Returns the position in the String object of the parameter
substring	Two numbers	Returns 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

The `typeof` Operator

- Returns the type of its single operand
- Ie, it returns “number” or “string” or “boolean” for primitive types
- Returns “object” for an object or null
- Objects do not have types
- If the operand is a variable that has not been assigned a value,typeof evaluates to “undefined”
- Two syntactic forms
  - `typeof x`
  - `typeof(x)`
  - `Both are equivalent`

Assignment Statements

- simple assignment indicated by =
- Compound assignment with
  - +=   -=   /=   *=   %=   …
- a += 7 means the same as
- a = a + 7

4.4 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
  - var today= new Date();
  - This creates a Date object for the time at which it was created

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

Window and Document

- The Window object represents the window in which the document containing the script is being displayed
- The Document object represents the document being displayed using DOM
- Window has two properties
  - window refers to the Window object itself
  - document refers to the Document object
- The Window object is the default object for JavaScript, so properties and methods of the Window object may be used without qualifying with the class name

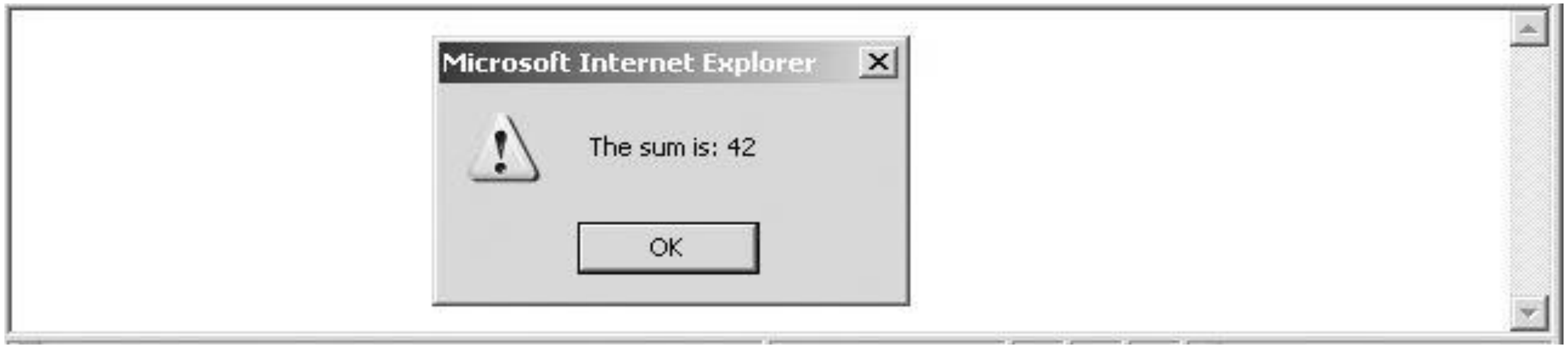
Screen Output and Keyboard Input

- Standard output for JavaScript embedded in a browser is the window displaying the page in which the JavaScript is embedded
- write method of the Document object write its parameters to the browser window
- The output is interpreted as HTML by the browser
- If a line break is needed in the output, use <br/>

The alert Method

- The alert method opens a dialog box with a message
- The output of the alert is *not* XHTML, so use new lines rather than <br/>

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

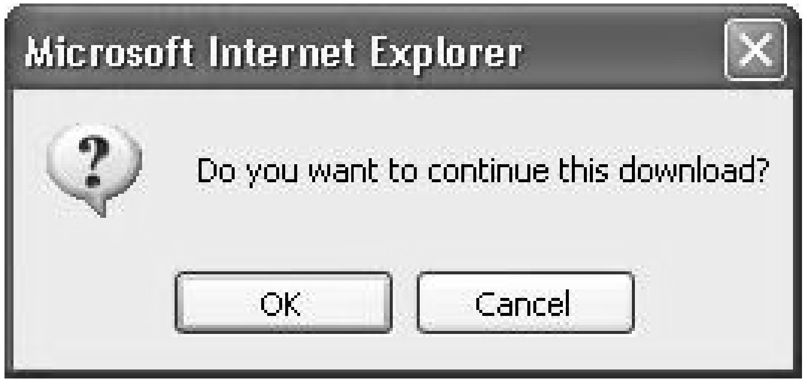


The sum is: 42

The confirm Method

- The confirm methods displays a message provided as a parameter
  - The confirm dialog has two buttons: OK and Cancel
- If the user presses OK, true is returned by the method
- If the user presses Cancel, false is returned

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



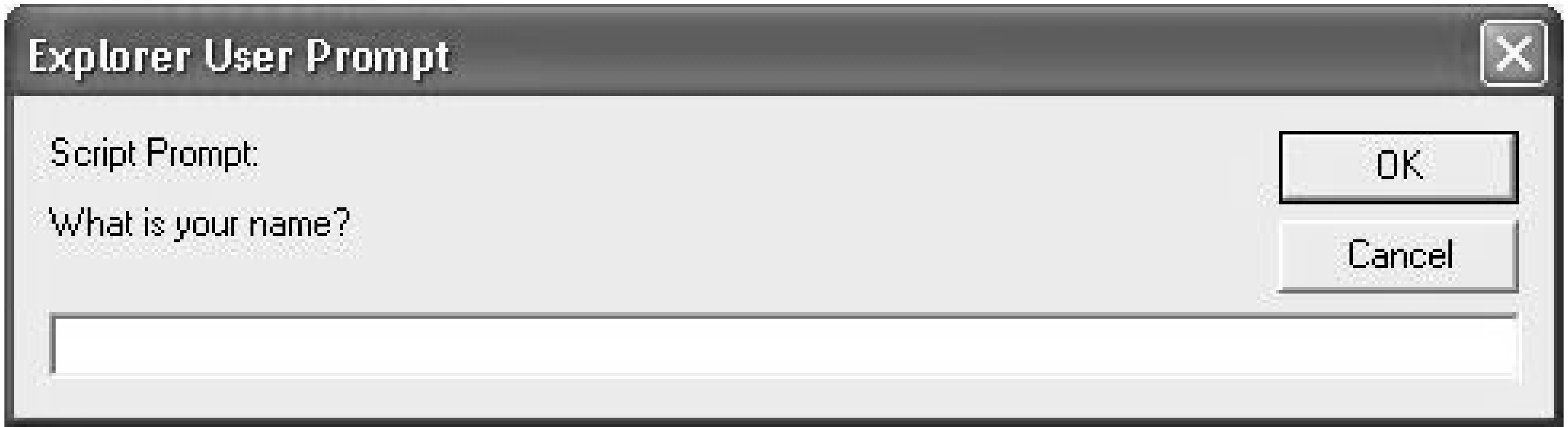
Do you want to continue this download?

The prompt Method

- This method displays its string argument in a dialog box
  - A second argument provides a default content for the user entry area
- The dialog box has an area for the user to enter text

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

- The method returns a String with the text entered by the user



Script Prompt:

What is your name?

Where JavaScript is placed

• most preferred ways to include JavaScript in an HTML file are as follows −

1) Script in <head>...</head> section.

2) Script in <body>...</body> section.

3) Script in <body>...</body> and <head>...</head> sections.

4) Script in an external file and then include in <head>...</head> section.

1) Script in <head>...</head> section.

- to have a script run on some event, such as when a user clicks somewhere (hello1.html)

```
<html>
  <head>
      <script type="text/javascript">
        function sayHello()
         {
          alert("Hello World")
        }
       </script>
      </head>
    <body>
    <input type="button" onclick="sayHello()" value="Say Hello" />
  </body>
  </html>
```

2) Script in <body>...</body> section.

- a script to run as the page loads so that the script generates content in the page, then the script goes in the <body> portion of the document.

```
<html>
 <head>
 </head>
    <body>
        <script type="text/javascript">
            document.write("Hello World")
              </script>
           <p>This is web page body </p>
        </body>
</html>
```

Script in <body>...</body> and <head>...</head> sections(1)

```
<html>
  <head>
    <script type="text/javascript">
             function sayHello()
         {
           alert("Hello World")
         }
        </script>
```

Script in <body>...</body> and <head>...</head> sections(2)

```
</head>
    <body>
    <script type="text/javascript">
      document.write("Hello World")
     </script>
     <input type="button" onclick="sayHello()" value="Say Hello" >
      </body>
</html>
```

External Scripts

- Place the code in a separate file
- Updation is easy
- <script > tag is not needed
- .js extension

4.6 Control Statements

- A *compound statement* in JavaScript is a sequence of 0 or more statements enclosed in curly braces
  - Compound statements can be used as components of control statements allowing multiple statements to be used where, syntactically, a single statement is specified
- A *control construct* is a control statement including the statements or compound statements that it contains

4.6.1 Control Expressions

- A control expression has a Boolean value
  - An expression with a non-Boolean value used in a control statement will have its value converted to Boolean automatically
- Comparison operators
  - ➢ `==` `!=` `<` `<=` `>` `>=`
  - ➢ `===` compares identity of values or objects
  - ➢ 3 == ‘3’ is true due to automatic conversion
  - ➢ 3 === ‘3’ is false (checking the data type also)
- Boolean operators

  `&&` `||` `!`
- Warning! A Boolean object evaluates as true
  - Unless the object is null or undefined

4.6.2 Selection Statements

- The if-then and if-then-else are similar to that in other programming languages, especially C/C++/Java

4.6.3 switch Statement Syntax

```
switch (expression)
 {
case value_1:
     // statement(s)
case value_2:
     // statement(s)
...
[default:
     // statement(s)]
}
```

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

4.6.4 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*)

- Day2.html
- Sum10.html
- Palicheck.html

4.7 Object Creation and Modification

- The new expression is used to create an object
  - This includes a call to a *constructor*
  - The new operator creates a blank object, the constructor creates and initializes all properties of the object
- Properties of an object are accessed using a dot notation: *object*.*property*
- Properties are not variables, so they are not declared
- The number of properties of an object may vary dynamically in JavaScript

4.7 Dynamic Properties

- Create my_car and add some properties

```
// 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";
```

- The delete operator can be used to delete a property from an object
  - `delete my_car.model`

4.7 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
```

4.8 Arrays

- Arrays are lists of elements indexed by a numerical value
- Array indexes in JavaScript begin at 0
- Arrays can be modified in size even after they have been created

4.8 `Array` Object Creation (1)

- 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)
- Var my_list = new Array(1,2,”three”,”four”);
- Usual way to create any object is with new operator & a call to a constructor
- In the case of arrays, the constructor is named Array:

4.8 `Array` Object Creation (2)

- Literal arrays can be specified using square brackets to include a list of elements

  var a _list = [1, 2, “three”, “four”];
- Elements of an array do not have to be of the same type

4.8 Characteristics of `Array` Objects (1)

- Lowest index of every JavaScript array is zero
- The length of an array is one more than the highest index to which a value has been assigned or the initial
- my_list[47] = 2222;
- New length of my_list will be 48

4.8 Characteristics of `Array` Objects (2)

- Assignment to an index greater than or equal to the current length simply increases the length of the array
- Only assigned elements of an array occupy space
  - Suppose an array were created using new Array(200)
  - Suppose only elements 150 through 174 were assigned values
  - Only the 25 assigned elements would be allocated storage, the other 175 would not be allocated storage

4.8 `Array` Methods

- join
- reverse
- sort
- concat
- slice

4.8.1 join method

- Converts all elements of an array to strings & catenates them into a single string
- If no parameter is provided to join, the values in the new string are separated by commas
- Join.html

4.8.2 Sort method

- JavaScript array **sort()** method sorts the elements of an array.
- Sort.html

4.8.3 Reverse Method

- Reverses the order of the elements of an array ie, the first becomes the last, and the last becomes the first.
- Reverse.html

4.8.4 concat

- Javascript array **concat()** method returns a new array comprised of this array joined with two or more arrays.
- Concat.html

4.8.4 Slice

- Extracts a section of an array and returns a new array.
- Slice.html

4.8 Dynamic List Operations (1)

- push
  - Add to the end
- pop
  - Remove from the end
- shift
  - Remove from the front
- unshift
  - Add to the front

4.8 Dynamic List Operations (2)

- Var list = [“ramu”, “rani”, “raj”];

  var deer = list.pop(); // deer is “raj”

  list.push(“raj”); // this puts “raj” back to list
- Shift & unshift remove & add an element to the beginning of an array

  var deer = list.shift(); // deer is now “ramu”

  list.unshift(“ramu”); // this puts “ramu” back to list

4.9 Function Fundamentals (1)

- 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

4.9 Function Fundamentals (2)

- 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

4.9 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
```

4.9 Parameters

- 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
  - For an object parameter, the reference is passed, so the function body can actually change the object
  - However, an assignment to the formal parameter will not change the actual parameter

callback function

- is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action.

```
function greeting(name)
 {
   alert('Hello ' + name);
}
function processUserInput(callback)
 {
var name = prompt('Please enter your name.');
   callback(name);
 }
processUserInput(greeting);
```

DOM

- Document Object Model
- When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.
- Used to acess information contained in HTML document such as forms & build such documents dynamically.
- is a W3C (World Wide Web Consortium)
- Defines a standard for accessing documents

W3C DOM

- Seperated into 3 different parts

i) Core DOM – std model for all document types

ii) XML DOM

iii) HTML DOM

HTML DOM model is constructed as a tree of **Objects**:

```
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head> <title> A simple document </title>
  </head>
  <body>
    <table>
      <tr>
        <th> Breakfast </th>
        <td> 0 </td>
        <td> 1 </td>
      </tr>
      <tr>
        <th> Lunch </th>
        <td> 1 </td>
        <td> 0 </td>
      </tr>
    </table>
  </body>
</html>
```

Figure 5.1 shows the DOM structure for this table.

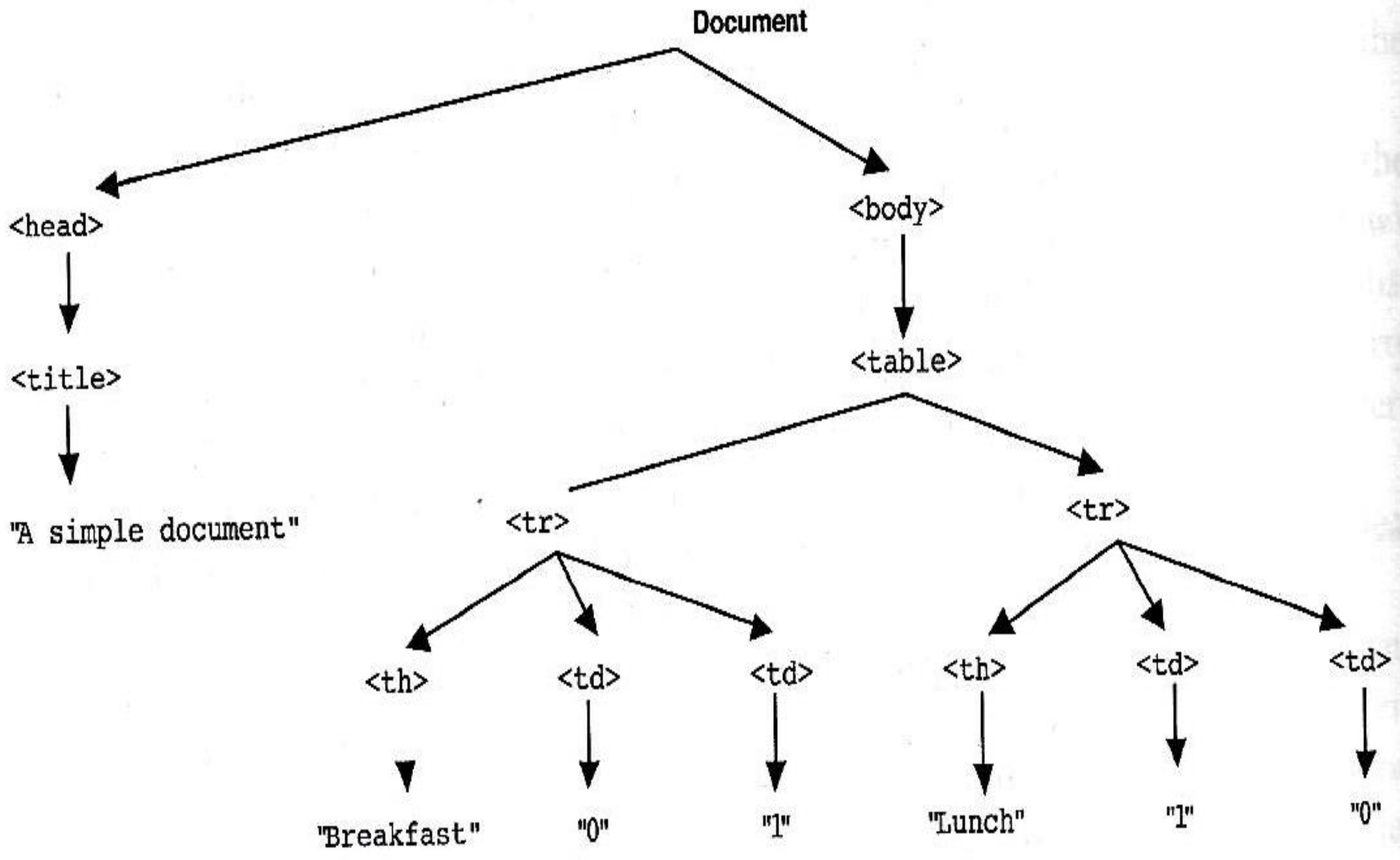


Figure 5.1 The DOM structure for a simple document

jQuery Introduction

- jQuery is a JavaScript Library.
- greatly simplifies JavaScript programming.
- is easy to learn.
- jQuery is a lightweight, "write less, do more", JavaScript library.

- The purpose of jQuery is to make it much easier to use JavaScript on your website.
- jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that we can call with a single line of code.
- jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.

- The jQuery library contains the following features:
- HTML/DOM manipulation
- CSS manipulation
- HTML event methods
- Effects and animations
- AJAX
- Utilities

Why jQuery?

- There are lots of other JavaScript frameworks out there, but jQuery seems to be the most popular, and also the most extendable.
- Many of the biggest companies on the Web use jQuery, such as:
- Google
- Microsoft
- IBM
- Netflix

jQuery Syntax

- Basic syntax is: **$(*selector*).*action*()**
- A $ sign to define/access jQuery
- A (*selector*) to "query (or find)" HTML elements
- A jQuery *action*() to be performed on the element(s)

Examples:

- $("this").hide() - hides the current element.
- $("p").hide() - hides all <p> elements.
- $(".test").hide() - hides all elements with class="test".
- $("#test").hide() - hides the element with id="test“.

The Document Ready Event

- all jQuery methods are inside a document ready event:
- allows us to execute a function when the document is fully loaded
- $(document).ready(function()

```
 {

   // jQuery methods go here...

});
```

Commonly Used jQuery Event Methods

- **click()**
- The function is executed when the user clicks on the HTML element.
-
```
$("p").click(function(){
    $(this).hide();
});
```

- **dblclick()**

```
$("p").dblclick(function(){
    $(this).hide();
 });
```

- **mouseenter()**
- The function is executed when the mouse pointer enters the HTML element:
- Example
- $("#p1").mouseenter(function()

```
 {
   alert("You entered p1!");
});
```

- **mouseleave()**
- The function is executed when the mouse pointer leaves the HTML element:
- Example

-
  ```
  $("#p1").mouseleave(function(){
      alert("Bye! You now leave p1!");
  });
  ```