JavaScript Loops

- Very often when you write code, you want the same block of code to run over and over again in a row.
- Instead of adding several almost equal lines in a script we can use loops to perform a task like this.

JavaScript Loops

- In JavaScript there are two different kind of loops:
- **for** - loops through a block of code a specified number of times.
- **while** - loops through a block of code while a specified condition is true.

The for Loop

- The for loop is used when you know in advance how many times the script should run.

```
for (var=start;var<=end;var=var+increment)
{
   code to be executed
}
```

The for Loop

- Ex: define a loop that starts with i=0. The loop will continue to run as long as i is less than, or equal to 10. i will increase by 1 each time the loop runs.

The for Loop

- Ex: define a loop that starts with i=0. The loop will continue to run as long as i is less than, or equal to 10. i will increase by 1 each time the loop runs.

```
<html>
<body>
<script type="text/javascript">
var i=0
for (i=0;i<=10;i++)
{
document.write("The number is " + i)
document.write("<br />")
}
</script>
</body>
</html>
```

The while loop

- The while loop is used when you want the loop to execute and continue executing while the specified condition is true.

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

The while loop

- Ex: define a loop that starts with i=0. The loop will continue to run as long as i is less than, or equal to 10. i will increase by 1 each time the loop runs.

The while loop

- Ex: define a loop that starts with i=0. The loop will continue to run as long as i is less than, or equal to 10. i will increase by 1 each time the loop runs.

```
<html>
<body>
<script type="text/javascript">
var i=0
while (i<=10)
{
document.write("The number is " + i)
document.write("<br />")
i=i+1
}
</script>
</body>
</html>
```

The do...while Loop

- The do...while loop is a variant of the while loop. This loop will always execute a block of code ONCE, and then it will repeat the loop as long as the specified condition is true.
- This loop will always be executed at least once, even if the condition is false, because the code is executed before the condition is tested.

The do...while Loop

- The do...while loop is a variant of the while loop. This loop will always execute a block of code ONCE, and then it will repeat the loop as long as the specified condition is true.
- This loop will always be executed at least once, even if the condition is false, because the code is executed before the condition is tested.

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

The do...while Loop

- The do...while loop is a variant of the while loop. This loop will always execute a block of code ONCE, and then it will repeat the loop as long as the specified condition is true.
- This loop will always be executed at least once, even if the condition is false, because the code is executed before the condition is tested.

```
<html>
<body>
<script type="text/javascript">
var i=0
do
{
document.write("The number is " + i)
document.write("<br />")
i=i+1
}
while (i<10)
</script>
</body>
</html>
```

JavaScript break and continue Statements

- There are two special statements that can be used inside loops: break and continue.
- **Break**
  - The break command will break the loop and continue executing the code that follows after the loop (if any).

JavaScript break and continue Statements

- There are two special statements that can be used inside loops: break and continue.
- **Break**
  - The break command will break the loop and continue executing the code that follows after the loop (if any).

```
<html>
<body>
<script type="text/javascript">
var i=0
for (i=0;i<=10;i++)
{
if (i==3){break}
document.write("The number is " + i)
document.write("<br />")
}
</script>
</body>
</html>
```

JavaScript break and continue Statements

- The **continue** command will break the current loop and continue with the next value.

JavaScript break and continue Statements

- There are two special statements that can be used inside loops: break and continue.
- **Break**
  - The break command will break the loop and continue executing the code that follows after the loop (if any).

```
<html>
<body>
<script type="text/javascript">
var i=0
for (i=0;i<=10;i++)
{
if (i==3){continue}
document.write("The number is " + i)
document.write("<br />")
}
</script>
</body>
</html>
```

JavaScript For...In Statement

- The **for...in** statement is used to loop (iterate) through the elements of an array or through the properties of an object.
- The code in the body of the **for ... in** loop is executed once for each element/ property.

JavaScript For...In Statement

- The **for...in** statement is used to loop (iterate) through the elements of an array or through the properties of an object.
- The code in the body of the **for ... in** loop is executed once for each element/ property.

```
for (variable in object)
{
  code to be executed
}
```

Example

```
<html>
<body>
<script type="text/javascript">
var x
var mycars = new Array()
mycars[0] = "Saab"
mycars[1] = "Volvo"
mycars[2] = "BMW"

for (x in mycars)
{
document.write(mycars[x] + "<br />")
}
</script>
</body>
</html>
```

Events

- Events are actions that can be detected by JavaScript.
- Every element on a web page has certain events which can trigger JavaScript functions.
- We can use the onClick event of a button element to indicate that a function will run when a user clicks on the button.

Events

- We define the events in the HTML tags.
- Examples of events:
  - ✓ A mouse click
  - ✓ A web page or an image loading
  - ✓ Mousing over a hot spot on the web page
  - ✓ Selecting an input box in an HTML form
  - ✓ Submitting an HTML form
  - ✓ A keystroke

onload and onUnload

- The onload and onUnload events are triggered when the user enters or leaves the page.
- The onload event is often used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.

onload and onUnload

- Both the onload and onUnload events are also often used to deal with cookies that should be set when a user enters or leaves a page.
- For example, you could have a popup asking for the user's name upon his first arrival to your page.

onload and onUnload

```
<html>
<head>
<script type="text/javascript">
var username="guest"
function load()
{
username=prompt("enter your username", username)
document.getElementById("name").innerHTML = username;
}

function writeusername()
{
document.write("Hello")
}
</script>
</head>
<body onload="load()">
Hello, <span id="name"></span><br>
</body>
```

- Both the onload and onUnload events are also often used to deal with cookies that should be set when a user enters or leaves a page.
- For example, you could have a popup asking for the user's name upon his first arrival to your page.

onFocus, onBlur and onChange

- The onFocus, onBlur and onChange events are often used in combination with validation of form fields.
- Below is an example of how to use the onChange event. The checkEmail() function will be called whenever the user changes the content of the field

onFocus, onBlur and onChange

- The onFocus, onBlur and onChange events are often used in combination with validation of form fields.
- Below is an example of how to use the onChange event. The checkEmail() function will be called whenever the user changes the content of the field

```
<input type="text" size="30"
id="email" onchange="checkEmail()">;
```

onSubmit

- The onSubmit event is used to validate ALL form fields before submitting it.
- The checkForm() function will be called when the user clicks the submit button in the form.
- If the field values are not accepted, the submit should be cancelled.

onSubmit

- The function checkForm() returns either true or false.
- If it returns true the form will be submitted, otherwise the submit will be cancelled.

onSubmit

- The function checkForm() returns either true or false.
- If it returns true the form will be submitted, otherwise the submit will be cancelled.

```
<form method="post" action="xxx.php"
onsubmit="return checkForm()">
```

onMouseOver and onMouseOut

- onMouseOver and onMouseOut are often used to create "animated" buttons.
- As example of an onMouseOver event. An alert box appears when an onMouseOver event is detected.

onMouseOver and onMouseOut

- onMouseOver and onMouseOut are often used to create "animated" buttons.
- As example of an onMouseOver event. An alert box appears when an onMouseOver event is detected.

```
<a href="http://www.somewhere.com"
onmouseover="alert('An onMouseOver event');return
false">
<img src="myIcon.gif" width="100" height="30">
</a>
```

JavaScript - Catching Errors

- There are two ways of catching errors in a Web page:
  - By using the try...catch statement (available in IE5+, Mozilla 1.0, and Netscape 6)
  - By using the onerror event. This is the old standard solution to catch errors (available since Netscape 3)

Try...Catch Statement

- The try...catch statement allows you to test a block of code for errors.
- The **try** block contains the code to be run, and the **catch** block contains the code to be executed if an error occurs.

Try...Catch Statement

- The try...catch statement allows you to test a block of code for errors.
- The **try** block contains the code to be run, and the catch **block** contains the code to be executed if an error occurs.

```
try
{
//Run some code here
}
catch(err)
{
//Handle errors here
}
```

Example

```
<html>
<head>
<script type="text/javascript">
function message()
{
adddlert("Welcome guest!")
}
</script>
</head>

<body>
<input type="button" value="View message"
onclick="message()" />
</body>
</html>
```

```
function message()
{
adddlert("Welcome guest!")
}
```

Example

```
function message()
{
try
  {
  adddlert("Welcome guest!")
  }
catch(err)
  {
  txt="There was an error on this page.\n\n"
  txt+="Error description: " + err.description
  txt+="\nClick OK to continue.\n\n"
  alert(txt)
  }
}
```

The Throw Statement

- The throw statement allows you to create an exception.
- If you use this statement together with the **try...catch** statement, you can control program flow and generate accurate error messages.

The Throw Statement

- The throw statement allows you to create an exception.
- If you use this statement together with the **try...catch** ~~statement~~, you can control program flow and generate accurate error messages.

`throw(exception)`

Example

```
<html>
<body>
<script type="text/javascript">
var x=prompt("Enter a number (0..10):","")
try
{
if(x>10)
throw "Err1"
else if(x<0)
throw "Err2"
}
catch(er)
{
if(er=="Err1")
alert("Error! The [illegible]ue is too high")
if(er == "Err2")
alert("Error! The value is too low")
}
</script>
</body>
</html>
```

The onerror Event

- The onerror event is fired whenever there is a script error in the page.
- To use the onerror event, you must create a function to handle the errors.
- Then you call the function with the onerror event handler.

The onerror Event

- The event handler is called with three arguments:
  - msg (error message)
  - url (the url of the page that caused error)
  - line (the line where the error occurred)

The onerror Event

- The event handler is called with three arguments:
  - msg (error message)
  - url (the url of the page that caused error)
  - line (the line where the error occurred)

```
onerror=handleErr

function handleErr(msg,url,l)
{
//Handle the error here
return true or false
}
```

The onerror Event

- The value returned by onerror determines whether the browser displays a standard error message.
- If you return **false**, the browser displays the standard error message in the JavaScript console.
- If you return **true**, the browser does not display the standard error message.

Example

```
<html>
<head>
<script type="text/javascript">
onerror=handleErr
var txt=""

function handleErr(msg,url,l)
{
txt="There was an err   on this page.\n\n"
txt+="Error: " + msg + "\n"
txt+="URL: " + url + "\n"
txt+="Line: " + l +   \n"
txt+="Click OK to    tinue.\n\n"
alert(txt)
return true
}
```

Example

```
function message()
{
adddlert("Welcome guest!")
}
</script>
</head>

<body>
<input type="button" value="View message"
onclick="message()" />
</body>

</html>
```

javascript objects

costantino pistagna <pistagna@dmi.unict.it>

A Basic Class & Istance

```
function myFunc(){
}

var myObject = new myFunc();
alert(typeof myObj   );  // displays "object"
```

We've just created out own object.

How does JavaScript know how to create an object?

```
function myFunc(){
 return 5;
}

var myObject = myFunc();
alert(typeof myObject); // displays "number"
```

How does JavaScript know how to create an object?

```
function myFunc(){
 return 5;
}

var myObject = myFunc();
alert(typeof myObject); // displays "number"
```

The answer is:

NEW

Public Member Variable & Functions

Public functions and variables
Are available to access on an instance of a class.

Public Member Variable & Functions

Public functions and variables
Are available to access on an instance of a class.

```
//public member variable
MyClass.prototype.publicVar = “My Public Variable”;

//public member function
MyClass.prototype.publicFunction = function () {
      alert( this.publicVar );
}
```

Create an istance and run a member function

```
//public member variable
MyClass.prototype.publicVar =  “My Public Variable”;

//public member function
MyClass.prototype.publicFunction = function () {
      alert( this.publicVar );
}
```

```
//create an instance
var oClass = new MyClass();

//run a member function
oClass.publicFunction();   //Alert: “My Public
Variable”
```

Private members

Private member functions and variables are hidden to outside code. Only public functions can access them.

Private members

Private member functions and variables are hidden to outside code. Only public functions can access them.

```
function MyClass () {
   //reference to this
   var self = this;
   //private member variable
   var privateVar = “My Private Variable”;
   //public member variable
   this.publicVar = “My Public Variable”;
   //private member function
   var privateFunction = function () {
      self.publicVar += ” Modified By A Private Fucntion”;
      alert( self.publicVar );
   }
}

//create an instance
var oClass = new MyClass();
//run a private member function
oClass.privateFunction();   //private function is undefined
//get a private member var
alert( oClass.privateVar );   //private var is undefined
```

Static members

A static function or variable is available on the base class but is not available to the class instance.

Static members

A static function or variable is available on the base class but is not available to the class instance.

```
function MyClass () {
   //...
}

//declare a static member
MyClass.staticVar = “My static variable”;
//declare a static function
MyClass.staticFunction = function ( pInput ) {
   return new MyClass( MyClass.staticVar , pInput );
}

//create an instance
var oClass = new MyClass();

//run a static function (NO access to private or public)
oClass.staticFunction( 9 );  //staticFunction is undefined on an instance
//run a privileged member function on the class
MyClass.privilegedFunction();   //The function runs
```

Privileged members

A privileged member function:
- Has access to private variables
- Is available publicly.

Privileged members

```
function MyClass () {

   //private member variable
   var privateVar = "My Private Variable";

   //privileged member function
   this.privilegedFunction = function () {
      alert( privateVar );
   }
}
//create an instance
var oClass = new MyClass();

//run a privileged member function
//Output: alerts the value of the private var
oClass.privilegedFunction();
```

A privileged member function:

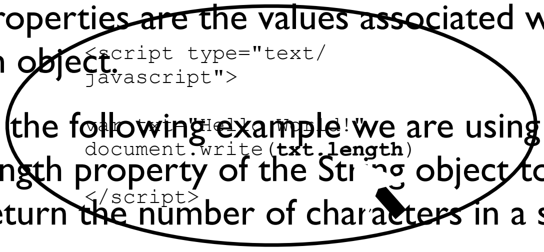
- Has access to private variables
- Is available publicly.

Properties

- Properties are the values associated with an object.
- In the following example we are using the length property of the String object to return the number of characters in a string

Properties

- Properties are the values associated with an object.
- In the following example we are using the length property of the String object to return the number of characters in a string



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

var txt="Hello World!"
document.write(txt.length)

</script>
```

Methods

- Methods are the actions that can be performed on objects.
- In the following example we are using the toUpperCase() method of the String object to display a text in uppercase letters:

Methods

- Methods are the actions that can be performed on objects.
- In the following example we are using the toUpperCase() method of the String object to display a text in uppercase letters:

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

var str="Hello world!"
document.write (str.toUpperCase())

</script>
```

String object

- The String object is used to manipulate a stored piece of text.

```
var txt="Hello world!"
document.write(txt.length)
```

```
var txt="Hello world!"
document.write(txt.toUpperCase())
```

Defining Dates

- The Date object is used to work with dates and times.
- We define a Date object with the new keyword. The following code line defines a Date object called myDate:

```
var myDate=new Date()
```

Manipulate Dates

- We can easily manipulate the date by using the methods available for the Date object.
- In the example below we set a Date object to a specific date (14th January 2008):

```
var myDate=new Date()
myDate.setFullYear(2008,0,14)
```

Manipulate Dates

- And in the following example we set a Date object to be 5 days into the future:

```
var myDate=new Date()
myDate.setDate(myDate.getDate()+5)
```

Comparing Dates

- The Date object is also used to compare two dates.
- The following example compares today's date with the 14th January 2010.

Comparing Dates

- The Date object is also used to compare two date.
- The following example compares today's date with the 14th January 2010.

```
var myDate=new Date()
myDate.setFullYear(2010,0,14)
var today = new Date()

if (myDate>today
  alert("Today is before 14th January
2010")
else
  alert("Today is after 14th January 2010")
```

Defining Arrays

- The Array object is used to store a set of values in a single variable name.
- We define an Array object with the **new** keyword.

Defining Arrays

- The Array object is used to store a set of values in a single variable name.
- We define an Array object with the **new** keyword.

```
var myArray=new Array()
```

Defining Arrays

- There are two ways of adding values to an array:
  - you can add as many values as you need to define as many variables you require.

Defining Arrays

- There are two ways of adding values to an array:
  - you can add as many values as you need to define as many variables you require.

```
var mycars=new Array()
mycars[0]="Saab"
mycars[1]="Volvo"
mycars[2]="BMW"
```

Defining Arrays

- You could also pass an integer argument to control the array's size

Defining Arrays

- You could also pass an integer argument to control the array's size.

```
var mycars=new Array(3)
mycars[0]="Saab"
mycars[1]="Volvo"
mycars[2]="BMW"
```

Defining Arrays

- You could also pass an integer argument to control the array's size

```
var mycars=new Array("Saab","Volvo","BMW")
```

Accessing Arrays

- You can refer to a particular element in an array by referring to the name of the array and the index number.
- The index number starts at 0.

Accessing Arrays

- You can refer to a particular element in an array by referring to the name of the array and the index number.
- The index number starts at 0.

```
document.write(mycars[0])
```

Modify Values in Existing Arrays

- To modify a value in an existing array, just add a new value to the array with a specified index number.

Modify Values in Existing Arrays

- To modify a value in an existing array, just add a new value to the array with a specified index number.

```
mycars[0]="Opel";
```

Boolean Object

- The Boolean object is an object wrapper for a Boolean value.
- The Boolean object is used to convert a non-Boolean value to a Boolean value (true or false).
- We define a Boolean object with the new keyword.

Boolean Object

- The Boolean object is an object wrapper for a Boolean value.
- The Boolean object is used to convert a non-Boolean value to a Boolean value (true or false).
- We define a Boolean object with the new keyword.

```
var myBoolean=new Boolean()
```

Math Object

- The Math object allows you to perform common mathematical tasks.
- The Math object includes several mathematical values and functions.
- You do not need to define the Math object before using it.

Math Object

- The Math object allows you to perform common mathematical tasks.

```
document.write(Math.round(4.7))
```

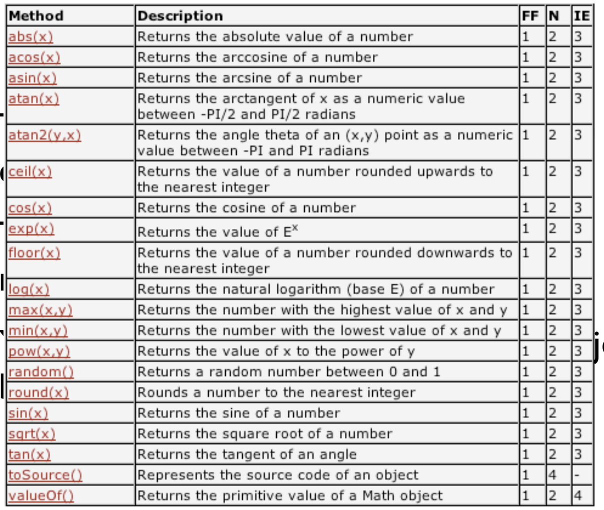
- The Math object includes several mathematical values and functions.

```
document.write(Math.random())
```

- You do not need to define the Math object before using it.

```
document.write(Math.floor(Math.random()*11))
```

Math Object



Method	Description	FF	N	IE
abs(x)	Returns the absolute value of a number	1	2	3
acos(x)	Returns the arccosine of a number	1	2	3
asin(x)	Returns the arcsine of a number	1	2	3
atan(x)	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians	1	2	3
atan2(y,x)	Returns the angle theta of an (x,y) point as a numeric value between -PI and PI radians	1	2	3
ceil(x)	Returns the value of a number rounded upwards to the nearest integer	1	2	3
cos(x)	Returns the cosine of a number	1	2	3
exp(x)	Returns the value of E^x	1	2	3
floor(x)	Returns the value of a number rounded downwards to the nearest integer	1	2	3
log(x)	Returns the natural logarithm (base E) of a number	1	2	3
max(x,y)	Returns the number with the highest value of x and y	1	2	3
min(x,y)	Returns the number with the lowest value of x and y	1	2	3
pow(x,y)	Returns the value of x to the power of y	1	2	3
random()	Returns a random number between 0 and 1	1	2	3
round(x)	Rounds a number to the nearest integer	1	2	3
sin(x)	Returns the sine of a number	1	2	3
sqrt(x)	Returns the square root of a number	1	2	3
tan(x)	Returns the tangent of an angle	1	2	3
toSource()	Represents the source code of an object	1	4	-
valueOf()	Returns the primitive value of a Math object	1	2	4

ject

The HTML DOM

- The HTML DOM is a W3C standard and it is an abbreviation for the Document Object Model for HTML.
- The HTML DOM defines a standard set of objects for HTML, and a standard way to access and manipulate HTML documents.

The HTML DOM

- All HTML elements, along with their containing text and attributes, can be accessed through the DOM.
- The contents can be modified or deleted, and new elements can be created.

The HTML DOM

- The HTML DOM is platform and language independent.
- It can be used by any programming language like Java, **JavaScript**, and VBScript.

Browser Detection

- there are some things that just don't work on certain browsers - specially on older browsers.
- So, sometimes it can be very useful to detect the visitor's browser type and version, and then serve up the appropriate information.

Browser Detection

- The best way to do this is to make your web pages smart enough to look one way to some browsers and another way to other browsers.
- JavaScript includes an object called the **Navigator** object, that can be used for this purpose.

Browser Detection

- The best way to do this is to make your web pages smart enough to look one way to some browsers and another way to other browsers.
- JavaScript includes an object called the **Navigator** object, that can be used for this purpose.

```
<html>
<body>
<script type="text/javascript">
var browser=navigator.appName
var b_version=navigator.appVersion
var version=parseFloat(b_version)
document.write("Browser name: "+ browser)
document.write("<br />")
document.write("Browser version: "+ version)
</script>
</body>
</html>
```

Browser Detection

- The best way to do this is to make your web pages smart enough to look one way to some browsers and another way to other browsers.
- JavaScript includes an object called the **Navigator** object, that can be used for this purpose.

```
<html>
<head>
<script type="text/javascript">
function detectBrowser()
{
var browser=navigator.appName
var b_version=navigator.appVersion
var version=parseFloat(b_version)
if ((browser=="Netscape"||browser=="Microsoft Internet Explorer")
&& (version>=4))
  {alert("Your browser is good enough!")}
else
  {alert("It's time to upgrade your browser!")}
}
</script>
</head>

<body onload="detectBrowser()">
</body>
</html>
```

What is a Cookie?

- A cookie is a variable that is stored on the visitor's computer.
- Each time the same computer requests a page with a browser, it will send the cookie too.
- With JavaScript, you can both create and retrieve cookie values.

Create and Store a Cookie

- We will create a cookie that stores the name of a visitor.
- The first time a visitor arrives to the web page, he will be asked to enter his name
- The name is then stored in a cookie.
- The next time the visitor arrives at the same page, he will get welcome message.

Create and Store a Cookie

- We will create a cookie that stores the name of a visitor.
- The first time a visitor arrives to the web page, he will be asked to enter his name
- The name is then stored in a cookie.
- The next time the visitor arrives at the same page, he will get welcome message.

```
function setCookie(c_name,value,expiredays)
{
var exdate=new Date()
exdate.setDate(exdate.getDate()+expiredays)
document.cookie=c_name+ "=" +escape(value)+
((expiredays==null) ? "" : ";expires="+exdate.toGMTString())
}
```

Create and Store a Cookie

- We will create a cookie that stores the name of a visitor.
- The first time a visitor arrives to the web page, he will be asked to enter his name
- The name is then stored in a cookie.
- The next time the visitor arrives at the same page, he will get welcome message.

```
function getCookie(c_name)
{
if (document.cookie.length>0)
  {
  c_start=document.cookie.indexOf(c_name + "=")
  if (c_start!=-1)
    {
    c_start=c_start + c_name.length+1
    c_end=document.cookie.indexOf(";",c_start)
    if (c_end==-1) c_end=document.cookie.length
    return unescape(document.cookie.substring(c_start,c_end))
    }
  }
return ""
}
```

Create and Store a Cookie

- We will create a cookie that stores the name of a visitor.
- The first time a visitor arrives to the web page, he will be asked to enter his name
- The name is then stored in a cookie.
- The next time the visitor arrives at the same page, he will get welcome message.

```
function checkCookie()
{
username=getCookie('username')
if (username!=null && username!="")
  {alert('Welcome again '+username+'!')}
else
  {
  username=prompt('Please enter your name:',"")
  if (username!=null && username!="")
    {
    setCookie('username',username,365)
    }
  }
}
```

JavaScript Form Validation

```
function validate_required(field,alerttxt)
{
with (field)
{
if (value==null||value=="")
  {alert(alerttxt);return false}
else {return true}
}
}
```

JavaScript Form Validation

```
function validate_form(thisform)
{
with (thisform)
{
if (validate_required(email,"Email must be filled out!")==false)
  {email.focus();return false}
}
}
```

JavaScript Form Validation

```
<body>
<form action="submitpage.htm"
onsubmit="return validate_form(this)"
method="post">
Email: <input type="text" name="email" size="30">
<input type="submit" value="Submit">
</form>
</body>

</html>
```

JavaScript Code More Readable Using with()

- Using with(), you can reduce object references and make your code more readable.
- It is possible to have nested with() statements.

JavaScript Code More Readable Using with()

- Using with(), you can reduce object references and make your code more readable.
- It is possible to have nested with() statements.

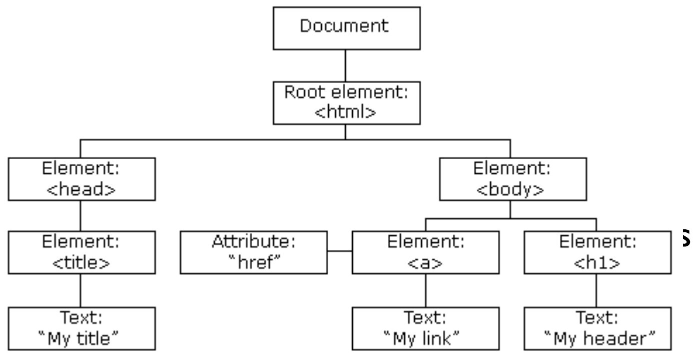
```
function foo(){
   var x = document.forms[0].elements[0].value;
   var y = document.forms[0].elements[1].value;
   var z = document.forms[0].elements[2].options[
         document.forms[0].elements[2].selectedIndex].text;
}
```

```
function foo(){
   with(document.forms[0]){
      var x = elements[0].value;
      var y = elements[1].value;
   with(elements[2]){
   var z = options[selectedIndex].text
}
}
}
}
```

HTML DOM

- The HTML Document Object Model (HTML DOM) defines a standard way for accessing and manipulating HTML documents.
- The DOM presents an HTML document as a tree-structure (a node tree), with elements, attributes, and text.

HTML DOM



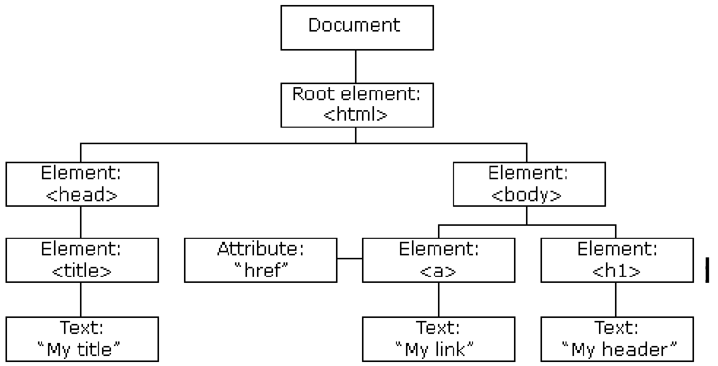
Nodes

- According to the DOM, everything in an HTML document is a node.
- The DOM says that:
  - The entire document is a document node
  - Every HTML tag is an element node
  - The texts contained in the HTML elements are text nodes
  - Every HTML attribute is an attribute node
  - Comments are comment nodes

Node Hierarchy

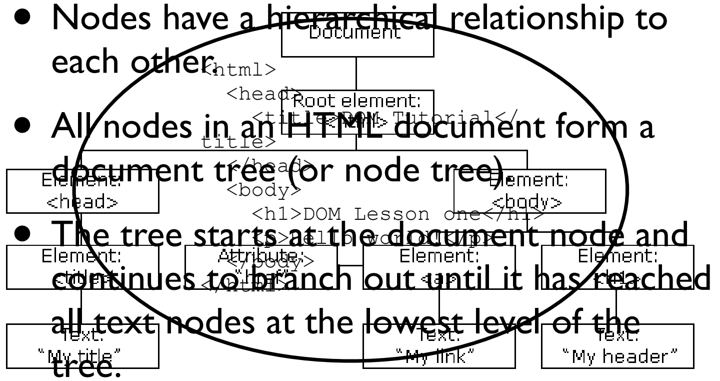
- Nodes have a hierarchical relationship to each other.
- All nodes in an HTML document form a document tree (or node tree).
- The tree starts at the document node and continues to branch out until it has reached all text nodes at the lowest level of the tree.

Node Hierarchy



Node Hierarchy

- Nodes have a hierarchical relationship to each other.
- All nodes in an HTML document form a document tree (or node tree).
- The tree starts at the document node and continues to branch out until it has reached all text nodes at the lowest level of the tree.



```
<html>
  <head>
    <title>DOM Tutorial</
title>
  </head>
  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>
</html>
```

Find and Access Nodes

- You can find the element you want to manipulate in several ways:
  - By using the getElementById() and getElementsByTagName() methods
  - By using the parentNode, firstChild, and lastChild properties of an element node

getElementById() and getElementsByTagName()

- The two methods getElementById() and getElementsByTagName(), can find any HTML element in the entire document.
- These methods ignore the document structure.
- If you want to find all <p> elements in the document, the getElementsByTagName() method will find them all

getElementById()

- The getElementById() method returns the element with the specified ID:

```
document.getElementById("someID");
```

getElementsByTagName()

- The getElementsByTagName() can be used on any HTML element, and also on the document

```
document.getElementsByTagName("tagname");
```

Example

- The following example returns a nodeList of all <p> elements that are descendants of the element with id="maindiv":

```
document.getElementById('maindiv').getElementsByTagName("p");
```

nodeList

- When working with a nodeList, we usually store the list in a variable

```
var x=document.getElementsByTagName("p");
```

nodeList

- Now the variable x contains a list of all <p> elements in the page, and we can access the <p> elements by their index numbers.

```
var x=document.getElementsByTagName("p");
for (var i=0;i<x.length;i++)
  {
  // do something with each paragraph
  }
```

Example

```
<html>
<body>

<form name="Form1"></form>
<form name="Form2"></form>
<form name="Form3"></form>

<script type="text/javascript">
document.write("This document contains: " + document.forms.length
+ " forms.")
</script>

</body>
</html>
```

Example

```
<html>
<body>
<form id="Form1" name="Form1">Your name: <input type="text">
</form>
<form id="Form2" name="Form2">Your car: <input type="text">
</form>
<p>To access an item in a collection you can either use the
number or the name of the item:</p>

<script type="text/javascript">
document.write("<p>The first form's name is: " +
document.forms[0].name + "</p>")
document.write("<p>The first form's name is: " +
document.getElementById("Form1").name + "</p>")
</script>

</body>
</html>
```

Example

```
<html>
<body>
<img border="0" src="hackanm.gif" width="48" height="48">
<br />
<img border="0" src="compman.gif" width="107" height="98">
<br /><br />

<script type="text/javascript">
document.write("This document contains: " +
document.images.length + " images.")
</script>

</body>
</html>
```

Example

```
<html>
<head>
<script type="text/javascript">
function changeSrc()
  {
  document.getElementById("myImage").src="hackanm.gif"
  }
</script>
</head>

<body>
<img id="myImage" src="compman.gif" width="107" height="98" />
<br /><br />
<input type="button" onclick="changeSrc()" value="Change image">
</body>

</html>
```

Example

```
<html>
<head>
<script type="text/javascript">
function changeSize()
{
document.getElementById("compman").height="250"
document.getElementById("compman").width="300"
}
</script>
</head>

<body>
<img id="compman" src="compman.gif" width="107" height="98" />
<br /><br />
<input type="button" onclick="changeSize()" value="Change height
and width of image">
</body>

</html>
```

Example

```
<html>
<head>
<script type="text/javascript">
function whichButton(event) {
if (event.button==2) {
alert("You clic[redacted]d the right mouse button!")
}
else{
alert [redacted]You clicked the left mouse button!")
}
}
</script>
</head>
<body onmousedown="whichButton(event)">
<p>Click in the document. An a[redacted]rt box will alert which mouse
button you clicked.</p>
</body>
</html>
```

Example

```
<html>
<head>
<script type="text/javascript">
function show_coords(event) {
x=event.clientX
y=event.clientY
alert("X coords: [redacted]+ x + ", Y coords: " + y)
}
</script>
</head>

<body onmousedown="show_coords(event)">
<p>Click in the d[redacted]ment. An alert box will alert the x and y
coordinates of the [redacted]ouse pointer.</p>
</body>
</html>
```

Example

```
<html>
<head>
<script type="text/javascript">
function whichButton(event) {
alert(event.keyCode)
}

</script>
</head>
<body onkeyup="whichButton(event)">
<p><b>Note:</b> Make sure the right frame has focus when trying
this example!</p>
<p>Press a key on your keyboard. An alert box will alert the
unicode of the key pressed.</p>
</body>
</html>
```

Example

```
<html>
<head>
<script type="text/javascript">
function changeAction() {
var x=document.getElementById("myForm")
alert("Original action: " + x.action)
x.action="http://127.0.0.1"
alert("New action: " + x.action)
x.submit()
}
</script>
</head>
<body>
<form id="myForm" action="esempio.php">
Name: <input type="text" value="" />
<input type="button" onclick="changeAction()"
value="Change action attribute and submit form" />
</form>
</body>
</html>
```

Example

```
<html>
<head>
<script type="text/javascript">
function alertId() {
var txt="Id: " + document.getElementById("myButton").id
txt=txt + ", type: " + document.getElementById("myButton").type
txt=txt + ", value: " + document.getElementById("myButton").val
alert(txt)
document.getElementById("myButton").disabled=true
}
</script>
</head>
<body>
<form>
<input type="button" value="Click me!" id="myButton"
onclick="alertId()" />
</form>
</body>
</html>
```

Example

```
<script type="text/javascript">
function favBrowser() {
var mylist=document.getElementById("myList")
document.getElementById("favorite").value=mylist.options[mylist.s
electedIndex].text
}
</script>

<form>
Select your favorite browser:
<select id="myList" onchange="favBrowser()">
<option>Internet Explorer</option>
<option>Netscape</option>
<option>Opera</option>
</select>
<p>Your favorite browser is: <input type="text" id="favorite"
size="20"></p>
</form>
```

Example

```
<html>
<head>
<script type="text/javascript">
function go()
{
window.location=document.getElementById("menu").value
}
</script>
</head>

<body>
<form>
<select id="menu" onchange="go()">
  <option>--Select a page--</option>
  <option value="http://www.w3c.org">w3c</option>
  <option value="http://www.apple.com">Apple</option>
  <option value="http://www.gentoo.org">gentoo</option>
</select>
</form>
</body>

</html>
```

DOM Node List

- The getElementsByTagName() method returns a node list.
  - A node list is an array of nodes.

DOM Node List

- The following code stores a list of <p> nodes (a node list) in the variable x:
- The getElementsByTagName() method returns a node list.
  - A node list is an array of nodes.

```
x = document.getElementsByTagName("tagname");
```

DOM Node List

- The <p> elements in x can be accessed by index number.
- To access the second <p> you can write:

```
y=x[1];
```

Example

```
<html>
<body>

<p id="intro">DOM example</p>
<div id="main">
<p id="main1">The DOM is very useful</p>
<p id="main2">This example demonstrates how to use the
<b>getElementsByTagName and node list</b></p>
</div>
<script type="text/javascript">
x=document.getElementsByTagName("p");
document.write("Second paragraph text: " + x[1].innerHTML);
</script>

</body>
</html>
```

DOM Node List Length

- The length property defines the length of a node list (the number of nodes).
- You can loop through a node list by using the length property:

DOM Node List Length

- The length property defines the length of a node list (the number of nodes).
- You can loop through a node list by using the length property:

```
x=document.getElementsByTagName("p");
for (i=0;i<x.length;i++)
  {
  document.write(x[i].innerHTML);
  document.write("<br />");
  }
```

Navigating Node Relationships

- The three properties *parentNode*, *firstChild*, and *lastChild* follow the document structure
- They allow short-distance travel in the document.

Navigating Node Relationships

```
<html>
<body>

<p id= 
<div id=
<p id="m
<p id="m                     ates <b>node relationships</b></p>
</div>

</body>
</html>
```

- The three properties *parentNode*, *firstChild*, llow the document structure
- T stance travel in the d

parentNode

lastChild

Example

```
<html>
<body>

<p id="intro">Relationships example</p>
<div id="main">
<p id="main1">The DOM is very useful</p>
<p id="main2">This example demonstrates how to use the
<b>firstChild</b> property</p>
</div>
<script type="text/javascript">
x=document.getElementById("intro");
document.write("T   inside the first paragraph: " +
x.firstChild.nodeV  e);
</script>

</body>
</html>
```

Root Nodes

- There are two special document properties that allow access to the tags:
  - document.documentElement
  -

The first property returns the root

The second property is a special addition for HTML pages, and gives direct access to the <body> tag.

Example

```
<html>
<body>

<p id="intro">Document Roots example</p>
<div id="main">
<p id="main1">The DOM is very useful</p>
<p id="main2">This example demonstrates how to use
<b>document.body</b></p>
</div>
<script type="text/javascript">
x=document.body;
alert(x.innerHTML);
</script>

</body>
</html>
```

Node Properties

- In the HTML Document Object Model (DOM), each node is an object.
- Objects have methods (*functions*) and properties *(information about the object*), that can be accessed and manipulated by JavaScript.

Node Properties

- Three important HTML DOM node properties are:
  - *nodeName*
  - *nodeValue*
  - *nodeType*

- In the HTML Document Object Model (DOM), each node is an object.
- Objects have methods (*functions*) and properties *(information about the object*), that can be accessed and manipulated by JavaScript.

The nodeName Property

- The nodeName property specifies the name of a node.
  - nodeName is read-only.
  - nodeName of an element node is the same as the tag name.
  - nodeName of an attribute node is the attribute name.
  - nodeName of a text node is always **#text**.
  - nodeName of the document node is always **#document**.

The nodeValue Property

- The nodeValue property specifies the value of a node.
  - nodeValue for element nodes is undefined.
  - nodeValue for text nodes is the text itself.
  - nodeValue for attribute nodes is the attribute value.

Example: Get the Value of an Element

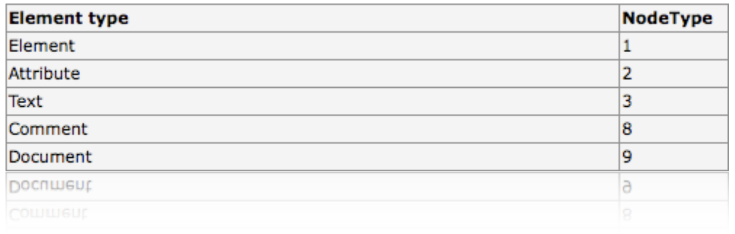
```
<html>
<body>

<p id="intro">Get value example</p>
<div id="main">
<p id="main1">The DOM is very useful</p>
<p id="main2">This example demonstrates how to use the
<b>nodeValue</b> property</p>
</div>
<script type="text/javascript">
x=document.getElementById("intro").firstChild;
document.write("Fi[illegible] paragraph text: " + x.nodeValue);
</script>

</body>
</html>
```

The nodeType Property

- The nodeType property returns the type of node.
  - nodeType is read only.
- The most important node types are:



Element type	NodeType
Element	1
Attribute	2
Text	3
Comment	8
Document	9

Changing an HTML Element

- The HTML DOM and JavaScript can be used to change the inner content and attributes of HTML elements **dynamically**.

Changing an HTML Element

- The HTML DOM and JavaScript can be used to change the inner content and attributes of HTML elements **dynamically**.

```
<html>
<body>
<script type="text/javascript">
document.body.bgColor="yellow";
</script>
</body>
</html>
```

The following example changes the background color of the <body> element.

Changing an HTML Element

- The easiest way to get or modify the content of an element is by using the innerHTML property.

Changing an HTML Element

- The easiest way to get or modify the content of an element is by using the innerHTML property.

```
<html>
<body>
<p id="p1">Hello World!</p>
<script type="text/javascript">
document.getElementById("p1").innerHTML="New text!";
</script>
</body>
</html>
```

The following example changes the text of the <p> element.

Changing an HTML Element *Using Events*

- An event handler allows you to execute code when an event occurs.
- Events are generated by the browser when the user clicks an element, when the page loads, when a form is submitted, etc.

Changing an HTML Element *Using Events*

- An event handler allows you to execute code when an event occurs.
- Events are generated by the browser when the user clicks an element, when the page loads, when a form is submitted, etc.

```
<html>
<body>
<input type="button"
onclick="document.body.bgColor='yellow';"
value="Click me to change background color">
</body>
</html>
```

The following example changes the background color of the <body> element when the button is clicked.

Example

```
<html>
<head>
<script type="text/javascript">
function ChangeText() {
   document.getElementById("p1").innerHTML="New text!";
}
</script>
</head>
```

onclick="ChangeText()"

The following example uses a function to change the text of the <p> element when the button is clicked.

Using the Style Object

- The Style object represents of each HTML element represents its individual style.
- The Style object can be accessed from the document or from the elements to which that style is applied.

Using the Style Object

- The Style object represents of each HTML element represents its individual style.
- The Style ob ect can be accessed from the document or from the elements to which that style is applied

```
<html>
<head>
<script type="text/javascript">
function ChangeBackground() {
document.body.style.backgroundColor="yellow";
}
</script>
</head>
```

onclick="ChangeBackground()"

The following example uses a function to change the style of the <body> element when the button is clicked.

Example

```
<html>
<head>
<script type="text/javascript">
function ChangeText() {
document.getElementById("p1").style.color="blue";
document.getElementById("p1").style.fontFamily="Arial";
}
</script>
</head>
<
<
<
v
<
<
```

The following example uses a function to change the style of the <p> element when the button is clicked.