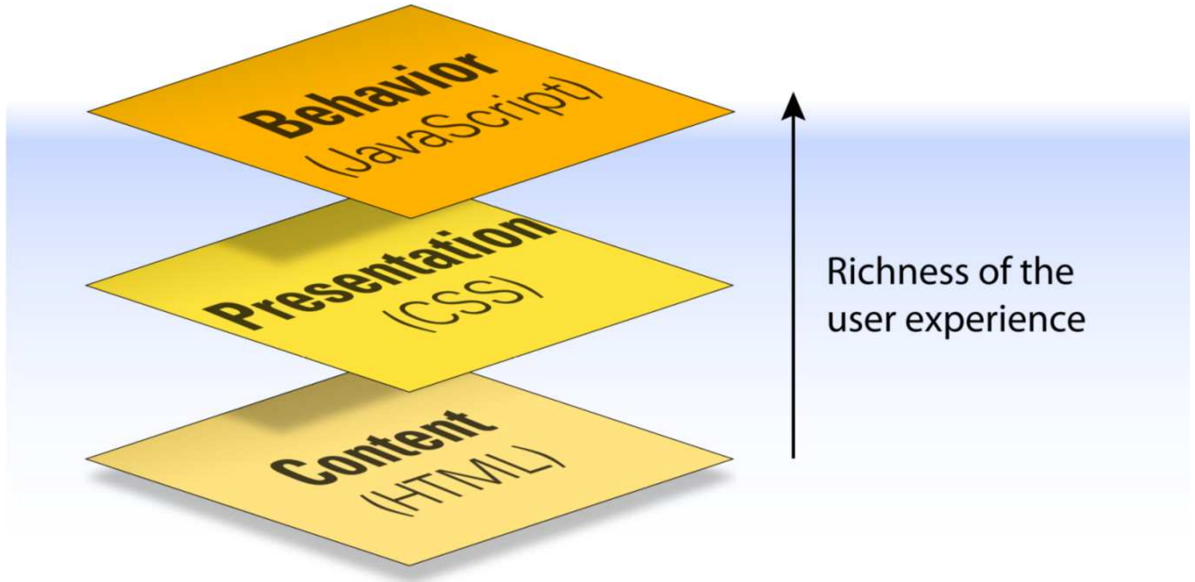
MIS6070

Web Based Information Systems



Lesson 5

HTML, CSS and JavaScript

- When building a site, we work through these layers from the bottom up:
  - Producing the **content** in **HTML5** format.
    - This is the base layer, which any visitor using any kind of browser should be able to view.
  - Making the site look better, by adding a layer of **presentation** information using **CSS**.
    - The site look good to users able to display CSS styles.
  - Use **JavaScript** to introduce an added layer of interactivity and dynamic **behaviour,** which will make the site easier to use in browsers equipped with JavaScript.

What can JavaScript Do?

- JavaScript is a programming language that provides action in applications.
- **Interactivity** enables an end user to take an action in an application, usually by clicking a button or pressing a key.
- A **dynamic application** adjusts and responds to such actions by end users.
- JavaScript also expands the opportunities to animate content.
- An **ordinary JavaScript program** is a sequence of statements. Statements are separated by semi-colons

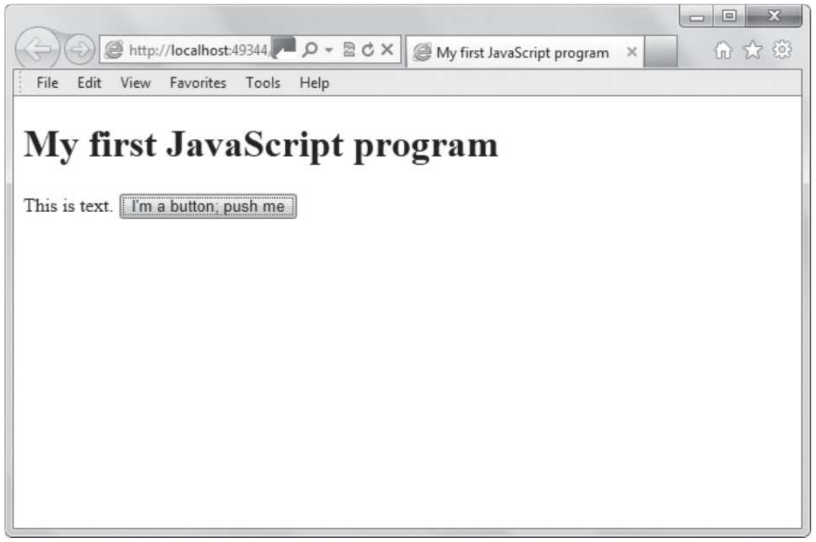
Pros and Cons of JavaScript

- Pros:
  - Allows more dynamic HTML pages, even complete web applications
- Cons:
  - Requires a JavaScript-enabled browser
  - Requires a client who trusts the server enough to run the code the server provides
- JavaScript has some protection in place but can still cause security problems for clients
- JavaScript was first used in 1555
  - web browsers have changed a lot since then

To create a simple JavaScript program

To create a simple JavaScript program, perform the following steps:

```
<!doctype html>
<html>
<head>
<title>My first JavaScript program</title>
</head>
<body>
<h1>My first JavaScript program</h1>
<p>This is text.
<button type = 'button' onclick = "alert('You clicked the button');"> I'm a button; click me</button>
</body>
</html>
```


http://localhost:49344/
My first JavaScript program
File Edit View Favorites Tools Help

My first JavaScript program

This is text. I'm a button; push me

Locating and Accessing Elements

- One important way to access display elements is with the getElementById() method.
- This method returns a reference to the first object with the specified id or NAME attribute.
- If you see something in your browser, JavaScript can “get at it” and put it under programmatic control.
- You can use the getElementById() method to access display elements.

Locating and Accessing Elements

- One of the most important things to do when learning a new language is to master basic input and output
- Document object
  - Represents content of a browser's window
- Create new Web page text with the write() method or the writeln() method of the Document object
  - Both methods require a text string as an argument
  - Text string or literal string
    - Text contained within double or single quotation marks
- writeln() method adds a line break after the line of text

write() and writeln() Methods

```
<html>
<head>
<tittle>JavaScript<tittle>
</head>
<body>
    <script type='text/javascript'>
    document.write('Hello World!');
</script>
</body>
</html>
```

JavaScript

```
<html>
    <head>
        <title>JavaScriptExample</title>
    </head>
    <body bgcolor="pink">
        <center>
            <h1>JavaScript</h1>
    <script type="text/javascript" language="javascript">
    document.write("Creating a Script in an HTML Document”);
    document.writeln("Creating my first");
    document.write(“Make-Up Class, Lesson 5”);
    document.write(“Wake-Up Class, Lesson 5a”);
    </script>
</body>
</html>
```

Functions

- A **function** is a segment of a program defined and performed in isolation from other parts.
- JavaScript programmers sometimes identify functions that return no value as **subroutines.**
- The *expression* of a function—the “function example1() {. . .}” part—doesn’t perform any of the code within the function.
- What you see in the source code is only the *definition* of a function.
- When the function is invoked, executed, or launched, something useful happen.

Function Example

```
<!doctype html>
<html>
<head>
<title>First use of a function</title>
<script type = "text/javascript">
function example1() {
    alert("This is the first alert.");
    alert("This is the second alert.");
}
</script>
</head>
<body>
<h1>First use of a function</h1>
<p>This is text.
<button type = 'button' onclick = "example1();">I'm a button;
click me</button>
</p>
</body>
</html>
```

JavaScript Outputs

- In JavaScript we can create three kinds of popup boxes: Alert box, Confirm box, and Prompt box.
- **Alert boxes** are commonly used to test the operation of JavaScript programs.
  - An alert box can help you ensure information is displayed to the user.
  - A user has to click OK to close an alert box.
    - Alert boxes will stop your scripts from running until the user clicks the OK button.

```
<body>
        <script type='text/javascript'>
        alert('Hello World!');
        </script>
</body>
```

JavaScript

- An ordinary JavaScript program is a sequence of statements.
- Statements are separated by semi-colons.

```
alert('This is the first alert');
alert('This is the second alert');
```

- **Where Do You Place Scripts?**
- Scripts can be in the either ***<head>*** section or <*body*> section
- Convention is to place it in the <*head*> section <html>

```
<head>
<script type="text/javascript">
....
</script>
</head>
```

JavaScript

- **In-Line JavaScript**
- To define a JavaScript block in your web page, simply use the following block of HTML.

```
<script type='text/javascript‘language="javascript">
// Your script goes here.
</script>
```

- Some older browsers do not recognize JavaScript.
- These browsers would sometimes display JavaScript code in the page as if it were part of the contents of the page.
- Therefore, it is conventional to place JavaScript code between comment tags as follows:

```
<script>
<!--
..JavaScript code goes here..
//-->
</script>
```

JavaScript

- **In-Line JavaScript Example**

```
<body bgcolor="pink">
<center> <h1>JavaScript</h1>
<script type="text/javascript" language="javascript">
document.write("Creating a Script in an HTML Document");
</script>
</body>
```

JavaScript

- **Scripts can be provided locally or remotely accessible JavaScript file using src attribute**

```
<body>
<script src="myScript.js"> </script>
<body>
```

When using different folders

```
<script language="JavaScript" type="text/javascript"
src="myOwnSubdirectory/myOwn2ndJavaScript.js">
</script>
```

Example

Create a js file myScript with the content

```
alert('Hello World!');
document.write("Using an External Script File to Add a Script");
```

Create the following html file

```
<html>
<head>
  <title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
  <center><h1>JavaScript</h1>
  <script src="myScript.js"> </script>
  </center>
</body>
</html>
```

Adding Comments to a JavaScript Program

- Comments
  - Nonprinting lines placed in code containing various types of remarks
- Line comment
  - Hides a single line of code
  - Add two slashes // before the comment text
- Block comments
  - Hide multiple lines of code
  - Add /* before the first character included in the block and */ after the last character in the block

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. **Lesson 5, Example 1 and 2**

```
<script language="javascript">
function show_alert()
{
alert("Hi! This is alert box!!")
}
</script>
```

JavaScript Popup Boxes

- **Confirm Box**
  - A confirm box is often used if you want the user to verify or accept something.
  - When a 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 the user clicks "Cancel", the box returns false. **Lesson 5, Example 3**

```
<script language="javascript">
function disp_confirm()
{
var r=confirm("Press a button")
if (r==true)
{
document.write("You pressed
OK!")
}
else
{
document.write("You pressed
Cancel!")
}
}
</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. **Lesson 5, Example 4**

```
<script language=
“JavaScript”>
<!—
var yourname =
prompt('Type your name
here');
document.writeln("<center
>Welcome, "+
yourname+"</center>");
//-->
</script>
```

Declaring Variables

- Declaring Multiple variables
  - Var variable_name;
- Declaring Multiple variables
  - Var price, quantity;
- To assign value to a variable
  - Variable_name=value; e.g. price=20;
- To assign value to multiple variables
  - Var price=5, quantity=5;

Variables, Date Types and Operators

- A variable is a "container" for information you want to store.
- A variable's value can change during he script. You can refer to a variable by name to see its value or to change its value.
- Rules for variable names:
  - Variable names are case sensitive
  - They must begin with a letter or the underscore character

Variables, Date Types and Operators

- JavaScript is a loosely typed language. The only key word for declaring a JavaScript variable is **var.**
- The actual data type used depends on the initial value you assign to the variable. Consider the following examples:
  - var AccountNumber
    - var AccountNumber = 1111
  - var LastName = “Akinyi”

Variables, Date Types and Operators

- String variables must be enclosed in "double quotes".
- Capital letters and lower case are distinct, and a SPACE is just another character.
  - You can have a space in a string, but not in a variable name - for example, you could not call a variable my name.
- To avoid this problem, whilst still making the script meaningful, is to make variable names out of words, with no spaces, but with capital letters to mark word starts. For example
  - hisName
  - basicIncomeTax
  - priceOfFish

```
var myName;
myName = "Walter Masiga";
```

Variables

```
<HTML>
        <HEAD>
                <SCRIPT Language = JavaScript>
                        var day = prompt("Enter the day of the week", "Day");
                </SCRIPT>
        </HEAD>
        <BODY>
                <SCRIPT Language = "JavaScript">
                        document.write("<H2> The day is " + day  + "</H2>");
                </SCRIPT>
</BODY>
</HTML>
```

- Prompt method picks up the string day from the user which is assigned to the variable day
- The JavaScript code document.write() embedded in the body tags writes the content of the variable day to the browser

Variables

```
<HTML>
            <HEAD>
                        <SCRIPT Language = JavaScript>
                                    var day = prompt("Enter the day of the week", "Day");
                        </SCRIPT>
            </HEAD>
            <BODY>
                        <SCRIPT Language = "JavaScript">
                                    document.write("<H2> The day is " + day  + "</H2>");
                        </SCRIPT>
<script>
if(day=="Saturday")
{
document.write("It is a weekend");
alert("It's a weekend");
}
</SCRIPT>
</BODY>
</HTML>
```

Variables

- The action of assigning an initial value to a variable is called *initialization*. You give the variable a value using the *assignment operator*—the equal sign:
  - var *variableName* = *initialValue*.
- You only need to use the var key word when you create the variable.
- When you want to refer to the variable, you only use its name.
- Assign a value to a variable (after it has been declared) in the following fashion:
  - *variableName* = *anyValue*

Variables, Date Types and Operators

- As the following piece of code demonstrates, you can also per form calculations when you assign a value:
  - var answer
  - answer = 4 * 2 + 5
  - document.write(answer)

JavaScript Operators

The Arithmetic Operators:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -5
*	Multiply both operands	A * B will give 200
/	Divide numerator by denumerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 5

JavaScript Operators

- **The Comparison Operators:**

Operator	Description
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

JavaScript Operators

The Assignment Operators:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

Simple arithmetic

- To calculate a total on the basis of multiplying (* means multiply) the price and the quantity.
- This involves three variables, which would all have had to be declared with var.
  - var price;
  - var quantity;
  - var total;
- Then the user must be asked to input values for the price and the quantity.
  - price = prompt("Enter the price:","5.00");
  - quantity = prompt("How many: ","1");

Simple arithmetic

- The computer must calculate the total.
- Finally the total must be displayed.
  - total = price * quantity;
  - alert("The total is "+total);
- **Lesson 5, Example 5, 6 and 7**
- We want to add on VAT to this, at 17.5%, so we need to work out the total by
- Modify example 7 to achieve this.

Control Flow Statements

- The scripts written in JavaScript are sequentially executed in a top-down manner,
  - To change the sequence in which the statements are executed you use the control flow statements.
- In JavaScript control statements are classified as:
  - **Selection statements**-allows the execution of a group of statements from multiple groups of statements
  - **Loops**-allows repeated execution of a group of statements
  - **Jump statements**-allows the execution to skip or jump over certain statements

Selection (Making decisions)

- Conditional statements enable you to test for various conditions and take action based on the results of the test. Specifically, conditional statements execute when the tested condition proves to be true.
  - Conditional statements include the if, if...else, and switch statements.
- The if statement checks whether a logical condition is true; if it is, it then executes one or more statements.
- The basic syntax of the if statements is shown here:

```
if (condition)
Statement
```

Lesson 5, Example 8

Selection (Making decisions)

- In some situations we need the computer to make a decision on the basis of some data.
- For example, a customer may get a cheaper price if they purchase a larger quantity of items.
- In this case we need the computer to make a decision on the basis of the quantity. In this situation you must use the reserved word **if**.
  - Using the if statement

```
If(condition)
{
Statement1
}
```

For example

```
<script type="text/javascript" language="javascript">
var Number=45;
if((Number%2) != 0)
{
document.write(Number + " is an odd number");
}
document.write("<br/>Thank you!");
</script>
```

Selection (Making decisions)

- **If…else statement**
- **If** allows you to execute a set of statements only when particular condition is true.
- If you want to execute a set of statements when the condition is false, then use if…else statement.

```
If
{
Statement1
}
Else
{
Statement2
}
```

For example

```
<script type="text/javascript" language="javascript">
var Number=44;
if((Number%2) != 0)
{
document.write(Number + " is an odd number");
}
else
{
document.write(Number + " is an even number");
}
document.write("<br/>Thank you!");
</script>
```

Selection (Making decisions)

- The switch statement provides a simpler alternative to the nested if…else statements.
- The switch statement evaluates a series of conditional tests or cases and executes additional statements based on whether the case proves true.
- The syntax of the switch statement is shown here:
- switch (expression) {

```
        case label1:
                statements;
        break;
                .
        case label:
                statements;
        break;
        default:
                statements;
        }
```

For example

```
<script type="text/javascript" language="javascript">
var letter="I";
switch(letter)
{
default: document.write("consonant");
break;
case "A":
document.write("vowel A");
break;
case "E":
document.write("vowel E");
break;
case "I":
document.write("vowel I");
break;
case "O":
document.write("vowel O");
break; case "U":
document.write("vowel U");
break;
}
document.write("<br/>Using switch statement!");
</script>
```

Making decisions

- The switch statement compares the result of the expression against the label for each case.
- The statements for the first case that proves true are executed.
- If no case proves true, the statements associated with the default statement are executed if the optional default statement was included.
- **Lesson 5, Example 13**

Working with Loops

- A loop is a series of statements that executes repeatedly, allowing you to perform iterative operations within your script.
- JavaScript statements that support looping logic include:
  - for,
  - while,
  - do...while.
- The nice thing about loops is that they enable you to write just a few line of code and then to execute them repeatedly, making your scripts easier to write and maintain.

Loops or iteration

- ***The while Statement***
- Use the while loop when you want to check the condition at the start of the loop.
- The while statement executes a loop as long as a condition is true.
- The syntax of the while statement is shown here:

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

For example

```
<script type="text/javascript“ language="javascript">
var totalDistance=0, fare=0;
while(totalDistance<=5)
{
fare=fare+5; totalDistance=totalDistance+2;
document.write("fare: Shillings."+fare+"<br/>");
}
document.write("Thank You!");
</script>
```

Lesson 5, Example 21

The group statements inside the while loop are executed (fare+5, totaldistance+2, six times when totaldistance=12 when the condition becomes false.

Loops or iteration

- ***The do…while Statement***
- The do…while statement executes a loop until a condition becomes false.
  - The group of statements is executed at least once unlike the while loop
- The syntax of the do…while statement is outlined below:

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

For example

```
<script type="text/javascript" language="javascript"> var
totalDistance=5, fare=0;
do
{
fare=fare+5; totalDistance=totalDistance+2;
document.write("fare: Shillings."+fare+"<br/>");
}
while(totalDistance<=5);
document.write("Thank You!");
</script>
```

Lesson 5, Example 22

The group statements inside the do…while loop are executed after which the condition is evaluated. Note that in the first iteration condition becomes false.

Loops or iteration

- ***The do…while Statement***
- The difference between the do…while loop and the while loop is that the do…while loop always executes at least once.
- This is because the condition is checked at the end of the first execution of the loop instead of at the beginning.

Loops or iteration

- ***The for Statement***
- The for statement executes until a condition becomes false and uses a variable to control the number of times the loop executes.
  - Executes a block of statements for a number of **predetermined times**
- The for loop is comprised of three parts: **a starting expression**, **a test condition**, and **an increment statement**.
- The syntax of the for statement is shown here:

```
for (initialization_statement; condition; updation_statement)
{
statements;
}
```

For example

```
<script type="text/javascript" language="javascript">
var Number;
document.write("Even numbers from 0 to 5 <br/>");
for(Number=0;Number<=5;Number=Number+2)
{
document.write(Number+"<br/>");
}
document.write("Thank You!");
</script>
```

Lesson 5, Example 23

Loops or iteration

- *The for Statement*
- Example 14, Lesson 5 Scripts
  - Shows how the for statement can be used to set up a loop that iterates five times.
  - When the script begins to execute, the value of i is 0. Each time the loop executes, the value of i is incremented by 1
    - for (i=0; i<5; i++)

Working with Jump statements

- The jump statements allow you to jump over or skip certain statements in a script and execute some other statements
- There are two jump statements-break and continue
- Using the break statement
  - Used with switch statement to prevent the execution of the subsequent case statements
  - It also allows you to break or exit a loop
    - When used inside a loop, the break statement stops executing the loop and causes the loop to be exited immediately

Working with Jump statements

- If the value of the count is less that 5, the while loop is executed.
- The if…else statement specified inside the while loop checks if the count variable is divisible by 2 or not.
- If count is divisible by 2, then the break statement is executed.
- As a result, the while loop is immediately exited and the statement immediately following the while loop is executed.

For example

```
<script type="text/javascript" language="javascript">
var count=0;
while(count<5)
{
++count;
if((count%2==0))
break;
else document.write("count="+count+"<br/>");
}
document.write("while loop exited"); </script>
```

Lesson 5, Example 25

Working with Jump statements

- Continue statement
  - The condition if…else statement checks whether the count variable is divisible by 2 or not.
  - If count is divisible by 2, then the continue statement is executed.
  - As a result, the execution of the loop is halted, the remaining statements are skipped, and the condition of the while loop is evaluated for the next iteration.

```
<script type="text/javascript" language="javascript">
var count=0; while(count<5)
{
++count;
if(count%2==0)
continue;
else
document.write("count="+count+"<br/>");
}
document.write("while loop exited");
</script>
```

Lesson 5, Example 26

Working with functions

- A function refers to a cohesive block of statements that has a name and can be accessed or called from anywhere and any number of times in the script.
  - The statement in which the function is called is known as the calling statement.

Function function_name(parameter1, parameter2, …parameterN)

{

Statement1

}

```
<script type="text/javascript" language="javascript">
function calculateAverage()
{ var a=15, b=25; c=35;
var average= (a+b+c)/3;
document.write("Average = "+average);
}
document.write("The calculateAverage() function is to be called...<br/>");
calculateAverage();
</script>
```

  - A function can zero or more number of parameters

Lesson 5, Example 26

  - A function is called by specifying its name and an empty pair of parentheses

Working with functions

- A function can also return some information or value to the calling statement.
- You can use a variable in the calling statement to store the value returned by the function.
- To return a value to the calling statement, you need to use the return statement.

  return value;

  - This allows you to return a single value to the calling statement.
  - Though return statement is specified at the end of the function just before the closing curly brace (}) you can specify the return anywhere in the function

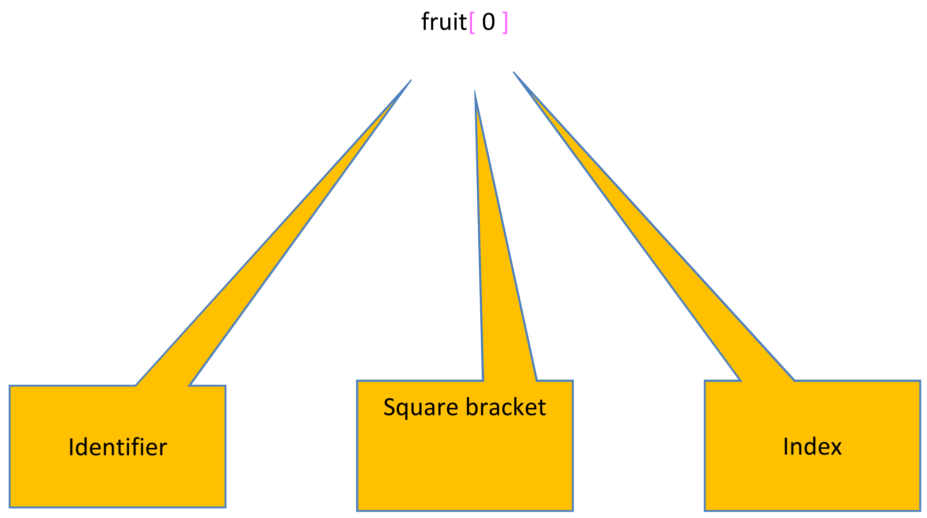
```
<script type="text/javascript" language="javascript">
function calculateAverage(a, b, c)
{
var average= (a+b+c)/3;
return average;
}
document.write("The calculateAverage() function is to be called...<br/>");
var avg=calculateAverage(5,20,30);
document.write("Average is "+avg);
</script>
```

Using JavaScript Arrays

- An array is similar to a normal variable, in that you can use it to hold any type of data.
  - However, it has one important difference
- The difference between such a normal variable and an array is that an array can hold *more than one* item of data at the same time.
- Each place where a piece of data can be stored in an array is called an *element*.
- An **array** is a named collection of different values
  - The individual values are represented by their respective array elements, each of which has a unique index.
    - The index is a whole number that indicates the position of the array elements with respect to one another.
- Var array_name=new array(array_length)
  - var bookprices=new array(3);

Using JavaScript Arrays

- An indexed list of elements
- fruit[0], fruit[1], fruit[2], and fruit[3] are the elements of an array
- ‘fruit’ is the identifier for that array
- The length of the ‘fruit’ array is 4, i.e. ‘fruit’ has four elements



Using JavaScript Arrays

- var student1, student2, student3, student4 ;
- student1 = “Akinyi” ;
- student2 = “Kumar” ;
- student3 = “Maina” ;
- student4 = “Naisho” ;
- document.write(student1) ;
- document.write(student2) ;
- document.write(student3) ;
- document.write(student4) ;

```
student = new Array( 4 )  ;   //array declaration

student[ 0 ] = “Akinyi” ;
student[ 1 ] = “Kumar” ;
student[ 2 ] = “Maina” ;
student[ 3 ] = “Naisho” ;

for ( x = 0 ; x < 4 ; x = x + 1 ) {
        document.write( student[ x ] ) ;
}
```

Using JavaScript Arrays

- var student1, student2, student3, student4 ;
- student1 = “Akinyi” ;
- student2 = “Kumar” ;
- student3 = “Maina” ;
- student4 = “Naisho” ;
- document.write(student1) ;
- document.write(student2) ;
- document.write(student3) ;
- document.write(student4) ;

Sorts the elements in alphabetical order

```
student = new Array( 4 )  ;  //array declaration
student[ 0 ] = “Akinyi” ;
student[ 1 ] = “Kumar” ;
student[ 2 ] = “Maina” ;
student[ 3 ] = “Naisho” ;
x.sort( ) ;
for ( x = 0 ; x < 4 ; x = x + 1 ) {
        document.write( student[ x ] ) ;
}
```

Using JavaScript Arrays

- The **Array** object let's you store multiple values in a single variable.
- How do you distinguish between pieces of data in an array?
  - You give each piece of data an *index* value.
- To refer to that piece of data you enclose its index value in square brackets after the name of the array.
  - myArray[0] 25
  - myArray[1] 35

    - Notice that the index values start at 0 and not 1.

Using JavaScript Arrays

- To create a new array, you need to declare a variable name and tell JavaScript that you want it to be a new array using the new keyword and the Array() function.
- For example, the array myArray could be defined like this:
  - var myArray = new Array();

Using JavaScript Arrays

- You can say up front how many elements the array will hold if you want to, although this is not necessary.
- You do this by putting the number of elements you want to specify between the parentheses after Array.
- For example, to create an array that will hold six elements, you write the following:
- var myArray = new Array(6);

Using JavaScript Arrays

- An *array* is an indexed list of values that can be referred to as a unit.
- Arrays can contain any type of value.
- Before you can use an array, you must first declare it.
  - An array that will hold 6 strings that contain information about Private cars inventory

```
Auto = new Array(6);
Auto[0] = 58 Ford Escort;
Auto[1] = 57 Ford Cortina;
Auto[2] = 85 Suzuki Baleno;
Auto[3] = 56 Suzuki Cultus;
Auto[4] = 50 Honda Civic;
Auto[5] = 57 Honda CRV;
```

- The first statement uses the **new** keyword to create an Array object named Auto that will contain 6 entries.
- The rest of the statements assign string values to the array.
- As you can see, the array's index
- starts with 0 and goes to 5.

Using JavaScript Arrays

- For example, you could have an array called emp that contains employees' names indexed by their employee number.
- So emp[1] would be employee number one, emp[2] employee number two, and so on.
- **To create an Array object:**
    - arrayObjectName = new Array(element0, element1, ..., element N)
    - arrayObjectName = new Array(arrayLength)

Using JavaScript Arrays

- **Lesson 5 Script 15**, uses the an array to demonstrate how to display an array element by using the document.write() statement with the element's array name and index number.
- **Processing Arrays with for Loops**
  - A **for loop** can be used to traverse the entire array and display all of its contents as shown in **Lesson 5, Script 16**.
  - The loop begins at the beginning of the array, Auto[0], and uses an increment of 1 to step through the elements of the array until it reaches the end.
  - To determine the end of the array, the script creates a variable called **arrayLength** and sets its value equal to the array length property

Using JavaScript Arrays

- **Dense Arrays**
  - You can also create dense arrays.
    - A ***dense array*** is one that is populated at the time it is declared.
  - This is a very efficient technique for creating small arrays.
  - The following example shows you how to create a dense array named animals that consists of five entries.

animals = new Array("mice", "dog", "cat", “pig", "fish");

  - After creating the array, the script prints its contents using a for loop and the **Animals.length** property as shown in **Lesson 5 Script 17.**

Using JavaScript Arrays

- **Sorting Arrays**
  - Arrays have a sort() method you can use to sort their contents.
  - The **Lesson 5, Script 18** shows how to display a sorted list of the contents of an array.
  - The script first defines a dense array with five entries.
  - Next, it displays a heading, and it then uses the sort() method inside a document.write() statement to display the sorted list.
  - Notice that you do not have to create a for loop to step iteratively through the array's index.
  - The sort() method takes care of this.

Using JavaScript Arrays

- **Populating Arrays with Dynamic Data**
  - You can create scripts that enable users to supply them with their data, as opposed to hard coding it into the script.
  - **Lesson 5, Script 15,** prompts users to type their three favourite things and then creates an array to contain the list.
  - The script uses a for statement to control program execution.
  - Each time the loop executes, it asks the user to type a response and assigns the response to the array by giving it the next available array index number (as designated by the current value of i).

Using JavaScript Arrays

- **Multiple-dimension array**
- A multiple-dimension array can be used to represent data stored in table format.
- **Lesson 5, Script 20** demonstrates how to build and display the contents of a multidimensional array.
  - Comments have been used to divide this script into three parts, each of which performs a specific activity.
  - In Part 1, a variable called arraySize that contains the number of cars the user wants to enter is defined.
  - Then created an array called MyArray using the values contained in arraySize.

Using JavaScript Arrays

- **Lesson 5, Script 20** demonstrates how to build and display the contents of a multidimensional array.
  - In Part 2, a for loop that creates the second dimension of the array. In this case, the for loop starts at 0, incrementing by 1 until it has executed for each element in the array.
    - This step populates each column in the table as it iterates from top to bottom on a row-by-row basis. In each iteration, the user is asked to enter the type, color, and price of a car. In its first iteration, the script starts at row 0 (MyArray[0]) and populates each column in row 0 (MyArray[0][0], MyArray[0][1], MyArray[0][2]). On its next iteration, the value of row has been incremented by 1 to 1.
    - Again, the script prompts the user for information about the car and populates the cells in the second row (MyArray[1][0], MyArray[1][1], MyArray[1][2]). The process repeats as many times as the user has cars to enter.

Using JavaScript Arrays

- **Lesson 5, Script 20** demonstrates how to build and display the contents of a multidimensional array.
  - In Part 3, the script prints out the contents of the multidimensional array using nested for loops.
  - The first loop controls the row of the table that is processed. It starts at array element 0 and increments by 1 until the entire array has been processed.
  - The second loop prints the contents of the multidimensional array by executing repeatedly for each row in the table, starting at column 0 and finishing with column 2.

Understanding Events

- Event
  - Specific circumstance monitored by JavaScript
  - Script can respond to in some way
  - Allows users to interact with Web pages
- Common events: actions users perform
- Can also monitor events not resulting from user actions

Events

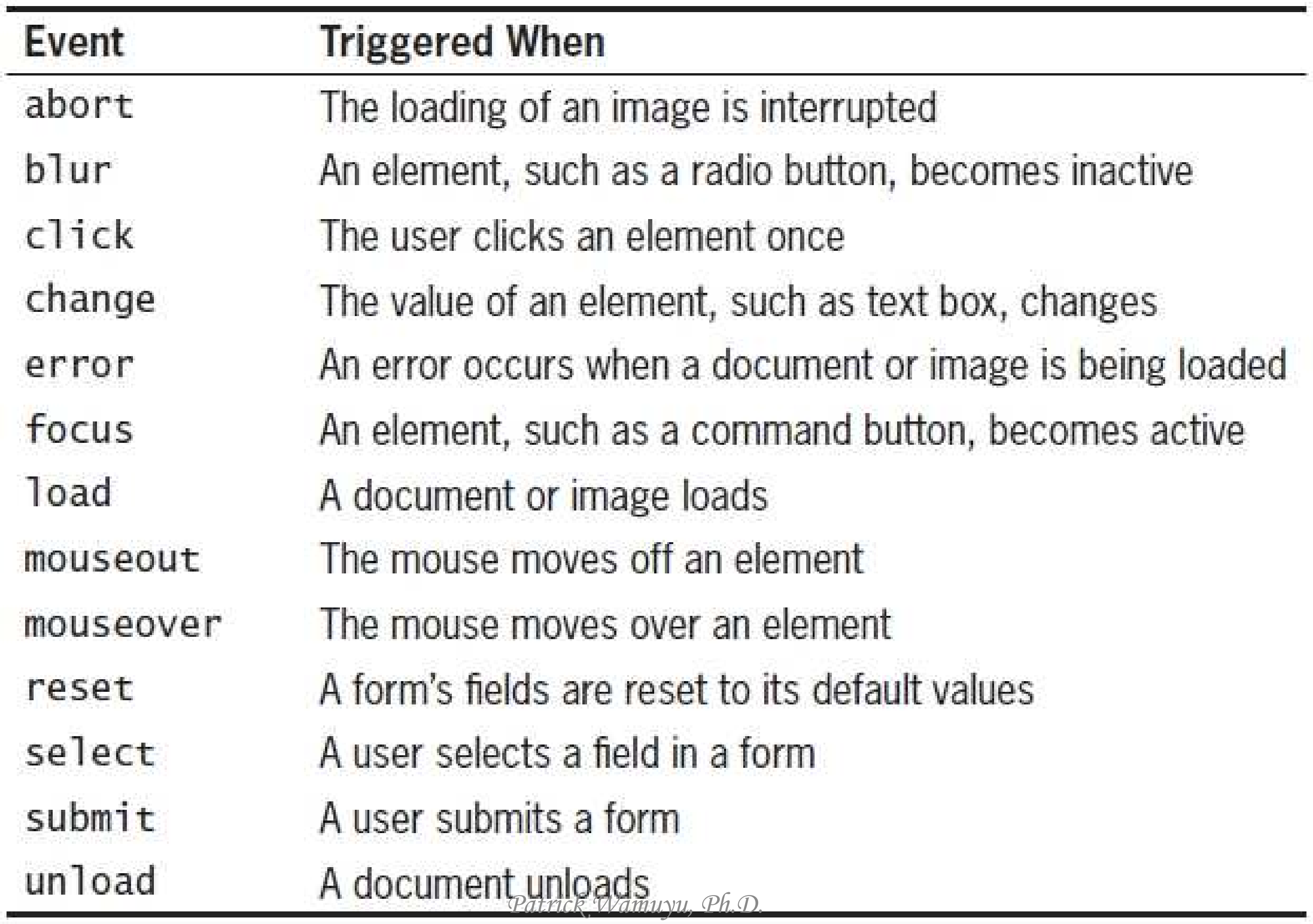
- Events are the actions that occur as a result of browser activities or user interactions with the web pages.
  - Such as the user performs an action (mouse click or enters data)
  - We can validate the data entered by a user in a web form
  - Communicate with Java applets and browser plug-ins

Event Categories

- Keyboard and mouse events
  - Capturing a mouse event is simple
- Load events
  - The page first appears on the screen: “Loads”, leaves: “Unloads”, …
- Form-related events
  - onFocus() refers to placing the cursor into the text input in the form.
- Others
  - Errors, window resizing.

Understanding Events

- JavaScript Events

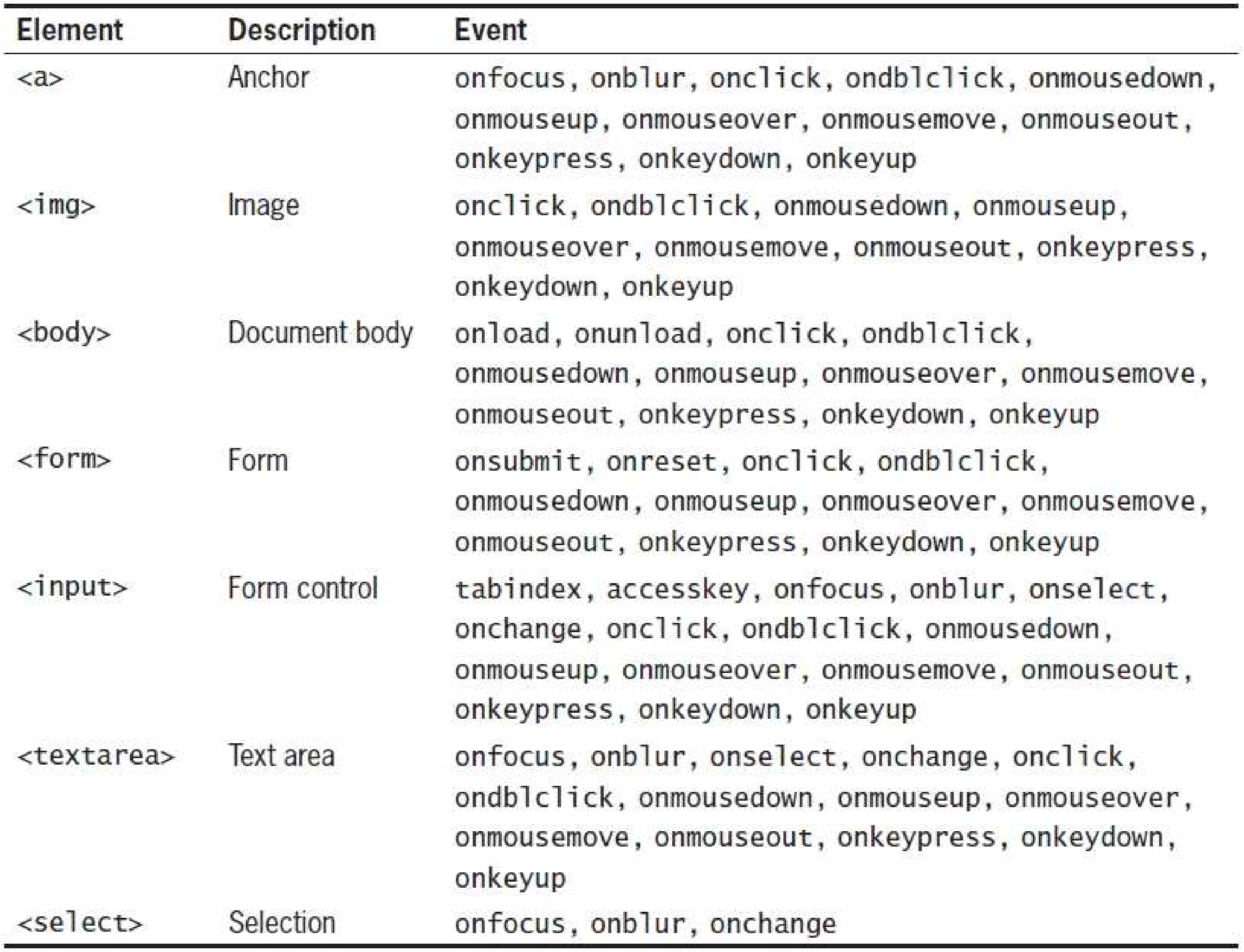


Event	Triggered When
`abort`	The loading of an image is interrupted
`blur`	An element, such as a radio button, becomes inactive
`click`	The user clicks an element once
`change`	The value of an element, such as text box, changes
`error`	An error occurs when a document or image is being loaded
`focus`	An element, such as a command button, becomes active
`load`	A document or image loads
`mouseout`	The mouse moves off an element
`mouseover`	The mouse moves over an element
`reset`	A form's fields are reset to its default values
`select`	A user selects a field in a form
`submit`	A user submits a form
`unload`	A document unloads

Understanding Events

- Working with elements and events
  - Events: associated with XHTML elements
  - Event handler
    - Code that executes in response to a specific event
  - JavaScript code for an event handler
    - Contained within the quotation marks following the name of the JavaScript event handler

Elements and their associated events



Element	Description	Event
`<a>`	Anchor	onfocus, onblur, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
`<img>`	Image	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
`<body>`	Document body	onload, onunload, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
`<form>`	Form	onsubmit, onreset, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
`<input>`	Form control	tabindex, accesskey, onfocus, onblur, onselect, onchange, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
`<textarea>`	Text area	onfocus, onblur, onselect, onchange, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
`<select>`	Selection	onfocus, onblur, onchange

Understanding Events

- Referencing Web page elements
  - Append element’s name to the name of any elements in which it is nested
    - Start with the `Document` object
  - Specific element properties
    - Appended to the element name
  - Allows for the retrieval of information about an element or the ability to change the values assigned to its attributes

Browser Objects

- **Handling Events**
  - To make your Web pages really interactive, you must add another tool to your scripting skill set: **event handlers.**
  - An *event handler* is a trap that recognizes the occurrence of a particular type of event.
  - You can add code to your scripts that alters the browser's default response to events.
    - For example, instead of automatically loading the URL specified by a link, you could display a confirmation dialog box that asks the user to agree to certain terms before proceeding.

Browser Objects

Handling Events

- The names of event handlers are based on the events that trigger them.
- Placing the word on in front of the event name creates the event handler's name.
  - For example the event handler for the click event is onClick.
- Each event is associated with a specific object. When an event occurs for a given object, its event handler executes (assuming that you have written one).
  - For example, the click event occurs whenever a user clicks on a button, document, check box, link, radio option, reset button, or submit button.

Browser Objects

Handling Events

- The names of event handlers are based on the events that trigger them.
- Placing the word on in front of the event name creates the event handler's name.
  - For example the event handler for the click event is onClick.
- Each event is associated with a specific object. When an event occurs for a given object, its event handler executes (assuming that you have written one).
  - For example, the click event occurs whenever a user clicks on a button, document, check box, link, radio option, reset button, or submit button.

Browser Objects

Handling Events

- Event handlers are surprisingly easy to define considering their power. You place them within HTML tags that define the object.
- For example, you can define an event to occur whenever a Web page is loaded by placing an onLoad event handler inside the first HTML <BODY> tag as shown here:

```
<BODY onLoad="window.alert('Web page loading: Complete.')">
```

Browser Objects

Handling Events

- In this example, an alert dialog appears when the page finishes loading.
- Notice that the event handler comes immediately after the tag and that its value is placed within quotation marks.
- You can use any JavaScript statement as the value for the event handler.
- You can even use multiple statements, provided that you separate them with semicolons.
- Alternatively, you can use a function call, which enables you to perform more complex actions in response to the event.

Browser Objects

The event Object

- The event object is populated on every occurrence of an event.
- The information in its properties can be referenced by event handlers, giving your script access to detailed information about the event.
- For example, the event.modifiers property specifies any modifier keys that were associated with a mouse or keyboard event.

Browser Objects

- **Types of Events**
  - JavaScript supports >24 different events and event handlers.
  - These events can be broadly divided into a few categories:
    - Window and frame events
    - Mouse events
    - Keyboard events
    - Error events

Browser Objects

- **Window and Frame Events**
  - Events that affect window and frame objects include the load, resize, unload, and move events. The event handlers for these events are onLoad, onResize, onUnload, and onMove.
  - These event handlers are placed inside the <BODY> tag.

Browser Objects

Mouse Events

- Mouse events execute whenever you do something with the mouse.
- The MouseOver event occurs when the pointer is moved over an object.
- The MouseOut event occurs when the pointer is moved off an object.
- The MouseDown event occurs when a mouse button is pressed.
- The MouseUp event occurs when a mouse button is released.
- The MouseMove event occurs whenever the mouse is moved.
- The Click event occurs whenever you single-click on an object.
- The DblClick event occurs whenever you double-click on an object.

Browser Objects

- **Mouse Events**
  - the use of the onClick and onDblClick event handlers
- **Keyboard Events**
  - Keyboard events are like mouse events in that they occur whenever the user presses or releases a keyboard key.
    - There are three k eyboard events: KeyDown, KeyUp, and KeyPress.

Browser Objects

Error Events

- An error event occurs whenever your JavaScripts run into an error. Error events automatically trigger the onerror event.
- By adding an onerror event handler to your scripts, you can intercept these errors and suppress them.
- After they are suppressed, you then can either attempt to fix them programmatically or display a customized error message.

Events defined by JavaScript

HTML elements	HTML tags	JavaScript defined events	Description
Link	<a>	click	Mouse is clicked on a link
		dblClick	Mouse is double-clicked on a link
		mouseDown	Mouse button is pressed
		mouseUp	Mouse button is released
		mouseOver	Mouse is moved over a link
Image	<img>	load	Image is loaded into a browser
		abort	Image loading is abandoned
		error	An error occurs during the image loading
Area	<area>	mouseOver	The mouse is moved over an image map area
		mouseOut	The mouse is moved from image map to outside
		dblClick	The mouse is double-clicked on an image map
Form	<form>	submit	The user submits a form
		Reset	The user refreshes a form
…	…	…	…

Event Handlers

- When an event occurs, a code segment is executed in response to a specific event is called "event handler".
- Event handler names are quite similar to the name of events they handle.
- E.g the event handler for the "click" event is "onClick".

- <HTMLtag eventhandler="JavaScript Code">

Event Handlers

Event Handlers	Triggered when
onChange	The value of the text field, textarea, or a drop down list is modified
onClick	A link, an image or a form element is clicked once
onDblClick	The element is double-clicked
onMouseDown	The user presses the mouse button
onLoad	A document or an image is loaded
onSubmit	A user submits a form
onReset	The form is reset
onUnLoad	The user closes a document or a frame
onResize	A form is resized by the user

onClick event Handler

```
<html>
<head>
    <TITLE> onClick & onDblClick Example</TITLE>
  </HEAD>
  <BODY>
    <FORM>
    <INPUT TYPE="button" VALUE="Click on me!" onClick="window.alert('You
single-clicked.')">
    <INPUT TYPE="button" VALUE="Double-Click on ME"
onDblClick="window.alert('You double-clicked!')">
    </FORM>
  </BODY>
</HTML>
```

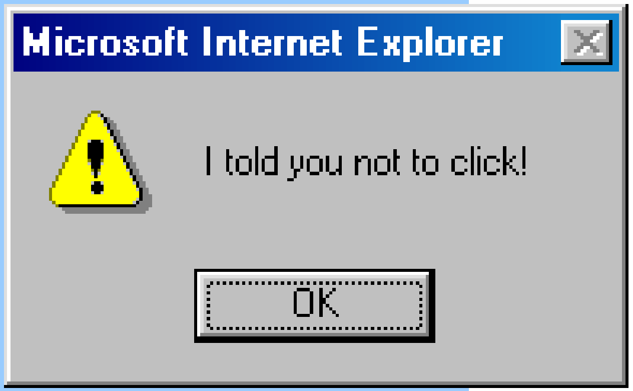
onLoad event Handler

```
<HTML>
 <HEAD>
  <TITLE>onLoad, onResize, onUnload & onMove Example</TITLE>
 </HEAD>
 <BODY onLoad="window.alert('Web page loading: Complete.')"
        onResize="window.alert('What is the matter with my current
size?')"
        onUnload="window.alert('Oh no, I am melting......')"
        onMove="window.alert('What's wrong with this spot?')">
 </BODY>
</HTML>
```

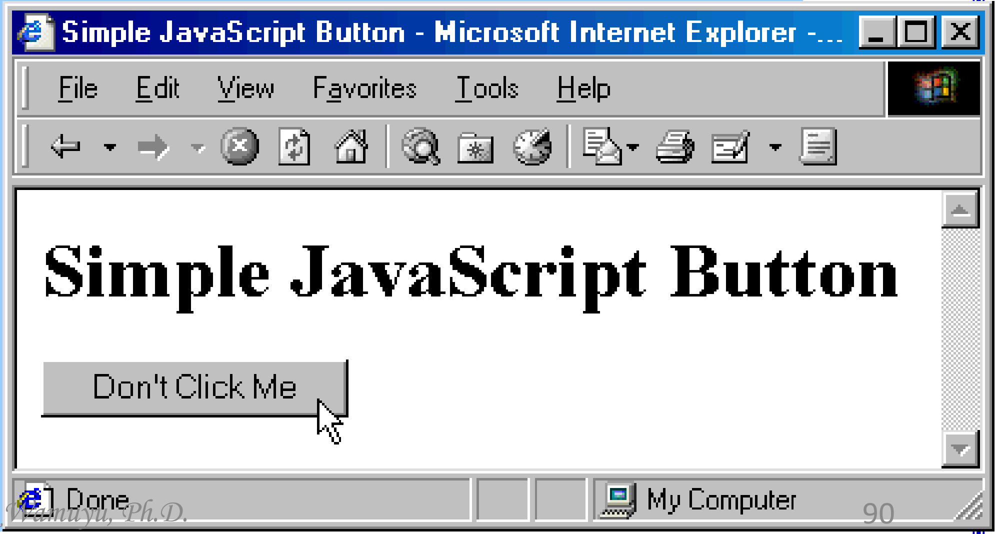
User Events, Form Example

```
<html><head>
  <title>Simple JavaScript Button</title>
<script language="JavaScript"><!--
function dontClick() {
  alert("I told you not to click!");
}
// --></script>
</head>

<body>
<h1>Simple JavaScript Button</h1>
<form>
  <input type="button"
        value="Don't Click Me"
        onClick="dontClick()">
</form>
</body></html>
```


Microsoft Internet Explorer

I told you not to click!


Simple JavaScript Button - Microsoft Internet Explorer -...

Simple JavaScript Button

onMouseOver and onMouseOut Event Handlers

```
<HTML>
  <HEAD>
    <TITLE>onMouseover, onMouseOut, onMouseDown & onMouseUp
     Example</TITLE>
  </HEAD>
  <BODY>
    <A HREF="http://www.microsoft.com"
      onMouseOver='document.bgColor="red"';
      onMouseOut='document.bgColor="white"';>
      onMouseDown and onMouseUp example</A><P>
<h4>This is onMouseOver and onMouseOut example</A><P></h4>
  </BODY>
</HTML>
```

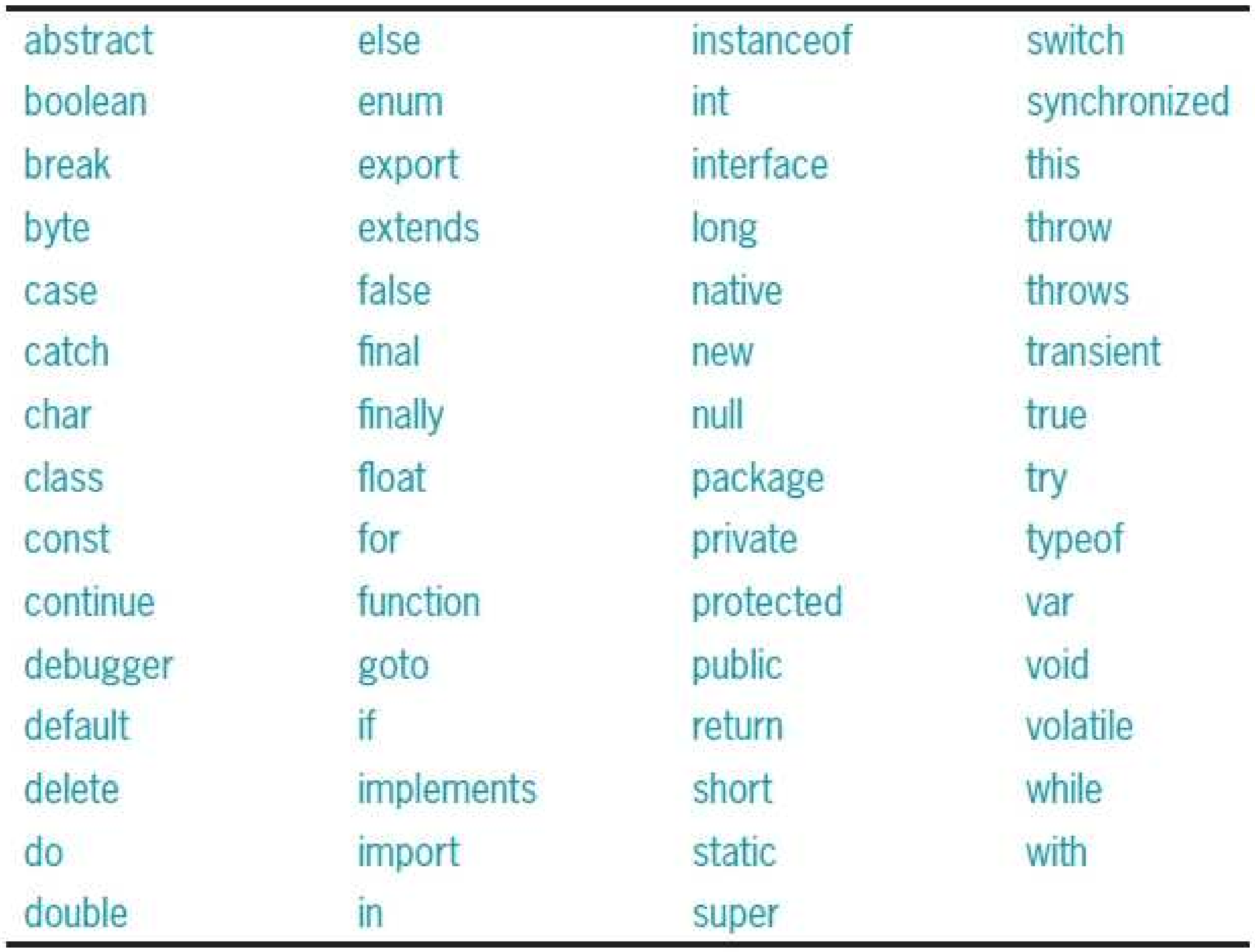
onKeyDown

```
<HTML>
  <HEAD>
    <TITLE>Age Verification Example</TITLE>
  </HEAD>
  <BODY>
    <A HREF="http://www.magazine.com"
     onClick="return(window.confirm('You must be older than 30 to
     visit this site. Are you?'))";> magazine.com</A>
  </BODY>
</HTML>
```

onError

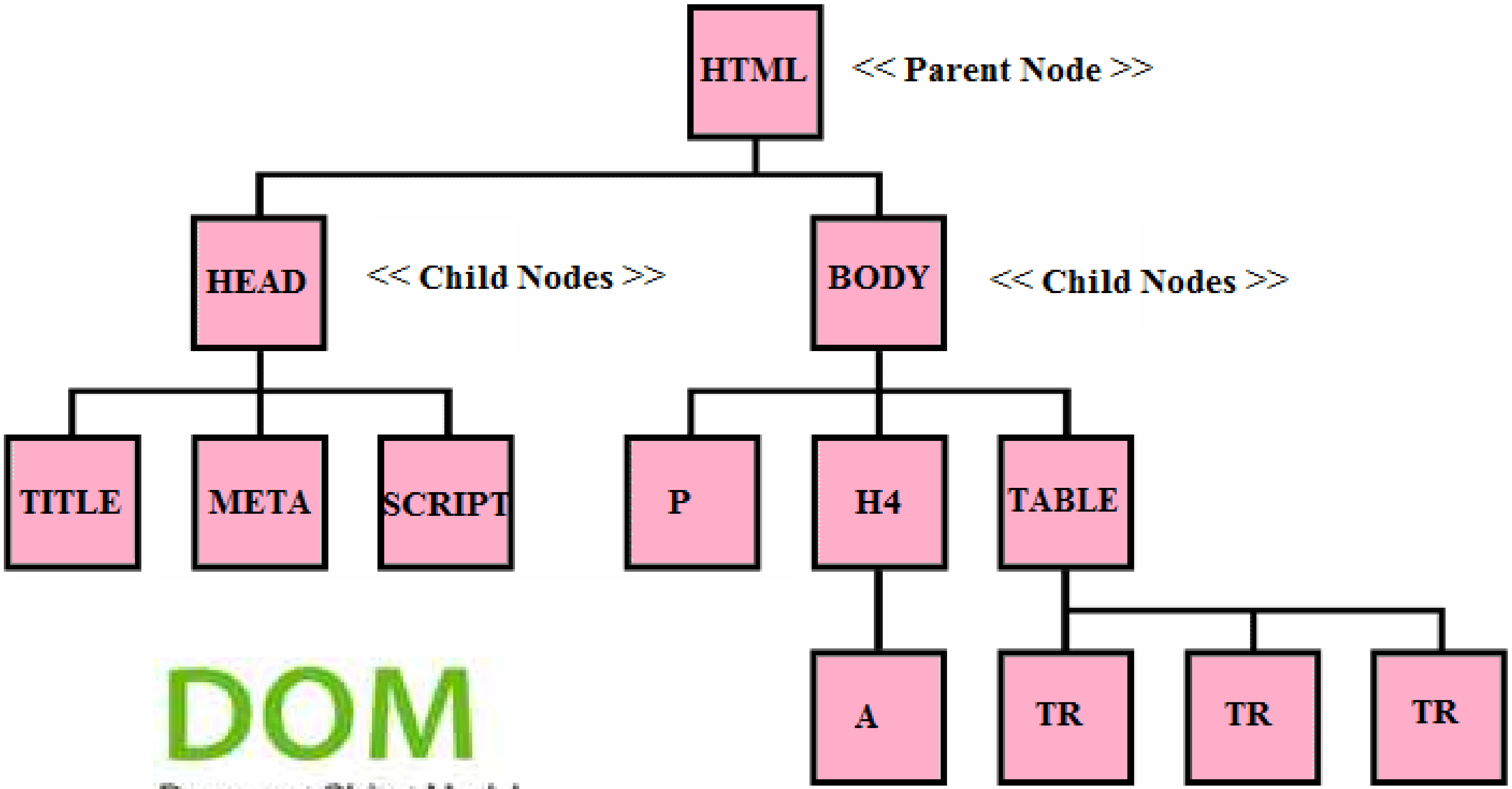
```
<HTML>
  <HEAD>
    <TITLE>onKeyDown Example</TITLE>
  </HEAD>
  <BODY>
    <FORM>
    <INPUT TYPE="text" VALUE="" onKeyDown="window.alert('You
     pressed: ' + String.fromCharCode(event.which))">
    </FORM>
  </BODY>
</HTML>
```

JavaScript reserved words



abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

Document Object Model (DOM)



Document Object Model (DOM)

- The Document Object Model gives you access to all the elements on a web page.
- With the DOM, XHTML elements can be treated as objects, and many attributes of XHTML elements can be treated as properties of those objects.
- The Document Object Model (DOM) is a model of programming that concerns the way in which we represent objects contained in a web document (single web page).
- There are several different levels of the DOM Standard, as proposed by the W3C

Document Object Model (DOM)

DOM is a representation of the current web page as a tree of HTML/XHTML/Javascript objects

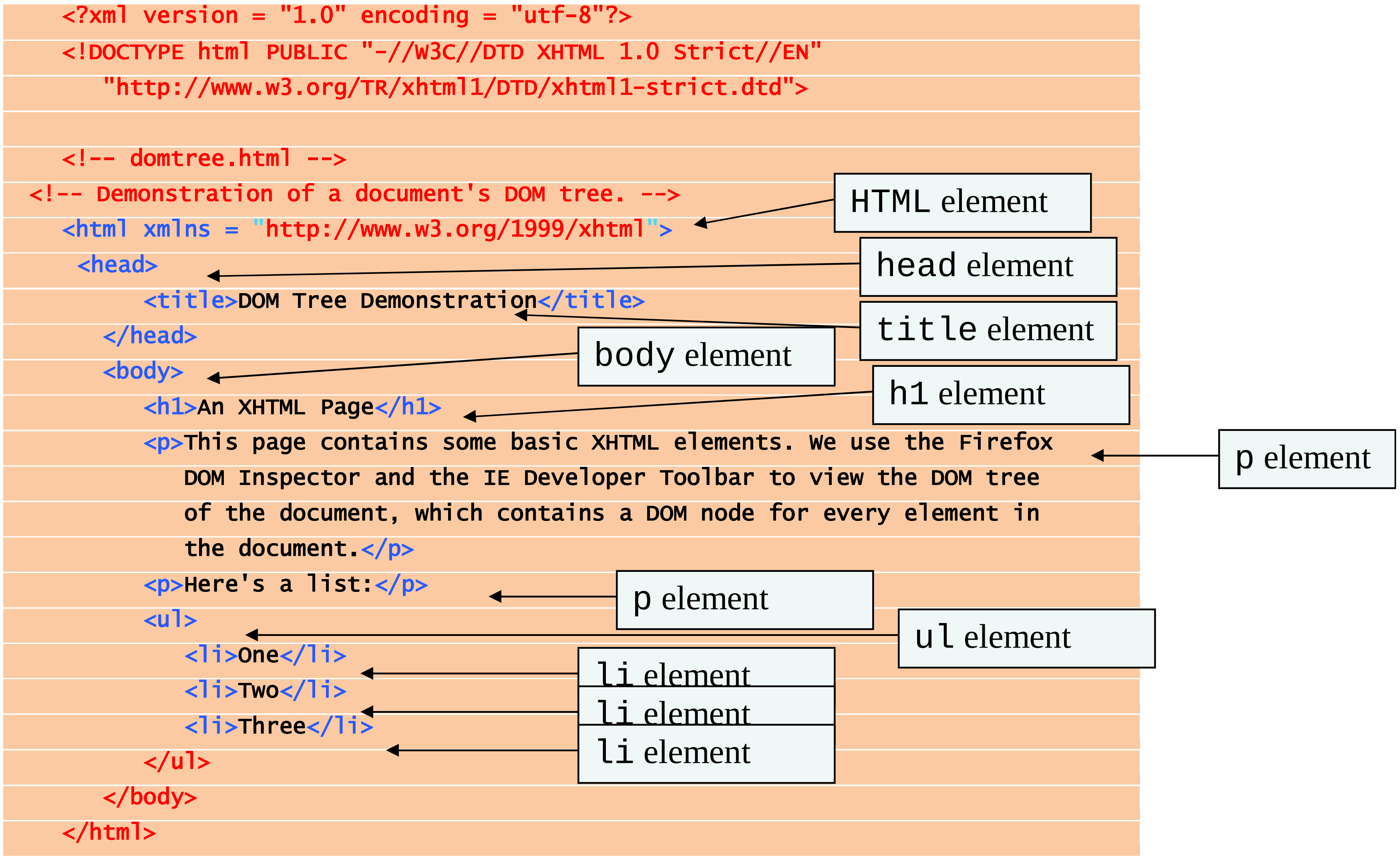
- Allows you to view/modify page elements in script code after page has loaded
- Client side = highly responsive interactions
- Browser-independent
- Allows progressive enhancement of pages

Linear vs. Hierarchical Views

- Level 0 of the DOM takes somewhat of a linear view of objects in a web document, forcing the programmer to “drill down” from the window object to relevant properties and methods.
- Level 1 and subsequent DOM Levels perceive objects using a hierarchical view.
- Instead of identifying specific HTML tag names, this view works through the idea of *nodes* in a tree diagram.

Linear vs. Hierarchical Views

- Each node has the potential to be a *parent*, *sibling*, or *child* node of other nodes in the document.
- This model may be a little trickier to program, at first, but saves us from having to know the names of all the elements in our document.
- All pages *must* be well-formed XHTML documents.
- All pages *must* include a valid DOCTYPE.
- You *must* include all text inside valid XHTML elements.



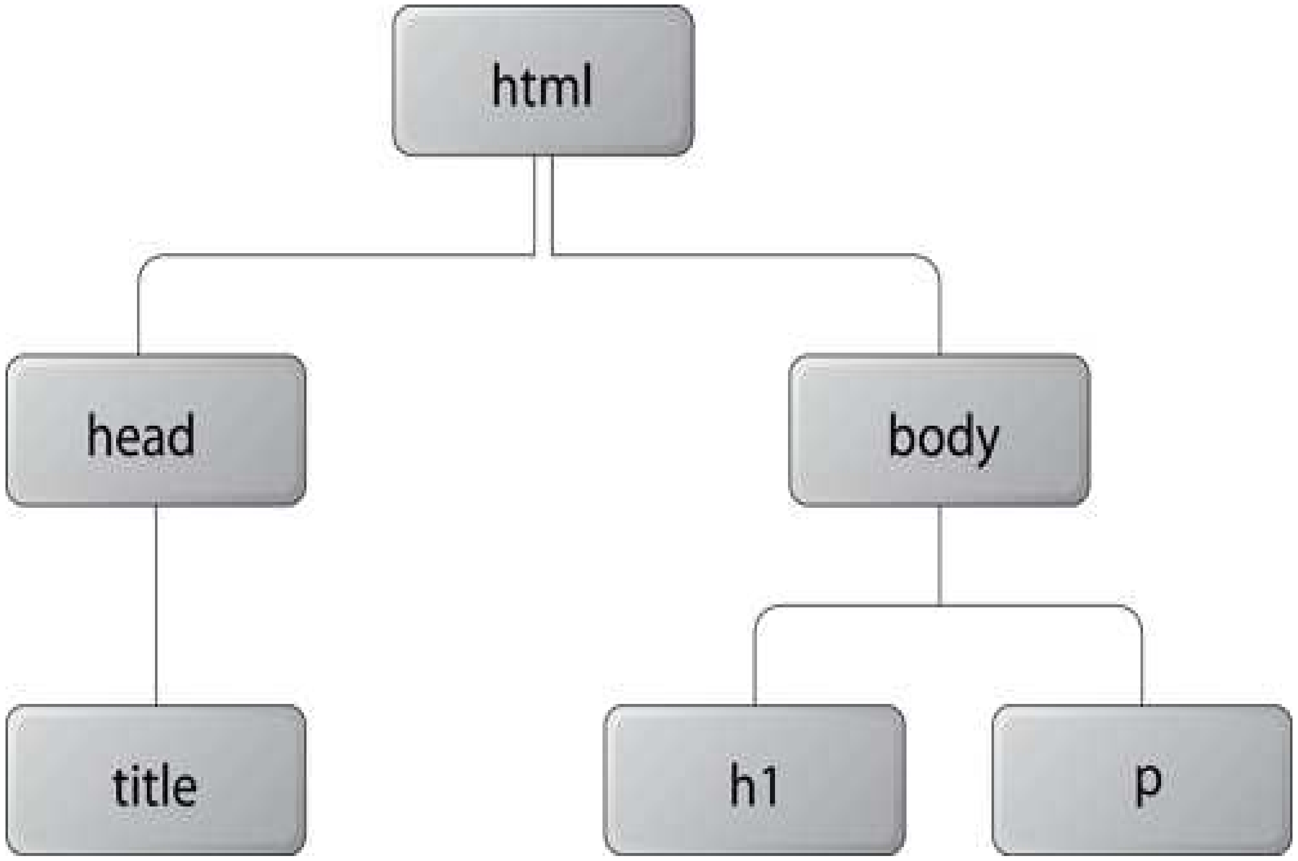
```
    <?xml version = "1.0" encoding = "utf-8"?>
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

    <!-- domtree.html -->
  <!-- Demonstration of a document's DOM tree. -->
    <html xmlns = "http://www.w3.org/1999/xhtml">
     <head>
          <title>DOM Tree Demonstration</title>
       </head>
       <body>
          <h1>An XHTML Page</h1>
          <p>This page contains some basic XHTML elements. We use the Firefox
             DOM Inspector and the IE Developer Toolbar to view the DOM tree
             of the document, which contains a DOM node for every element in
             the document.</p>
          <p>Here's a list:</p>
          <ul>
             <li>One</li>
             <li>Two</li>
             <li>Three</li>
          </ul>
       </body>
    </html>
```

An example XHTML page

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <title>Widgets</title>
  </head>
  <body>
    <h1>Widgets</h1>
    <p>Welcome to Widgets, the number one company
    in the world for selling widgets!</p>
  </body>
</html>
```

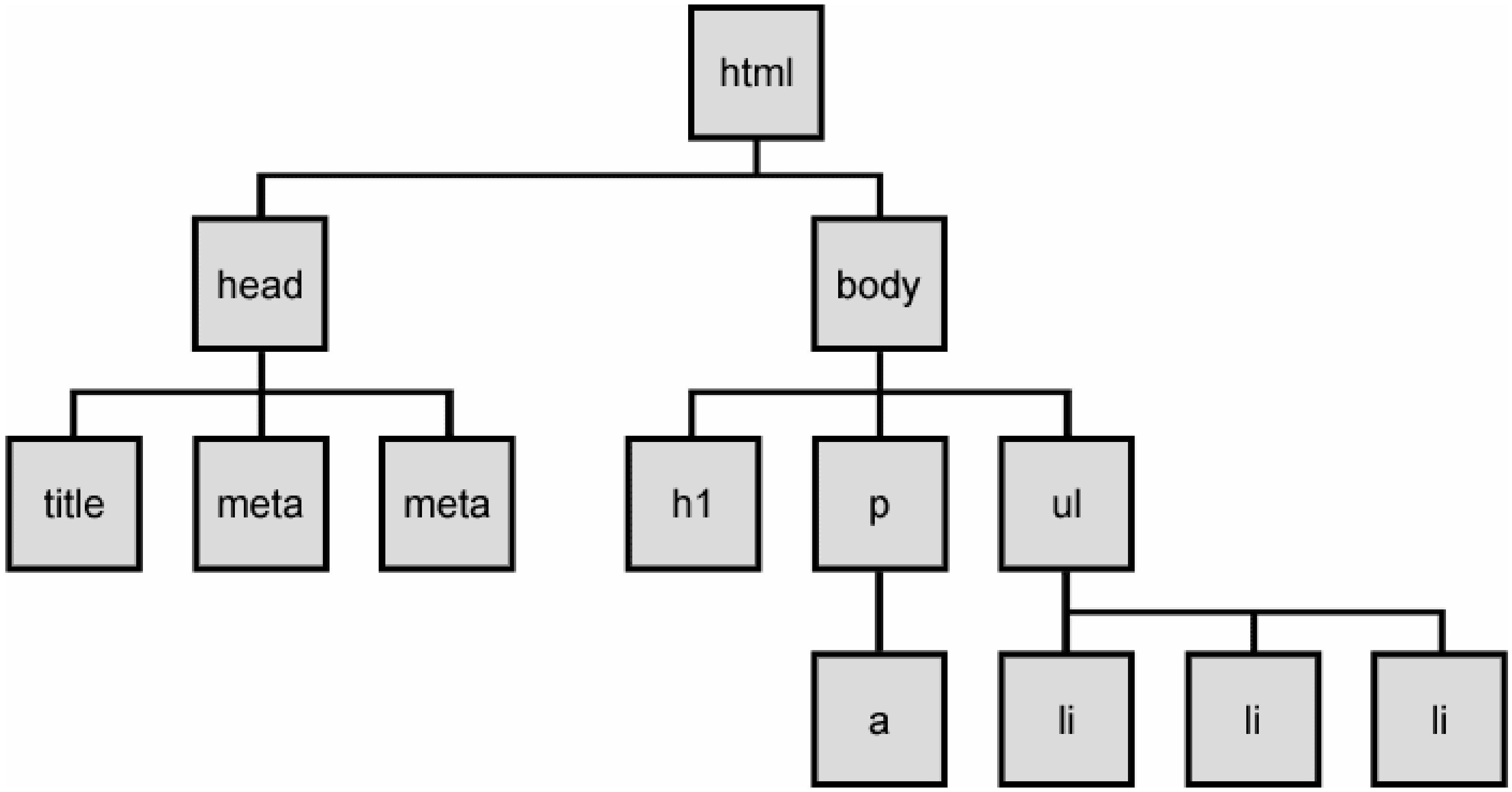
The resulting DOM tree



An example XHTML page

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/XHTML1/DTD/XHTML1-strict.dtd">
<html xmlns="http://www.w3.org/1555/XHTML" xml:lang="en" lang="en">
  <head>
              <title>Page Title</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<meta http-equiv="Content-Language" content="en-us" />
  </head>
  <body>
      <h1>This is a heading</h1>
       <p>A paragraph with a <a href="http://www.google.com/">link</a>.</p>
       <ul>
          <li>a list item</li>
          <li>another list item</li>
          <li>a third list item</li>
      </ul>
   </body>
</html>
```

The resulting DOM tree



Types of nodes

- **Element nodes** (HTML tag)
  - can have children and/or attributes
- **text nodes** (text in a block element)
  - a child within an element node
  - cannot have children or attributes
- **attribute nodes** (attribute/value pair inside the start of a tag)
  - a child within an element node
  - cannot have children or attributes