JavaScript

JAVASCRIPT

- JavaScript is used in millions of Web pages to improve the design, validate forms, detect browsers, create cookies, and much more.
- JavaScript is the most popular scripting language on the internet, and works in all major browsers, such as Internet Explorer, Firefox, Opera,etc.

Are Java and JavaScript the Same?

- NO
- Java and JavaScript are two completely different languages in both concept and design!
- Java (developed by Sun Microsystems) is a powerful and much more complex programming language - in the same category as C and C++.

JavaScript

- The original name for JavaScript was "LiveScript"
- The name was changed when Java became popular
- Statements in JavaScript resemble statements in Java, because both languages borrowed heavily from the C language
  - JavaScript easy for Java programmers to learn
  - JavaScript *is* a complete, full-featured, complex language
- JavaScript is seldom used to write complete "programs"
  - Instead, small bits of JavaScript are used to add functionality to HTML pages
  - JavaScript is often used in conjunction with HTML "forms"
- JavaScript is based on objects

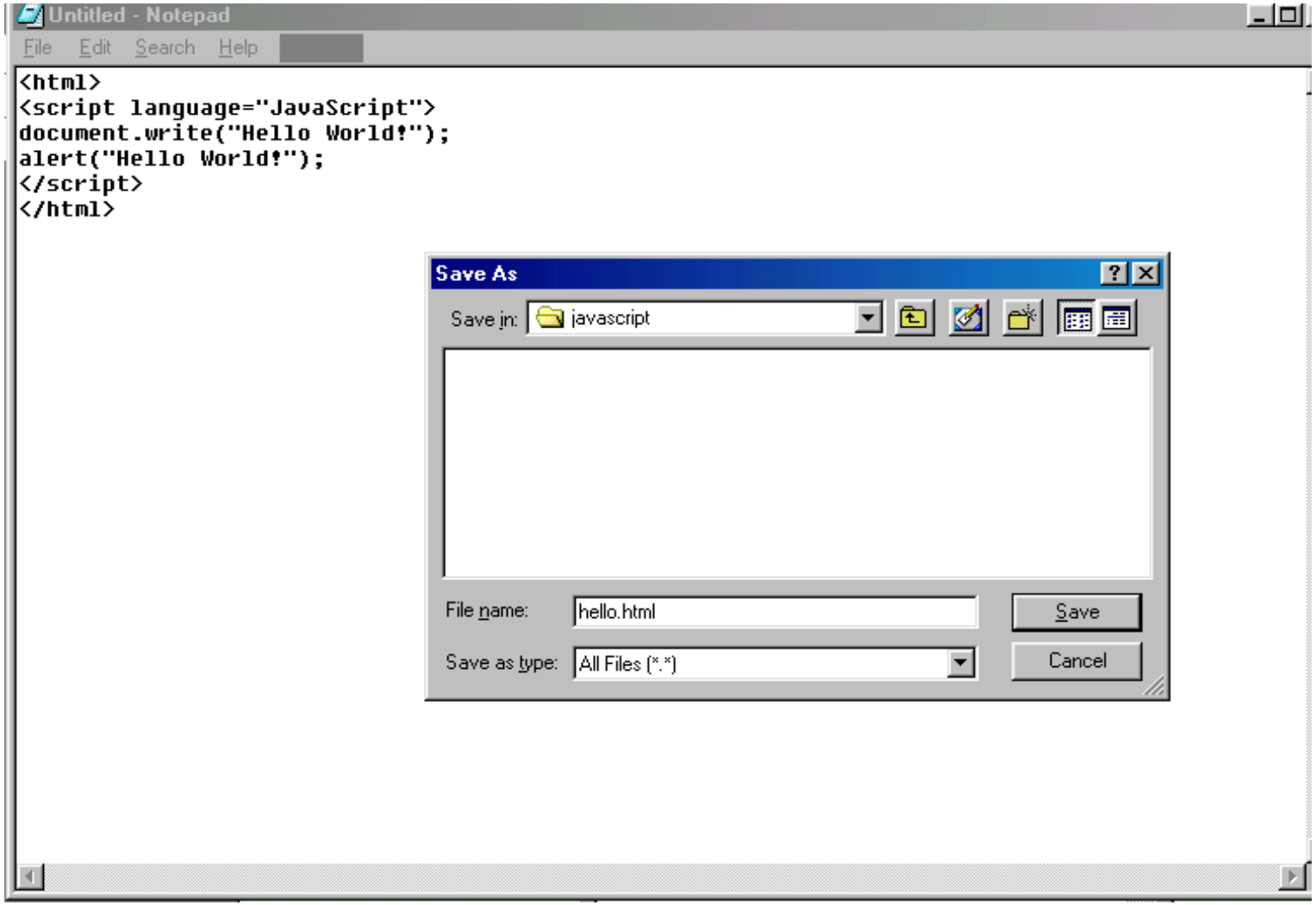
Using JavaScript in a browser

- JavaScript code is included within <script> tags:
  - <script type="text/javascript">
    document.write("<h1>Hello World!</h1>") ;
    </script>
  - The type attribute allows use of other scripting languages **(but JavaScript is the default)**
  - The above code is same as thing as putting <h1>Hello World!</h1> in the same place in the HTML document
  - The semicolon at the end of the JavaScript statement is **optional**
    - You need semicolons if you put two or more statements on the same line

- JavaScript is based on objects

How to Put a JavaScript Into an HTML Page?

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


Untitled - Notepad
File Edit Search Help
<html>
<script language="JavaScript">
document.write("Hello World!");
alert("Hello World!");
</script>
</html>
Save As
Save in: javascript
File name: hello.html
Save
Save as type: All Files (*.*)
Cancel

JavaScript Popup Boxes

- 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.

```
<script>
  alert("Hello World!")
</script>
```

JavaScript Popup Boxes

- Prompt Box
  - A prompt box is often used if you want the user to input a value before entering a 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.

```
<html>
<script type="text/javascript">
ans = 0;
firstnum = 0;
secondnum = 0;
firstnum = prompt("Enter the first number",0);
secondnum = prompt("Enter the second number",0);
ans = firstnum * secondnum;
document.write(ans);
</script>
</html>
```

Conditional Statements

- **if**
- **if...else**
- **if...else if....else**
- **switch**

```
<html>
<script language="JavaScript">
var ans = 0;
var firstnum = 0;
var secondnum = 0;
var whattodo;
firstnum = prompt("Enter the first number",0);
secondnum = prompt("Enter the second number",0);
whattodo = prompt("Enter * or /","");
if (whattodo == "*")
   {
   ans = firstnum * secondnum;
   }
else
   {
   ans = firstnum / secondnum;
   }
document.write("The answer is ", ans);
</script>
</html>
```

equal, use = =

```
<html>
<script language="JavaScript">
if (navigator.appName == "Netscape")
     {
      document.write("The browser is Netscape Navigator<BR>");
      document.write("Netscape is in use!!!");
     }
  else
     {
      document.write(“it may be Internet Explorer<BR>");
      }
</script>
</body>
</html>
```

```
<html>
<script language="JavaScript">
var data_input = 0;
data_input=prompt("How many times should I write the line?",0);
var ct=1;
while (ct <= data_input)
   {
    document.write("This is number " + ct + "<br>");
    ct = ct + 1;
   }
document.write("<br>" + "This is the end!!!")
</script>
</html>
```

The + means concatenation notes.

Where to put JavaScript

- JavaScript can be put in the <head> or in the <body> of an HTML document
  - JavaScript *functions* should be defined in the <head>
    - This ensures that the **function is loaded before it is needed**
  - JavaScript in the **<body>** will be executed as the **page loads**
- JavaScript can be put in a separate **.js** file
  - <script src="myJavaScriptFile.js"></script>
  - An external .js file lets you use the same JavaScript on multiple HTML pages
  - The external .js file cannot itself contain a <script> tag
- JavaScript can be put in an HTML *form object*, such as a button
  - This JavaScript will be executed when the form object is used

A Simple Script

```
<html>
<head><title>First JavaScript
  Page</title></head>
<body>
<h1>First JavaScript Page</h1>
<script type="text/javascript">
  document.write("<hr>");
  document.write("Hello World Wide Web");
  document.write("<hr>");
</script>
</body>
</html>
```

Embedding JavaScript

```
<html>
<head><title>First JavaScript Program</title></head>
<body>
<script type="text/javascript"
        src="my.js"></script>
</body>
</html>
```

my.js

```
document.write("<hr>");
document.write("Hello to CBES course");
document.write("<hr>");
```

- Use the **`src`** attribute to include JavaScript codes from an external file.
- The included code is inserted in place.

```
<html>
<script language="JavaScript">
monthArray = new Array(12);
monthArray[1]="January";
monthArray[2]="February";
monthArray[3]="March";
monthArray[4]="April";
monthArray[5]="May";
monthArray[6]="June";
monthArray[7]="July";
monthArray[8]="Auguest";
monthArray[9]="September";
monthArray[10]="October";
monthArray[11]="November";
monthArray[12]="December";
var user_month = 0;
var user_day = 0;
var user_year = 0;
user_month = prompt("Enter the month as a number",0);
user_day = prompt("Enter the day",0);
user_year = prompt("Enter the year",0);
document.write("The date is " + monthArray[user_month] + " " + user_day + ", " + user_year);
</script>
</html>
```

Operators

- String operator:
  +
- The conditional operator:
  condition ? ***value_if_true*** : ***value_if_false***
- Special equality tests:

  == and != try to convert their operands to the same type before performing the test
- Additional operators:
  new typeof void delete

Comments

- Comments are as in C or Java:
  - Between `//` and the end of the line
  - Between `/*` and `*/`

Statements

- Most JavaScript statements are borrowed from C
  - Assignment: greeting = "Hello, " + name;
  - Compound statement:
    { ***statement***; ...; ***statement*** }
  - If statements:
    if (***condition***) ***statement***;
    if (***condition***) ***statement***; else ***statement***;
  - Familiar loop statements:
    while (***condition***) ***statement***;
    do ***statement*** while (***condition***);
    for (***initialization***; ***condition***; ***increment***) ***statement***;

Statements, II

- The switch statement:

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

- Other familiar statements:
  - break;
  - continue;
  - The empty statement, as in ;; or { }

Exception handling, I

- Exception handling in JavaScript is almost the same as in Java
- throw **expression** creates and throws an exception
  - The expression is the value of the exception, and can be of any type
- **try** {
  statements to try
  } **catch (e)** { // Notice: no type declaration for **e**
  exception handling statements
  } **finally** { // optional, as usual
  code that is always executed
  }
  - With this form, there is only one **catch** clause

Exception handling, II

- try {
  statements to try
  } catch (***e*** if ***test1***) {
  exception handling for the case that test1 is true
  } catch (***e*** if ***test2***) {
  exception handling for when test1 is false and test2 is true
  } catch (***e***) {
  exception handling for when both test1and test2 are false
  } finally { // optional, as usual
  code that is always executed
  }
- Typically, the test would be something like
  e == "InvalidNameException"

Object literals

- Don’t declare the **types of variables** in JavaScript
- JavaScript has object literals, written with syntax:
  - { **name1** : **value1** , ... , **nameN** : **valueN** }
- Example :
  - car = {myCar: "Saturn", 7: "Mazda",
    getCar: CarTypes("Honda"), special: Sales}
    - The fields are myCar, getCar, 7 (this is a legal field name) , and special
    - "Saturn" and "Mazda" are Strings
    - CarTypes is a function call
    - Sales is a **variable** you defined earlier
  - Example : document.write("I own a " + car.myCar);

JavaScript keywords	Reserved for future use	Reserved for browser
break	abstract	alert
case	boolean	blur
catch	byte	closed
continue	char	document
default	class	focus
delete	const	frames
do	debugger	history
else	double	innerHeight
finally	enum	innerWidth
for	export	length
function	extends	location
if	final	navigator
in	float	open
instanceof	goto	outerHeight
new	implements	outerWidth
return	import	parent
switch	int	screen
this	interface	screenX
throw	long	screenY
try	native	statusbar
typeof	package	window
var	private	
void	protected	
while	public	
with	short	
	static	
	super	

Functions

- Functions should be defined in the <head> of an HTML page, to ensure that they are loaded first
- The syntax for defining a function is:
  function *name*(*arg1*, …, *argN*) { *statements* }
  - The function may contain return ***value***; statements
  - Any variables declared within the function are local to it
- The syntax for calling a function is just
  ***name*(*arg1*, …, *argN*)**
- Simple parameters are passed *by value*, objects are passed ***by reference***

Creating Objects

- JavaScript is not an OOP language.
- “**prototype**” is the closest thing to "class" in JavaScript.
- It is also possible to emulate "inheritance" in JavasScript.

Three ways to create an object

- You can use an object literal:
  - var course = { number: “CI211", teacher: "Dr. XYZ" }
- You can use new to create a “blank” object, and add fields to it later:
  - var course = new Object();
    course.number = "CI211";
    course.teacher = "Dr. XYZ";
- You can write and use a constructor:
  - function Course(n, t) { // best placed in <head>
        this.number = n;   // keyword "this" is required, not optional
        this.teacher = t;
    }
  - var course = new Course("CI211", "Dr. XYZ");

Creating objects using **new Object()**

```
var person = new Object();

// Assign fields to object "person"
person.firstName = “Bharat";
person.lastName = “Gupta";

// Assign a method to object "person"
person.sayHi = function() {
  alert("Hi! " + this.firstName + " " + this.lastName);
}

person.sayHi();  // Call the method in "person"
```

Creating objects using Literal Notation

```
var person = {
  // Declare fields ;Note: Use comma to separate fields
  firstName : “Bharat”, lastName : “Gupta",

  // Assign a method to object "person"
  sayHi : function() {
    alert("Hi! " + this.firstName + " " +
          this.lastName);
  }
}

person.sayHi();  // Call the method in "person"
```

Creating objects using Literal Notation

```
var triangle = {
  // Declare fields (each as an object of two fields)
  p1 : { x : 0, y : 3 },
  p2 : { x : 1, y : 4 },
  p3 : { x : 2, y : 5 }
}

alert(triangle.p1.y);   // Shows 3
```

Object Constructor and prototyping

```
function Person(fname, lname) {
  // Define and initialize fields
  this.firstName = fname;
  this.lastName = lname;

  // Define a method
  this.sayHi = function() {
    alert("Hi! " + this.firstName + " " +
          this.lastName);
  }
}

var p1 = new Person(“Bharat", “Gupta");
var p2 = new Person(“ABC", “XYZ");

p1.sayHi();  // Shows "Hi! Bharat Gupta "
p2.sayHi();  // Shows "Hi! ABC  XYZ"
```

Adding methods to objects using **prototype**

```
// Suppose we have defined the constructor  “Person”

var p1 = new Person("John", "Doe");
var p2 = new Person("Jane", "Dow");

// Attaching a new method to all instances of Person
Person.prototype.sayHello = function() {
  alert("Hello! " + this.firstName + " " + this.lastName);
}

// We can also introduce new fields via "prototype"

p1.sayHello();  // Show "Hello! John Doe"
p2.sayHello();  // Show "Hello! Jane Dow"
```

Array literals

- JavaScript has array literals, written with brackets and commas
  - Example: color = ["red", "yellow", "green", "blue"];
  - Arrays are zero-based: color[0] is "red"
- If you put two commas in a row, the array has an "empty" element in that location
  - Example: color = ["red", , , "green", "blue"];
    - color has 5 elements
  - However, a single comma at the end is ignored
    - Example: color = ["red", , , "green", "blue",]; still has 5 elements

Four ways to create an array

- You can use an array literal:
  var colors = ["red", "green", "blue"];
- You can use new Array() to create an empty array:
  - var colors = new Array();
  - You can add elements to the array later:
    colors[0] = "red"; colors[2] = "blue"; colors[1]="green";
- You can use new Array(***n***) with a single numeric argument to create an array of that size
  - var colors = new Array(3);
- You can use new Array(…) with *two or more* arguments to create an array containing those values:
  - var colors = new Array("red","green", "blue");

The length of an array

- If myArray is an array, its length is given by myArray.length
- Array length can be changed by assignment beyond the current length
  - Example: var myArray = new Array(5); **myArray[10]** = 3;
- Arrays are sparse, that is, space is only allocated for elements that have been assigned a value
  - Example: myArray[50000] = 3; is perfectly OK
  - But indices must be between 0 and 2^{32}-1
- As in C and Java, there are no two-dimensional arrays; but you can have an array of arrays: myArray[5][3]

Arrays and objects

- Arrays are objects
- car = { myCar: "Saturn", 7: "Mazda" }
  - car[7] is the same as car.7
  - car.myCar is the same as car["myCar"]
- If you know the name of a property, you can use dot notation: car.myCar
- If you don’t know the name of a property, but you have it in a variable (or can compute it), you must use array notation: car["my" + "Car"]

Array functions

- If myArray is an array,
  - myArray.sort() sorts the array alphabetically
  - myArray.reverse() reverses the array elements
  - myArray.push(…) adds any number of new elements to the end of the array, and increases the array's length
  - myArray.pop() removes and returns the last element of the array, and decrements the array's length
  - myArray.toString() returns a string containing the values of the array elements, separated by commas

Null & Undefined

- An undefined value is represented by the keyword "**undefined**".
  - It represents the value of an **uninitialized variable**

- The keyword "**null**" is used to represent "**nothing**"
  - Declare and define a variable as "null" if you want the variable to hold nothing.
  - **Avoid leaving a variable undefined**.

Type Conversion (To Boolean)

- The following values are treated as false
  - null
  - undefined
  - +0, -0, NaN (**N**ot **a N**umber)
  - "" (empty string)

Type Conversion

- Converting a value to a number

  ```
  var numberVar = someVariable  - 0;
  ```

- Converting a value to a string

  ```
  var stringVar = someVariable + "";
  ```

- Converting a value to a boolean

  ```
  var boolVar = !!someVariable;
  ```

Operators

- Arithmetic operators
  - +, -, *, /, %
- Post/pre increment/decrement
  - ++, --
- Comparison operators
  - ==, !=, >, >=, <, <=
  - **===, !==** (Strictly equals and strictly not equals)
    - i.e., **Type and value of operand must match/must not match**

== vs ===

```
// Type conversion is performed before comparison
var v1 = ("5" == 5);    // true

// No implicit type conversion.
// True if only if both types and values are equal
var v2 = ("5" === 5);  // false

var v3 = (5 === 5); // true

var v4 = (true == 1); // true (true is converted to 1)

var v5 = (true == 2); // false (true is converted to 1)

var v6 = (true == "1") // true
```

Logical Operators

- **!** – Logical NOT
- **&&** – Logical AND
- || – Logical OR

```
var tmp1 = null && 1000;  // tmp1 is null

var tmp2 = 1000 && 500;   // tmp2 is 500

var tmp3 = false || 500;        // tmp3 is 500

var tmp4 = "" || null;          // tmp4 is null

var tmp5 = 1000 || false;       // tmp5 is 1000


// If foo is null, undefined, false, zero, NaN, or an empty string, then set foo to 100.
foo = foo || 100;
```

Operators (continue)

- String concatenation operator
  - `+`
  - If one of the operand is a string, the other operand is automatically converted to its equivalent string value.

- Assignment operators
  - `=, +=, -=, *=, /=, %=`

- Bitwise operators
  - `&, |, ^, >>, <<`

document.write vs document.writeln

```
<body>

<p>Note that write() does NOT add a new line after each statement:</p>
<pre>
<script>
document.write("color:mediumblue">"Hello World!");
document.write("color:mediumblue">"Have a nice day!");
</script>
</pre>

<p>Note that writeln() add a new line after each statement:</p>
<pre>
<script>
document.writeln("color:mediumblue">"Hello World!");
document.writeln("color:mediumblue">"Have a nice day!");
</script>
</pre>
</body>
```

The with statement

- **with (object) statement**
- uses the **object** as the default prefix for variables in the **statement**
- For example, the following are equivalent:
  - **with** (document.myForm) {
    result.value = compute(myInput.value) ;
    }
  - document.myForm.result.value =
    compute(document.myForm.myInput.value);

“for/in” statement

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

- To iterate through all the properties in "object".
- "**variable**" takes the name of each property in "object"
- Can be used to iterate all the elements in an Array object.

```
var obj = new Object();  // Creating an object

// Adding three properties to obj
obj.prop1 = 123;
obj.prop2 = "456";
obj["prop3"] = true;    // same as obj.prop3 = true

var keys = "", values = "";
for (var p in obj) {
  keys += p + " ";
  values += obj[p] + " ";
}

alert(keys);
// Show "prop1 prop2 pro3 "

alert(values);
// Show "123 456 true "
```

Functions (Return Values)

```
// A function can return value of any type using the keyword "return".
// The same function can possibly return values  of different types
function foo (p1) {
        if (typeof(p1) == "number")
                return 0;           // Return a number
        else
        if (typeof(p1) == "string")
                return "zero"; // Return a string

        // If no value being explicitly returned
        // "undefined" is returned.
}

foo(1);             // returns 0
foo("abc");                 // returns "zero"
foo();              // returns undefined
```

Variable Arguments

```
// "arguments" is a local variable (an array) available  in every function
// You can either access the arguments through parameters  or through the
//"arguments" array.
function sum ()
{
  var s = 0;
  for (var i = 0; i < arguments.length; i++)
            s += arguments[i];
  return s;
}



sum(1, 2, 3);                 // returns 6
sum(1, 2, 3, 4, 5);           // returns 15
sum(1, 2, "3", 4, 5);         // returns ?
```

Built-In Functions

- **eval(expr)**
  - evaluates an expression or statement
    - eval("3 + 4"); // Returns 7 (Number)
    - eval("alert('Hello')"); // Calls the function alert('Hello')

- **isFinite(x)**
  - Determines if a number is finite

- **isNaN(x)**
  - Determines whether a value is **"Not a Number"**

Built-In Functions

- **parseInt(s)**
  - **Converts string** literals **to integers**
  - Parses up to any character that is not part of a valid integer
    - parseInt("3 chances") // returns 3
    - parseInt(" 5 alive") // returns 5
    - parseInt("How are you") // returns NaN
- **parseFloat(s)**
  - Finds a floating-point value at the beginning of a string.
    - parseFloat("3e-1 xyz") // returns 0.3
    - parseFloat("13.5 abc") // returns 13.5

Functions

```
<SCRIPT TYPE = "text/javascript">
        var input1 = window.prompt( "Enter first number", "0" );
        var input2 = window.prompt( "Enter second number", "0" );
        var input3 = window.prompt( "Enter third number", "0" );
        var value1 = parseFloat( input1 );
        var value2 = parseFloat( input2 );
        var value3 = parseFloat( input3 );
        var maxValue = maximum( value1, value2, value3 );
        document.writeln( "First number: " + value1 +  "<BR>Second number:
" + value2          +    "<BR>Third number: " + value3 +
         "<BR>Maximum is: " + maxValue );

        // maximum method definition (called from above)
        function maximum( x, y, z ) {
                return Math.max( x, Math.max( y, z ) );
    }
</SCRIPT>
```

Random Numbers

```
<SCRIPT TYPE="text/javascript">
        var value;
        document.writeln( "<H1>Random Numbers</H1>" +
          "<TABLE BORDER = '1' WIDTH = '50%'><TR>" );

        for ( var i = 1; i <= 20; i++ ) {
          value = Math.floor( 1 + Math.random() * 6 );

          document.writeln( "<TD>" + value + "</TD>" );
          if ( i % 5 == 0 && i != 20 )
            document.writeln( "</TR><TR>" );
        }
        document.writeln( "</TR></TABLE>" );
</SCRIPT>
```

Adding an item to the beginning of an array

```
var properties = ['red', '14px', 'Arial'];
    properties.unshift('bold');
```

array *properties contains four elements:*

['bold', 'red','14px', 'Arial']

Method	Original array	Example code	Resulting array	Explanation
.length property	`var p = [0,1,2,3]`	`p[p.length]=4`	[0,1,2,3,4]	Adds one value to the end of an array.
push()	`var p = [0,1,2,3]`	`p.push(4,5,6)`	[0,1,2,3,4,5,6]	Adds one or more items to the end of an array.

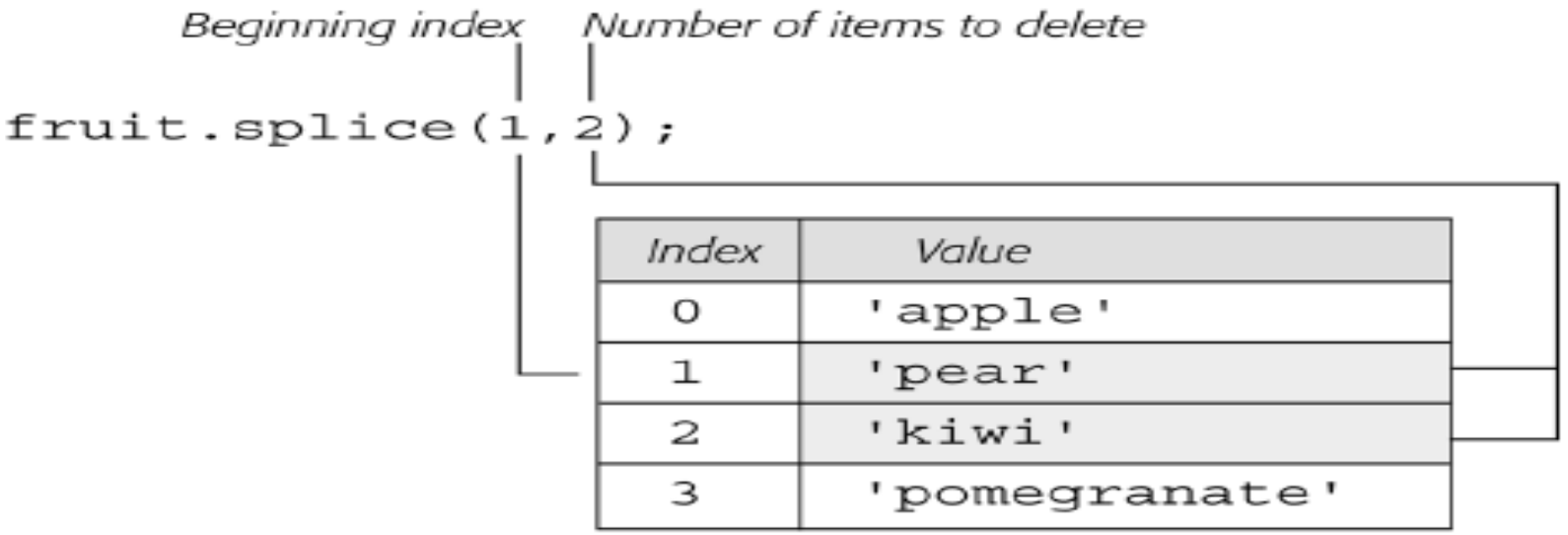
Method	Original array	Example code	Resulting array	Explanation
unshift()	`var p = [0,1,2,3]`	`p.unshift(4,5)`	[4,5,0,1,2,3]	Adds one or more item to the beginning of an array.

Method	Original array	Example code	Resulting array	Explanation
pop()	`var p = [0,1,2,3]`	p.pop()	`[0,1,2]`	Removes the last item from the array.
shift()	`var p = [0,1,2,3]`	p.shift()	`[1,2,3]`	Removes the first item from the array.

Adding and Deleting with *splice( )*

- **splice( ):** *lets you add items to an array* and delete items from an array.

- **var fruit=['apple','pear','kiwi','pomegranate'];**

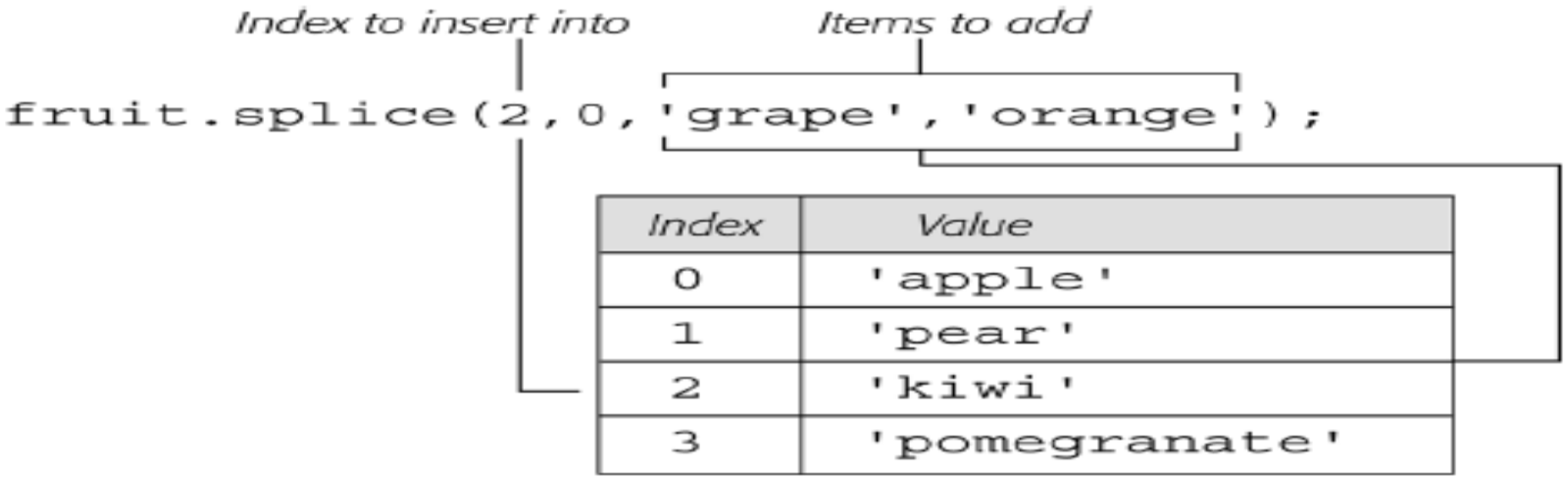


Results

Index	*Value*
0	'apple'
1	'pomegranate'

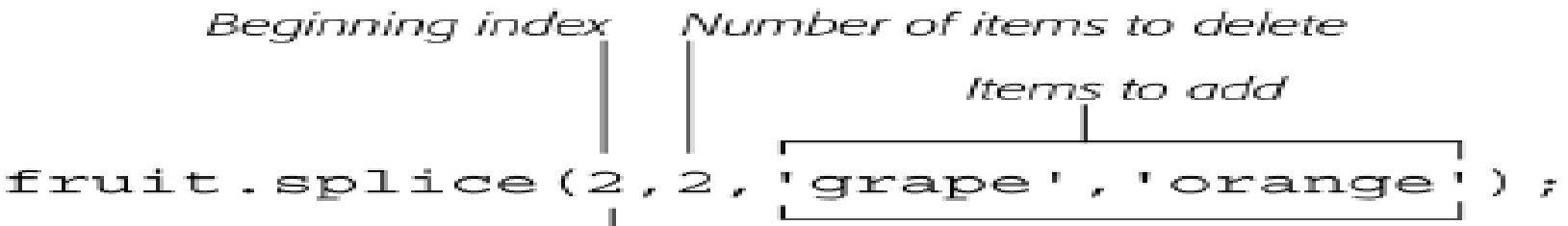
Adding items with splice( )

- var fruit=['apple','pear','kiwi','pomegranate'];
- fruit.splice(2,0,'grape','orange');
- **0 to indicate that you don't want to delete any items**



Replacing items with splice( )

- fruit.splice(2,2,'grape','orange');



Index	Value
0	'apple'
1	'pear'
2	'kiwi'
3	'pomegranate'

Results

Index	Value
0	'apple'
1	'pear'
2	'grape'
3	'orange'

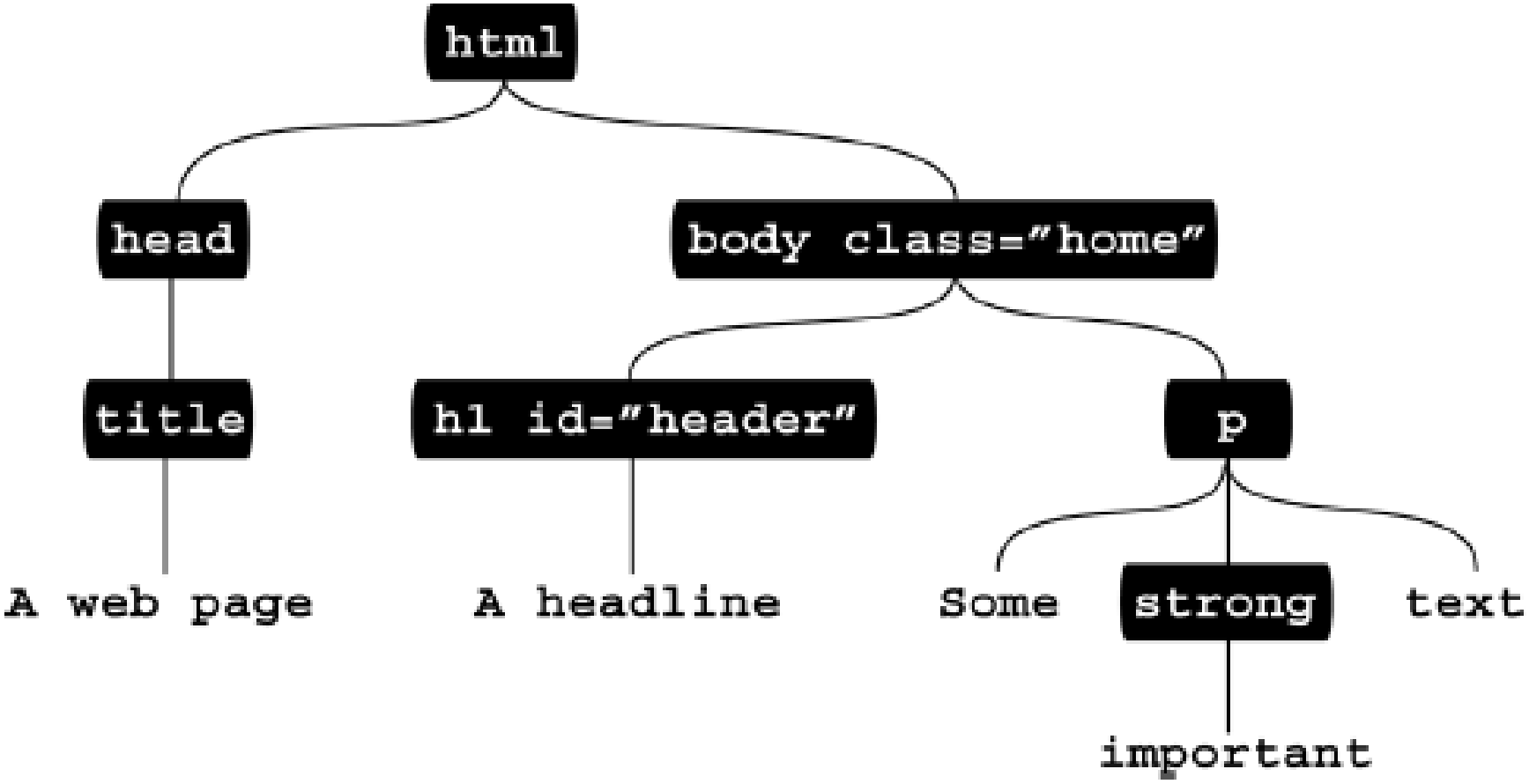
Dynamically Modifying Web Pages

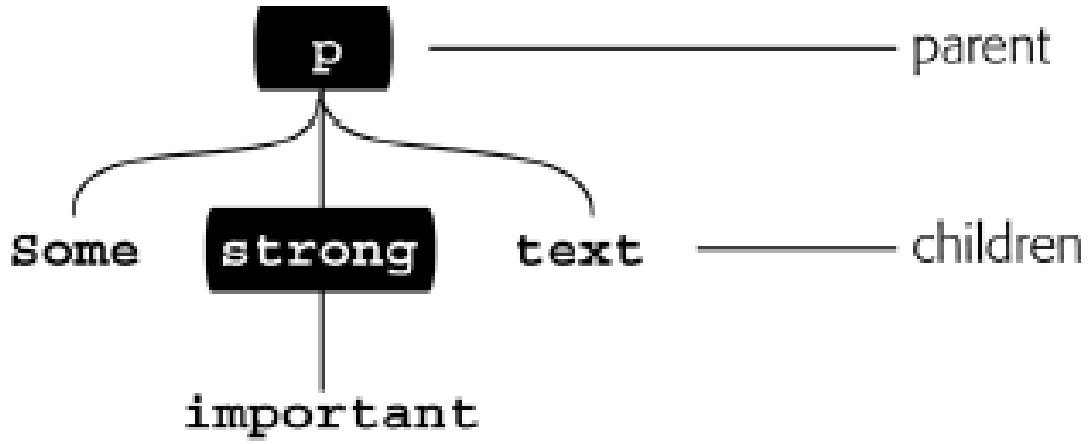
- **Document Object Model (DOM):** Representation of the page that the Web browser remembers the HTML tags, their attributes, and the order in which they appear in the file.
- The DOM provides the information needed for JavaScript to communicate with the elements on the Web page

simple Web page

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>A web page</title>
</head>
      <body class="home">
      <h1 id="header">A headline</h1>
      <p>Some <strong>important</strong> text</p>
</body>
</html>
```

tree


html
head
body class="home"
title
h1 id="header"
p
A web page
A headline
Some
strong
text
important


p
parent
Some
strong
text
children
important

Selecting a Page Element

- Two main methods for selecting nodes:
  - *getElementById( ) and*
  - *getElementsByTagName( ).*

getElementById( )

- Getting an element by ID means locating a single node that has a unique ID applied to it.
- document.getElementById('header')
- *var lookFor = 'header';*
- *var foundNode = document.getElementById(lookFor);*

getElementsByTagName( )

- Returns a list of nodes, instead of just a single node.
- Example: Search this document for every <a> tag and store the results in a variable named *pageLinks.*
- var pageLinks = document.getElementsByTagName('a');

- You can also use *getElementById( ) and getElementsByTagName( ) together.*
- var banner = document.getElementById('banner');
- var bannerLinks = banner.getElementsByTagName('a');
- var totalBannerLinks = bannerLinks.length;

Selecting nearby nodes

The DOM provides several methods of accessing nearby nodes

- **.childNodes**: It contains a list of all nodes that are direct children of that node.
- **.parentNode***:*It represents the direct parent of a particular node
- **.nextSibling and .previousSibling:** *properties that point to the node that* comes directly after the current node, or the node that comes before.
  - var headline = document.getElementById('header');
  - var headlineSibling = headline.nextSibling; //<p>

Events

- An event occurs as a result of some activity
  - e.g.:
    - A user clicks on a link in a page
    - Page finished loaded
    - Mouse cursor enter an area
    - A preset amount of time elapses
    - A form is being submitted

Event Handlers

- Event Handler – a segment of codes (usually a function) to be executed when an event occurs
- We can specify event handlers as attributes in the HTML tags.
- The attribute names typically take the form "**onXXX**" where **XXX** is the event name.
  - e.g.:

Other Website