

Tecnologie Web

JavaScript

Francesco Marcelloni

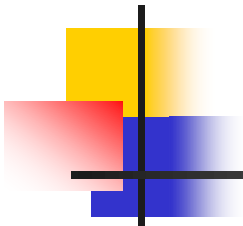
Dipartimento di Ingegneria dell'Informazione
Università di Pisa
ITALY





What is JavaScript?

- JavaScript is a **scripting language**.
- JavaScript **works on the client side**, which means that it runs on the user's computer and not on the Web server.
- JavaScript **is designed for use on Web pages** and is closely integrated with HTML.
- **JavaScript statements** embedded in an HTML page **can recognize and respond to User Events** such as Mouse Clicks, Form Input, and Page Navigation.



Scripting Languages

- Scripts offer authors a means to extend HTML documents in highly active and interactive ways. For example:
 - Scripts may be evaluated as a document loads to modify the contents of the document dynamically.
 - Scripts may accompany a form to process input as it is entered.
 - Scripts may be triggered by events that affect the document, such as loading, unloading, element focus, mouse movement, etc.
 - Scripts may be linked to form controls (e.g., buttons) to produce graphical user interface elements.

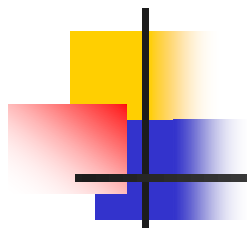




Main Features

Like all the scripting languages, JavaScript:

- **is interpreted** (easy to test program code, look at the results, make changes, and test it)
- **contains a limited and easy-to-learn command set and syntax**
- is designed for performing a **well-defined set of tasks**
- is suited to **producing small programs**
- is especially well designed for **repetitive, event-invoked tasks**
- is **integrated into the browser** and can interact directly with the **HTML pages**
- JavaScript makes it possible **to program responses to user events** such as mouse clicks and data entry in forms



Main Features

Strengths	Weakness
Quick Development	Limited Range of Built-in Methods
Easy to Learn	Generally, only client-side script
Platform Independence	No Code Hiding
Small Overhead	



JavaScript Objectives

- To make HTML pages dynamic, interactive and effective through graphics and animations.
- To develop client-side applications with particular functions on the HTML object management
- To validate the data input from the users
- To manage search tables on the client
- To store and retrieve the cookies so as to obtain information on the client-server interaction



JavaScript Standard

- JavaScript was invented by Netscape and was first used in Netscape browsers.
- Now, all the browsers support JavaScript
- **ECMAScript, standardized version of JavaScript, is documented in the ECMA-262 specification (5.1 Edition).**
- ECMA (European Computer Manufacturers Association). ECMA is an international standards association for information and communication systems.
- The **ECMA-262** standard is also approved by the ISO (International Organization for Standards) as **ISO-16262**.



What cannot you do with JavaScript

- You cannot access or write to server files
- You do, however, have access to documents that are loaded into other browser windows and frames, provided that the documents are from the same server
- You also cannot touch the local hard drive, send a print job, or edit the user's bookmarks or browser preferences. NOTE: The exception are cookies – which JavaScript can read from the hard drive and write to the hard drive.

JavaScript versus Object-Oriented Languages

JavaScript	Java-C++
Object-based. Code uses built-in, extensible objects, but no classes or inheritance.	Object-oriented. Applets consist of object classes with inheritance.
Variable data types not declared (loose typing).	Variable data types must be declared (strong typing).
You can write fully-functional JavaScript using a simple text editor	To write fully-functional Java/C++, you need the Java/C++ Developer's Kit.

How to insert a JavaScript into an HTML page?

- Use the `<script>` tag. Inside the `<script>` tag use the `type` attribute to define the scripting language.

```
<html>
```

```
<body>
```

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

```
...
```

```
</script>
```

```
</body>
```

```
</html>
```

- The script element may appear any number of times in the head or body of an HTML document.

How to insert a JavaScript into an HTML page?

- `<script>` tag

- `src = uri`

This attribute specifies the location of an external script.

- `type = content-type`

This attribute specifies the scripting language of the element's contents and overrides the default scripting language.

- The scripting language is specified as a content type (e.g., "text/javascript").
 - Authors must supply a value for this attribute. There is no default value for this attribute.

How to insert a JavaScript into an HTML page?

■ Example

- The document.write command is a standard JavaScript command for writing output to a page.
- By entering the document.write command between the `<script>` and `</script>` tags, the browser will recognize it as a JavaScript command and execute the code line.

```
<html>
```

```
<body>
```

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

```
    document.write("Hello World!");
```

```
</script>
```

```
</body>
```

```
</html>
```



How to handle simple browsers?

- Browsers that **do not support JavaScript** will display **JavaScript as page content**.
- To prevent them from doing this, and as a part of the JavaScript standard, **the HTML comment tag should be used to "hide" the JavaScript**.
- Just add an HTML comment tag `<!--` before the first JavaScript statement, and a `-->` (end of comment) after the last JavaScript statement.

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



Where to insert the JavaScript Code



Where to Put the JavaScript?

Head solution

- When in the head, JavaScripts in a page will be executed immediately while the page loads into the browser.
- To prevent the automatic execution, the script has to be inserted into a function.
- Functions can be put in the head section, thus they do not interfere with page content

```
<html>
<head>
<script type="text/javascript">
function message()
{
alert("This alert box was called
with the onload event");
}
</script>
</head>
<body onload="message()">
</body>
</html>
```

Where to Put the JavaScript?

Body Solution (not recommended)

- If you don't want your script to be placed inside a function, or if your script should write page content, **it should be placed in the body section**
- You can place an unlimited number of scripts in your document, so you can have scripts in both the body and the head section.

```
<html>  
<head>  
</head>
```

```
<body>  
<script type="text/javascript">  
document.write("This message  
is written by JavaScript");  
</script>  
</body>  
</html>
```

Note: we could have problems whether different scripts depend on each other

Where to Put the JavaScript?

External File

- If you want to run the same JavaScript on several pages, without having to write the same script on every page, you can write a JavaScript in **an external file**.
- Save the external JavaScript file with a **.js file extension**.
- Note: The external script **cannot contain any HTML tags** (in particular, `<script>` `</script>` tags!)
- To use the external script, point to the .js file in the "src" attribute of the `<script>` tag:

```
<html>
<head>
  <script type="text/javascript"
    src="xxx.js"> </script>
</head>
<body>
</body>
</html>
```



The default scripting language

- Authors should specify the default scripting language for all scripts in a document by including the following meta declaration in the head:

```
<meta http-equiv="Content-Script-Type" content="type">
```

where "type" is a content type naming the scripting language. Examples of values include "text/tcl", "text/javascript", "text/vbscript".

- The tag script allows using scripting languages different from the default.



The noscript element

- The **noscript element** allows authors to provide **alternate content when a script is not executed**.
- Its content should only be rendered if:
 - The user agent is configured not to evaluate scripts.
 - The user agent does not support a scripting language invoked by a script element earlier in the document.
- Example:

```
<noscript> <meta http-equiv="Refresh" content="2";  
url="paginasenzaJavaScript.html"> </noscript>
```
- If the noscript element is rendered, the user is re-addressed after 2 seconds to the url specified in the noscript element
- **Use both comments and the noscript element**



When is a script executed?

- Global code (not in the body of functions):
 - is executed when it is met during the rendering of the page. The global code can be in the HTML page or in an external file;
- Code in functions
 - is executed only if the function is called
- Event onLoad
 - The code corresponding to the event is executed when the page is loaded on the browser and after the execution of the code external to each function



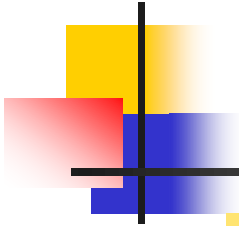
When is a script executed?

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="utf-8">  
  <title>Execution order</title>  
  <script type="text/javascript">  
  <!-- HTML comment
```



When is a script executed?

```
window.alert("Not in function");  
document.write("<h1> This is a heading. </h1>");  
document.write("<p>This is a paragraph.</p>");  
document.write("<p>This is another paragraph.</p>");  
function loading() { window.alert("onLoad event");} -->  
</script>  
</head>  
<body onLoad="loading()" >  
<script type="text/javascript">  
  window.alert("Code in Body");  
</script>  
</body>  
</html>
```



JavaScript Statements



JavaScript Code

- JavaScript code is a sequence of Javascript statements
- A JavaScript statement is a command to a browser.
`document.write("Hello Dolly");`
- The **semicolon is optional** (according to the JavaScript standard), and the browser is supposed to interpret the end of the line as the end of the statement.
 - Note: Using semicolons makes it possible to write multiple statements on one line.
- JavaScript is case sensitive



JavaScript Statements

- Statements are executed by the browser in the sequence they are written.

Example

```
<script type="text/javascript">
<!-- document.write("<h1>Introduction</h1>");
      document.write("<p>In this course we will introduce <a
href=\"http://devedge-
temp.mozilla.org/central/javascript/index_en.html\">
      Javascript</a></p>");
      document.write("<h2>Javascript Statements</h2>");
      document.write("<p>A Javascript statement is ... </p>"); -->
</script>
</head>
<body><p>Example</p></body>
```





JavaScript Blocks

- JavaScript statements can be grouped together in blocks.
 - Blocks start with a left curly bracket { and ends with a right curly bracket }.
 - The purpose of a block is to execute the sequence of statements together.

Example

```
<script type="text/javascript">
{ document.write("<h1>This is a heading</h1>");
  document.write("<p>This is a paragraph.</p>");
  document.write("<p>This is another paragraph.</p>");
}
</script>
```



JavaScript Comments

- **Comments** can be added to explain the JavaScript, or to make the code more readable.
 - Single line comments start with `//`.
 - Multi line comments start with `/*` and end with `*/`.

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

```
/*
```

The code below will write one heading and two paragraphs

```
*/
```

```
document.write("<h1>This is a heading</h1>");
```

```
// Write a heading
```

```
document.write("<p>This is a paragraph.</p>"); //paragraph
```

```
document.write("<p>This is another paragraph.</p>");
```

```
</script>
```



JavaScript Variables

- **Rules** for JavaScript variable names:
 - Variable names are **case sensitive** (y and Y are two different variables)
 - The first character in the name must be a letter (a-z or A-Z) or an underscore (_).
 - The rest of the name can be made up of letters (a-z or A-Z), numbers (0-9), or underscores (_).
 - Names should describe what variables are.
- **The type of the variable is not specified**



JavaScript Variables

- JavaScript allows declaring variables by simply using them
- Anyway, declaring variables helps to ensure that programs are well organized and helps to keep track of the scope of variables
- To declare JavaScript variables you can use the var statement

```
var x;
```

```
var carname;
```

After the declaration, the variables are empty (they have no value yet)



JavaScript Variables

- Variables can be initialized

```
var x=5;
```

```
var carname="Volvo";
```

- If you assign values to variables that have not been declared yet, the variables will automatically be declared.
- If you **redeclare** a JavaScript variable, it will not lose its original value.

```
var x=5;
```

```
var x;
```

- NOTE: the variable x will still have the value of 5. The value of x is not reset when you redeclare it.



Loosely typed

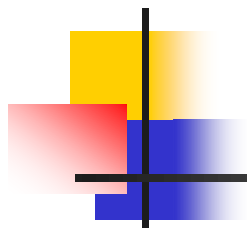
- JavaScript is what is called a loosely typed programming language:
 - the type of a variable is not defined when a variable is created and can, at times, change based on the context.

```
var text1 = "19";
```

```
var num1 = 96;
```

```
num1 = text1 + num1;
```

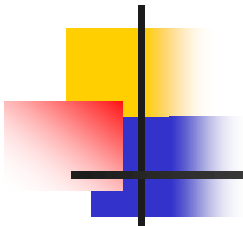
The variable num1 contains "1996".



Types of Values

- JavaScript recognizes the following types of values:
 - **Numeric**
 - **String**
 - **Boolean**
 - **Null** – a special keyword denoting a null value; null is also a primitive value
 - **Undefined** - a top-level property whose value is undefined;
 - undefined is also a primitive value (NaN in numeric contexts).





Numeric Values

■ Integer

NUMBER SYSTEM	NOTATION
Decimal (base 10)	A normal integer without a leading 0 (zero) (ie, 752)
Octal (base 8)	An integer with a leading 0 (zero) (ie, 056)
Hexadecimal (base 16)	An integer with a leading 0x or 0X (ie, 0x5F or 0XC72)

■ Floating Point Values

- 2.3e-3
- 2.3E-3





String Values

■ String

contains zero or more characters enclosed in single or double quotes

- NOTE: the empty string is distinct from the null value
- NOTE: Strings are different from other data types. Strings are actually Objects.
- The backslash (\) is used to insert apostrophes, new lines, quotes, and other special characters into a text string

Code	Outputs
\'	single quote
\"	double quote
\&	ampersand
\\	backslash



String type

Escape characters

	Character	Description
<code>\n</code>	new line	
<code>\t</code>	tab	
<code>\r</code>	carriage return	
<code>\f</code>	form feed	
<code>\b</code>	backspace	

NOTE: the escape characters only work in the following situations:

- within `<pre>` tags
- `alert()`, `confirm()` and `prompt()`
- within `<textarea>` tags



Boolean and null Values

- **boolean**
 - **Note:** Values of 1 and 0 are not considered Boolean values in JavaScript
- **null Value**
 - Represents Nothing
- **NaN** – Not a Number (returned by some functions like `parseInt()` and `parseFloat()`)



Variable Scope

```
<script type="text/javascript">
<!-- HTML comment
var cc = 0 ;      //global
var dd = scr();   // global
document.writeln("global: " + cc); // print value of cc
document.writeln("local: " + dd);  // print value of dd
function scr() {
    var cc = 3;    //local variable hides the global variable cc
    // without var, it would be an assignment to global variable cc
    return cc;
}
-->
</script>
</head>
```



JavaScript Operators

Arithmetic Operators

- Let us assume that $y=5$

Operator	Description	Example	Result
+	Addition	$x = y + 2$	$x = 7$
-	Subtraction	$x = y - 2$	$x = 3$
*	Multiplication	$x = y * 2$	$x = 10$
/	Division	$x = y / 2$	$x = 2.5$
%	Modulus (division remainder)	$x = y \% 2$	$x = 1$
++	Increment	$x = ++y$ $x = y++$	$x = 6$ $x = 5$
--	Decrement	$x = --y$ $x = y--$	$x = 4$ $x = 5$

JavaScript Operators

Assignment Operators

Operator	Example	Same As	Result
=	$x = y$		$x = 5$
+=	$x += y$	$x = x + y$	$x = 10$
-=	$x -= y$	$x = x - y$	$x = 5$
*=	$x *= y$	$x = x * y$	$x = 25$
/=	$x /= y$	$x = x / y$	$x = 5$
%=	$x \% = y$	$x = x \% y$	$x = 0$

JavaScript Operators

The + Operator used on Strings

- The + operator can also be used to add string variables or text values together

```
txt1="This is a very";  
txt2="nice day";  
txt3=txt1+" "+txt2;
```

- Note: if you add a number and a string, the result will be a string

```
<script type="text/javascript">  
x="5"+"5";  
document.write(x);  
document.write("<br>");           //55  
x=5+"5";  
document.write(x);  
document.write("<br>");           //55  
</script>
```


JavaScript Operators

Comparison Operators

- Given $x=5$, the table below explains the comparison operators

Operator	Description	Example
<code>==</code>	is equal to	<code>x==8</code> is false <code>x=='5'</code> is true
<code>===</code>	is exactly equal to (value and type)	<code>x===5</code> is true <code>x==='5'</code> is false
<code>!=</code>	is not equal (it attempts conversion)	<code>x!=8</code> is true <code>x!=5</code> is false
<code>!==</code>	is not equal and/or not of the same type	<code>x!== '5'</code> is true
<code>></code>	is greater than	<code>x>8</code> is false
<code><</code>	is less than	<code>x<8</code> is true
<code>>=</code>	is greater than or equal to	<code>x>=8</code> is false
<code><=</code>	is less than or equal to	<code>x<=8</code> is true

JavaScript Operators

Comparison Operators

- Comparison operators
 - If either or both values are NaN, then they are not equal.
 - Objects, arrays, and functions are compared by reference. This means that two variables are equal only if they refer to the same object.
 - If both are null, or both undefined, they are equal.
 - If one value is null and one undefined, they are equal.
- Two separate arrays are never equal by the definition of the == operator, even if they contain identical elements.

JavaScript Operators

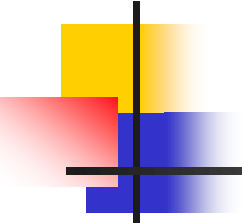
Logical Operators

- Given $x=6$ and $y=3$, the table below explains the logical operators

Operator	Description	Example
&&	and	$(x < 10 \ \&\& \ y > 1)$ is true
	or	$(x == 5 \ \ y == 5)$ is false
!	not	$!(x == y)$ is true

JavaScript Operators

Bitwise Operators

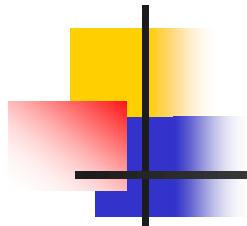


Operator	Description	Example
&	and	a & b
	or	a b
!	xor	a ^ b
~	not	~a
<<	Left shift	a<>	Sign-propagating right shift	a>>b
>>>	Zero-fill right shift	a>>>b



JavaScript Operators

- If the types of the two values differ, attempt to convert them into the same type so they can be compared:
 - If one value is a **number** and the other is a **string**
 - convert the string to a number and try the comparison again, using the converted value.
 - If **either value is true**
 - convert it to 1 and try the comparison again.
 - If **either value is false**
 - convert it to 0 and try the comparison again.
 - If one value is an **object** and the other is a **number or string**
 - convert the object to a primitive value by either its `toString()` method or its `valueOf()` method. Native JavaScript classes attempt `valueOf()` conversions before `toString()` conversion.
 - Any other combinations of types are not equal.



JavaScript Operators

- Conditional Operator

(condition) ? val1 : val2

Example:

```
var username = prompt("Please enter your name", "");  
var greeting = "Hello ";  
greeting += ((username != null) ? username : "guy");
```



JavaScript Operators

typeof operator

- **typeof operator**

Two ways:

1. `typeof operand`
2. `typeof (operand)`

- The **typeof operator** returns a string indicating the type of the operand. The parentheses are optional.

```
var num1 = 3; var num2 = 3.0;  
var bool=true; var shape="round";  
typeof num1;      // returns number  
typeof num2;      // returns number  
typeof bool;       // returns boolean  
typeof shape;     // returns string
```

JavaScript Operators

void operator

- void operator

Two ways:

1. void (expression)
 2. void expression
- The void operator specifies a JavaScript expression to be evaluated without returning a value. The parentheses surrounding the expression are optional, but it is good style to use them.
 - The following code creates a hypertext link that submits a form when the user clicks it.

```
<a href="javascript:void(document.form.submit())">
```

```
Click here to submit</a>
```


Conditional Statements

if statement



```
if (condition)
```

```
{
```

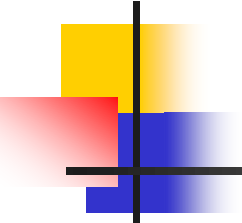
```
code to be executed if condition is true
```

```
}
```

NOTE: “if” is written in lowercase letters. Using uppercase letters (IF) will generate a JavaScript error!

Conditional Statements

if ... else statement



```
if (condition)
{
    code to be executed if condition is true
}
else
{
    code to be executed if condition is not true
}
```

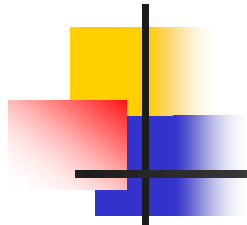
Conditional Statements

if ... else if ... else statement

```
if (condition1)
{
    code to be executed if condition1 is true
}
else if (condition2)
{
    code to be executed if condition2 is true
}
...
else
{
    code to be executed if condition1 and condition2 are not true
}
```

Conditional Statements

if ... else if ... else statement



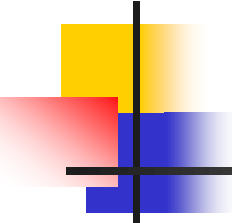
```
<script type="text/javascript">
var d = new Date();
var time = d.getHours();
if (time<12)
{
document.write("<em>Good
  morning</em>");
}
else if (time>=12 && time<17)
{
document.write("<em>Good
  afternoon</em>");
}
```

```
else if (time>=17 && time<20)
{
document.write("<em>Good
  evening</em>");
}
else
{
document.write("<em>Good
  night!</em>");
}
</script>
```



Conditional Statements

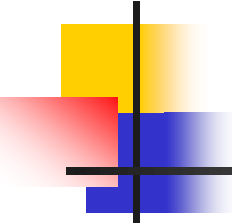
switch statement



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

Conditional Statements

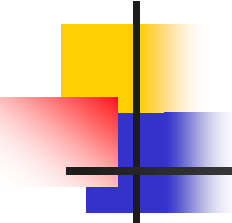
switch statement



```
<head>
<meta charset="utf-8">
<title>Example</title>
<style type="text/css">
p {color: white; background-color: grey; }
p.sat {color: red; background-color: black; }
p.sun {color: green; background-color: red; }
</style>
<script type="text/javascript">
<!-- HTML comment
var d=new Date();
theDay=d.getDay();
```

Conditional Statements

switch statement

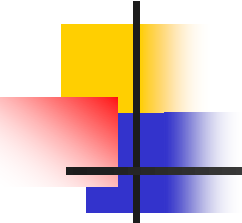


```
switch (theDay)
{ case 6:
    document.write("<p class='sat'>Super Saturday</p>");
    break;
  case 0:
    document.write("<p class='sun'>Sleepy Sunday</p>");
    break;
  default:
    document.write("<p>I'm looking forward to this weekend!</p>");
  }
-->
</script>
</head>
```



Loop statements

for



```
for (var=startvalue;var OP endvalue;var=var+increment)
{
code to be executed
}
```

where **OP** is any comparison operator.

Example

```
<script type="text/javascript">
var i=0;
for (i=0;i<=5;i++)
    { document.write("<p>The number is " + i + "</p>");}
</script>
```



Loop statements

while

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

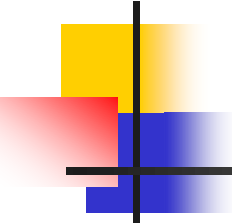
where **OP** is any comparison operator.

Example

```
<script type="text/javascript">
var i=0;
while (i<=5)
    {document.write("<p>The number is " + i++ + "</p>");}
</script>
```

Loop statements

do...while



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

where **OP** is any comparison operator.

Example

```
<script type="text/javascript">
var i=0;
do
{document.write("<p>The number is " + i++ + "</p>");}
while (i<=5)
</script>
```



Break statement

- The break statement will break the loop and continue executing the code that follows after the loop (if any).

```
<script type="text/javascript">  
var i=0;  
while (i<=5)  
  { document.write("<p>The number is " + i++ + "</p>");  
    if (i==4) break;  
  }  
</script>  
</head>
```





Continue statement

- The continue statement will break the current loop and continue with the next value

```
<script type="text/javascript">  
var i=0;  
for (i=0;i<=10;i++)  
    { if (i%4) continue;  
      document.write("<p>The number is " + i + "</p>");  
    }  
</script>
```



Label statement

- A **label** provides a statement with an identifier that lets you refer to it elsewhere in your program.
label : statement
- The value of label may be any JavaScript identifier that is not a reserved word.
- On using label with break and continue
 - **break** [*label*] - terminates the specified enclosing label statement
 - **continue** [*label*] - restarts a label statement or continues execution of a labelled loop with the next iteration



continue statement (example)

```
var i=0, j=8;
checki : while (i<8) {
    document.write("<p>i = " + i + "</p>");
    checkj : while (j>0) {
        j--;i++;
        if ((j%3)==0) continue checki;
        if ((j%2)==0) continue checkj;
        document.write("<p>j = " + j + " is odd.</p>");
    }
}
```

Loop statements

for .. in

- The for...in statement loops through the elements of an array or through the properties of an object

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

- Note: The code in the body of the for...in loop is executed once for each element/property.
- Note: The variable argument can be a named variable, an array element (not an index), or a property (not the value of the property, but the name) of an object.

Loop statements

for .. in

```
<script type="text/javascript">  
<!-- HTML comment  
var x;  
var mycars = new Array();  
mycars[0] = "Saab";  
mycars[1] = "Volvo";  
mycars[2] = "BMW";  
  
for (x in mycars)  
    document.write("<p>" + x + ": " + mycars[x] + "</p>");  
-->  
</script>
```





with statement

- The **with** statement establishes the default object for a set of statements.
- JavaScript looks up any unqualified names within the set of statements to determine if the names are properties of the default object. If an unqualified name matches a property, then the property is used in the statement; otherwise, a local or global variable is used.

- A with statement looks as follows:

```
with (object){  
  statements  
}
```

```
var a, x, y;  
var r=10;  
with (Math) {  
  a = PI * r * r;  
  x = r * cos(PI);  
  y = r * sin(PI/2);  
}
```



Functions

```
function name(var1,var2,...,varX)
{ var x;
  some code
  return x;
}
```

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



Functions

- All parameters are passed to functions by **value**; the value is passed to the function, but if the function changes the value of the parameter, this change is not reflected globally or in the calling function.
- Objects are passed by reference: if **the function changes the object's properties**, that change is visible outside the function.
- A function **can even be recursive**, that is, it can call itself.



Functions

```
<script type="text/javascript">  
function fun(s) {      // function definition  
    var a = 10;  
    var c = 10 * s;  
    document.write("<p>This is the result: " + c + "</p>");  
}  
</script>  
</head>
```

```
<body onLoad="fun(10)">  
<!-- call to the function fun -->  
<script type="text/javascript">  
    window.alert("Code in Body");  
</script>
```





Functions

- The arguments of a function are maintained in the array **"arguments"**.
- Within a function, you can address the parameters passed to it by:

arguments[i]

functionName.arguments[i]

where i is the ordinal number of the argument, starting at zero.

arguments[0] -> the first argument passed to a function.

- The total number of arguments is indicated by **arguments.length**.



Functions

- Using the arguments array, you can call a function with more arguments than it is formally declared to accept.
 - This is often useful if you do not know in advance how many arguments will be passed to the function.
 - You can use `arguments.length` to determine the number of arguments actually passed to the function, and then treat each argument using the arguments array.
- No check on the order of the parameters



Functions

```
<script type="text/javascript">  
<!-- HTML comment  
function myConcat(separator) {  
  result=""; // initialize list  
  // iterate through arguments  
  for (var i=1; i<arguments.length; i++) {  
    result += arguments[i] + separator;  
  }  
  result += "<br>";  
  return result;  
}  
-->  
</script>  
</head>
```



Functions

```
<body>
<script type="text/javascript">
<!-- HTML comment
// returns "red, orange, blue, "
document.write(myConcat(", ", "red", "orange", "blue"));
// returns "elephant; giraffe; lion; cheetah;"
document.write(myConcat(";", "elephant", "giraffe", "lion", "cheetah"));
// returns "sage. basil. oregano. pepper. parsley. "
document.write(myConcat(".", "sage", "basil", "oregano", "pepper",
    "parsley"));
-->
</script>
```





The return statement

- The return statement is used to specify the value that is returned from the function.

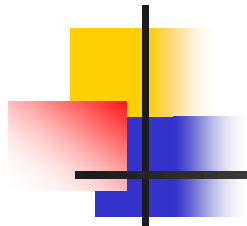
```
<head>
<script type="text/javascript">
function product(a,b)
{ return a*b; }
</script>
</head>
<body>
<script type="text/javascript">
document.write(product(4,3));
</script>
```





Predefined functions

- JavaScript has several top-level predefined functions:
 - **eval(string)** - Evaluates a string and executes it as if it was script code
`eval("x=10;y=20;document.write(x*y)"); //200`
 - **isFinite** - Determines whether a value is a finite, legal number
`document.write(isFinite(123)+ "
"); //true`
`document.write(isFinite("2005/12/12")+ "
"); //false`
 - **isNaN** -The isNaN() function determines whether a value is an illegal number (Not-a-Number).
This function returns true if the value is NaN, and false if not.
 - `document.write(isNaN(123)+ "
"); //false`
 - `document.write(isNaN("2005/12/12")+ "
"); //true`



Predefined functions

- **parseInt(string,radix)**

Parses a string and returns an integer of the specified radix (base).

radix - a number (from 2 to 36) that represents the numeral system to be used

- **parseFloat(string)**

Parse a string and returns a float number.

If the first character cannot be converted to a number, the two functions return NaN.

- **number(object)** and **string(object)**

Converts the object argument to a number or to a string that represent the object's value.

If the value cannot be converted to a legal number, NaN is returned.

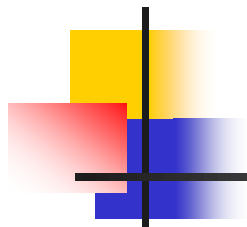




Predefined functions

```
<body>
<script type="text/javascript">
<!-- HTML comment
eval("x=10;y=20;document.write(x*y)");
document.write("<br>" + isFinite(123)+ "<br>");
document.write(isFinite("2005/12/12")+ "<br>");
document.write(isNaN(123)+ "<br>");
document.write(isNaN("2005/12/12")+ "<br>");
document.write(parseInt("His age is 40 years")+ "<br>");
document.write(parseInt("40 years")+ "<br>");
-->
</script>
</body>
```





Predefined functions

- **escape(string)** and **unescape(string)**

The `escape()` function encodes a string. This function makes a string portable, so it can be transmitted across any network to any computer that supports ASCII characters. This function encodes special characters, with the exception of: * @ - _ + . /

The `unescape()` function decodes an encoded string.

```
<script type="text/javascript">  
<!-- HTML comment  
str="Need tips? Visit W3Schools!";  
str_esc=escape(str);  
document.write(str_esc + "<br>");  
document.write(unescape(str_esc));  
-->  
</script>
```



document.write() and document.writeln() methods

- **document.write(exp1,exp2,exp3,...)**

The write() method writes HTML expressions or JavaScript code to a document.

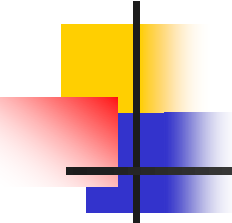
Multiple arguments can be listed and they will be appended to the document in order of occurrence

- **document.writeln(exp1,exp2,exp3,...)**

The writeln() method is identical to the write() method, with the addition of writing a newline character after each statement.

Since HTML ignores the newline characters, the effects of the methods on the HTML pages are equal except when used within `<pre>` tags

The document.write() and document.writeln() methods



```
<body>
<pre>
<script type="text/javascript">
  document.write("Hello World!");
  document.write("Have a nice day!");
</script>
</pre>
<pre>
<script type="text/javascript">
  document.writeln("Hello World!");
  document.writeln("Have a nice day!");
</script>
</pre>
</body>
```



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.

```
alert("sometext");
```


Popup Boxes

Alert Box

```
<script type="text/javascript">
function show_alert()
{ alert("I am an alert box!");}
</script>
</head>
<body>
<p><input type="button" onclick="show_alert()" value="Show alert
  box">
</p>
</body>
```



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**.

```
confirm("sometext");
```

Popup Boxes

Confirm Box

```
<script type="text/javascript">  
function show_confirm()  
{ var r=confirm("Press a button!");  
  if (r==true)  
    { alert("You pressed OK!"); }  
  else  
    { alert("You pressed Cancel!"); }  
}  
</script>
```



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**.

```
prompt("sometext","defaultvalue");
```

Popup Boxes

Prompt Box

```
<script type="text/javascript">
function show_prompt()
{ var name=prompt("Please enter your name","Harry Potter");
  if (name!=null && name!="")
    { document.write("Hello " + name + "! How are you today?"); }
}
</script>
</head>
<body>
<input type="button" onclick="show_prompt()" value="Show
prompt box">
</body>
```



Catching Errors

try...catch statement

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

- 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.
- *err* is initialized with the exception object
- *expression* is a test expression

Catching Errors

try...catch statement

```
try { myroutine();      // may throw three exceptions
}
catch (e if e instanceof TypeError) {
    // statements to handle TypeError exceptions
}
catch (e if e instanceof RangeError) {
    // statements to handle RangeError exceptions
}
catch (e if e instanceof EvalError) {
    // statements to handle EvalError exceptions
}
catch (e){
    // statements to handle any unspecified exceptions
    logMyErrors(e) // pass exception object to error handler
}
```

Catching Errors

try...catch statement

```
<script type="text/javascript">
function message()
{ try {  addlert("Welcome guest!"); }
  catch(err)
  {   var txt="There was an error on this page.\n\n";
      txt+="Click OK to continue viewing this page,\n";
      txt+="or Cancel to return to the home page.\n\n";
      if(!confirm(txt))
      {   document.location.href=
"http://devedge-temp.mozilla.org/central/javascript/index_en.html";
      }
  }
}
</script> </head>
```


Catching Errors

try...catch statement



```
<body>
```

```
<div>
```

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

```
</div>
```

```
</body>
```



Catching Errors

Throw statement

- The **throw statement** allows you to create an exception.

`throw(exception)`

- The exception can be a string, integer, Boolean or an object.

`<body>`

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

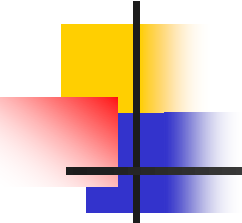
`var x=prompt("Enter a number between 0 and 10:","");`

`try`

```
{  if(x>10)
    {  throw "Err1";  }
  else if(x<0)
    {  throw "Err2";  }
  else if(isNaN(x))
    {  throw "Err3";  }
}
```

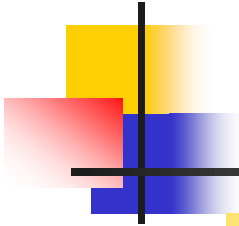
Catching Errors

Throw statement



```
catch(er)
{ if(er=="Err1")
  { alert("Error! The value is too high"); }
  if(er=="Err2")
  { alert("Error! The value is too low"); }
  if(er=="Err3")
  { alert("Error! The value is not a number"); }
}
</script>
</body>
```





JavaScript Objects





JavaScript Objects

- No inheritance
- No private properties and methods
- Both the object name and property name are case sensitive.
- You define a property by assigning it a value.
 - For example, suppose there is an object named myCar (for now, just assume the object already exists). You can give it properties named make, model, and year as follows:

```
myCar.make = "Ford";  
myCar.model = "Mustang";  
myCar.year = 1969;
```



JavaScript Objects

- Two ways for accessing the properties of an object:
 - `objectName.propertyName`
 - `objectName["propertyName"]`
- The second method is inherited from the associative arrays.
- Actually, associative arrays and objects in JavaScript are different interfaces of the same data structure
- **Associative arrays**: each index element is associated with a string value



JavaScript Objects

- Associative array notation (example)

```
function show_props(obj, obj_name) {  
    var result = "";  
    for (var i in obj) // i assumes as values all the property names  
        result += obj_name + "." + i + " = " + obj[i] + "\n";  
    return result  
}
```

The function call `show_props(myCar, "myCar")` returns

`myCar.make = Ford`
`myCar.model = Mustang`
`myCar.year = 1967`

JavaScript Objects

Creating new objects

- Two ways for creating objects
 - use an **object initializer**
 - create a **constructor function** and then instantiate an object using that function and the new operator.
- The first way is useful when you need to create a unique instance of the object; otherwise use the second way.

JavaScript Objects

Object_INITIALIZER

- Object initializer

`objectName = {property1:value1,..., propertyN:valueN}`

objectName: name of the new object

propertyI: identifier (either a name, a number, or a string literal),

valueI: expression whose value is assigned to the propertyI.

The **property of an object can be an object** in its turn.

The `objectName` and assignment is optional. If you do not need to refer to this object elsewhere, you do not need to assign it to a variable.

Note: **new properties can be added dynamically**

`objectName.newproperty=value`

JavaScript Objects

Object_INITIALIZER

- **Object initializer** (example)

```
guy = { name: "Paul",  
        age: 40,  
        country: "Italy",  
        auto: { trademark: "Ferrari", colour: "rosso" } }
```

...

```
<pre>  
<script type="text/javascript">  
for (var a in guy)  
    if (typeof(guy[a])=="object")  
        for (var i in guy[a])  
            document.writeln(a + '.' + i + ':' + guy[a][i]);  
    else document.writeln(a + ':' + guy[a]);  
</script>  
</pre>
```



JavaScript Objects

Constructor Function

- Constructor Function

1. Define the object type by writing a constructor function.
2. Create an instance of the object with new.

To define an object type, create a function for the object type that specifies its name, properties, and methods.

```
function Car(make, model, year) {  
  this.make = make  
  this.model = model  
  this.year = year  
}
```

this keyword!

```
mycar = new Car("Eagle", "Talon TSi", 1993)  
kenscar = new Car("Nissan", "300ZX", 1992)  
vpgscar = new Car("Mazda", "Miata", 1990)
```



JavaScript Objects

Constructor Function

- **Constructor Function**

An object can **have a property that is itself another object.**

```
function Person(name, age, sex) {  
  this.name = name  
  this.age = age  
  this.sex = sex  
}
```

```
function Car(make, model, year, owner) {  
  this.make = make  
  this.model = model  
  this.year = year  
  this.owner = owner  
}
```

```
owner = new Person("Frankie Black",45,"M")  
mycar = new Car("Eagle", "Talon TSi", 1993, owner)
```



JavaScript Operators

new operator

- new operator

You can use the new operator to create an instance of a user-defined object type or of one of the predefined object types Array, Boolean, Date, Function, Image, Number, Object, Option, RegExp, or String.

```
objectName = new objectType (param1 [,param2]  
...[,paramN] );
```

JavaScript Objects

Adding a new property

- Properties can be added to an object at run time.
- To add a property to a specific object you have to assign a value to the object
`car1.enginepower = 100`
- Note: only the car1 object will have the property enginepower

JavaScript Object Types

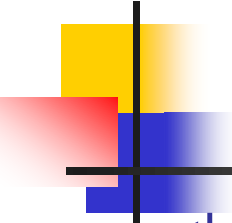
Adding a new property

- You can add a property to a previously defined object type by using the prototype property.
- This defines a property that is shared by all objects of the specified type, rather than by just one instance of the object.
- The following code adds a color property to all objects of type car, and then assigns a value to the color property of the object car1.

```
Car.prototype.color=null  
car1.color="black"
```

JavaScript Object Types

Adding a new property



```
<head>
<meta http-equiv="Content-type" content="text/html; charset=ISO-
8859-1">
<meta http-equiv="Content-Script-Type" content="text/javascript">
<title>Example</title>
<script type="text/javascript"><!--
function displayCar(car)
{ for (var a in car)
    if (typeof(car[a])=="object")
        for (var i in car[a])
            document.writeln(a + '.' + i + ':' + car[a][i]);
    else document.writeln(a+':'+car[a]);
    document.writeln();
}
</script></head>
```


JavaScript Object Types

Adding a new property

```
<script type="text/javascript"><!-- HTML comment
owner = new Person("Frankie Black",45,"M")
mycar1 = new Car("Eagle", "Talon TSi", 1993, owner)
displayCar(mycar1)
Car.prototype.color="red"
mycar2 = new Car("Nissan", "300ZX", 1992,owner)
mycar3 = new Car("Mazda", "Miata", 1990,owner)
mycar3.color = "black"
displayCar(mycar1)
displayCar(mycar2)
displayCar(mycar3) -->
```

JavaScript Objects

Defining methods

- The following syntax associates a function with an existing object:

`object.methodname = function_name`

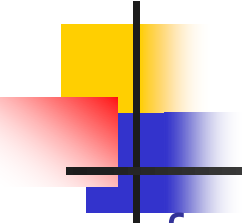
- You can then call the method in the context of the object as follows:

`object.methodname(params);`

- You can define methods for an object type by including a method definition in the object constructor function.

JavaScript Objects

Defining methods



```
function displayCar(){
  for (var a in this)
    if (typeof(this[a])=="object")
      for (var i in this[a]) document.writeln(a + '.' + i + ':' + this[a][i]);
    else if (typeof(this[a])!="function") document.writeln(a+':'+this[a]);
  document.writeln();
}
```

```
function Car(make, model, year, owner) {
  this.make = make;
  this.model = model;
  this.year = year;
  this.owner = owner;
  this.displayCar = displayCar;
}
```

JavaScript Objects

Defining methods



```
<script type="text/javascript">
<!-- HTML comment
    owner = new Person("Frankie Black",45,"M")
    mycar1 = new Car("Eagle", "Talon TSi", 1993, owner)
    mycar1.displayCar();
    Car.prototype.color="red"
    mycar2 = new Car("Nissan", "300ZX", 1992,owner)
    mycar3 = new Car("Mazda", "Miata", 1990,owner)
    mycar3.color = "black"
    mycar1.displayCar();
    mycar2.displayCar();
    mycar3.displayCar();
-->
</script>
```



JavaScript Objects

Defining methods

Alternatively

```
function Car(make, model, year, owner) {  
    this.make = make;  
    this.model = model;  
    this.year = year;  
    this.owner = owner;  
}
```

```
Car.prototype.displayCar = function () {  
    for (var a in this)  
        if (typeof(this[a])=="object")  
            for (var i in this[a]) document.writeln(a + '.' + i + ':' + this[a][i]);  
        else if (typeof(this[a])!="function") document.writeln(a+':'+this[a]);  
    document.writeln();  
}
```



JavaScript Operators

delete operator

delete `objectName`
delete `objectName.property`
delete `objectName[index]`
delete `property` // legal only within a “with” statement

where `objectName` is the name of an object, `property` is an existing property, and `index` is an integer representing the location of an element in an array.

- If the `delete operator succeeds`, it sets the property or element to `undefined`.
- The delete operator returns true if the operation is possible; it returns false if the operation is not possible.
- You can use the delete operator to delete variables declared implicitly but not those declared with the var statement.

JavaScript Operators

delete operator

- delete operator (example)

```
x=42;
```

```
var y= 43;
```

```
myobj=new Number();
```

```
myobj.h=4; // create property h
```

```
delete x; // returns true (can delete if declared implicitly)
```

```
delete y; // returns false (cannot delete if declared with var)
```

```
delete Math.PI; // returns false (cannot delete predefined  
properties)
```

```
delete myobj.h; // returns true (can delete user-defined properties)
```

```
delete myobj; // returns true (delete user-defined object)
```

```
var myobj1=new Number();
```

```
window.alert(delete myobj1); //returns false (cannot delete if  
declared with var)
```

JavaScript Operators

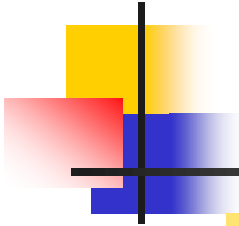
delete operator

- delete operator (array elements)

When you delete an array element, the array length is not affected. For example, if you delete `a[3]`, `a[4]` is still `a[4]` and `a[3]` is undefined.

When the delete operator removes an array element, that element is no longer in the array.

```
trees=new Array("redwood","bay","cedar","oak","maple");
delete trees[3];
if (3 in trees) {
  // this does not get executed
}
trees[3] = "oak";
if (3 in trees) {
  // this does get executed
}
```

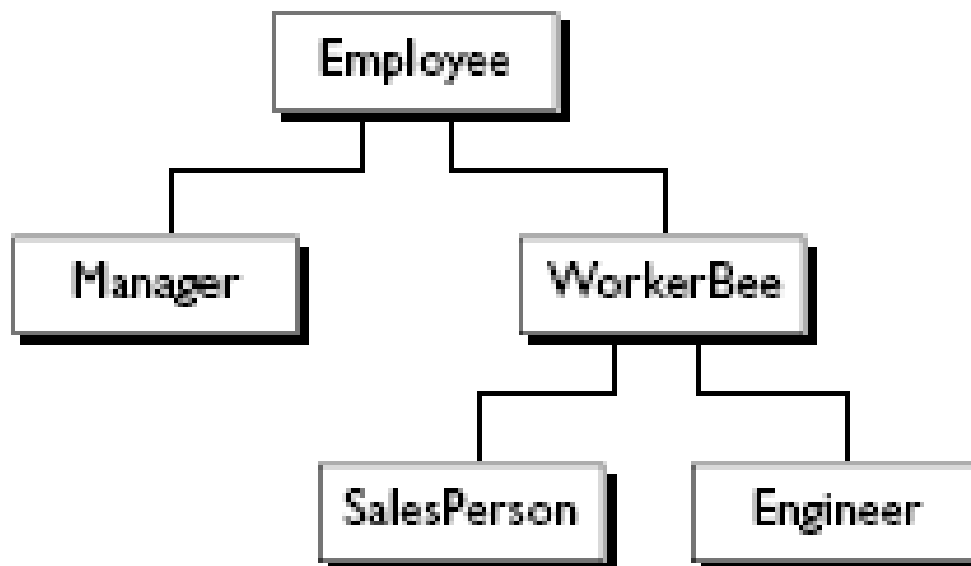
Inheritance



JavaScript Objects

Inheritance

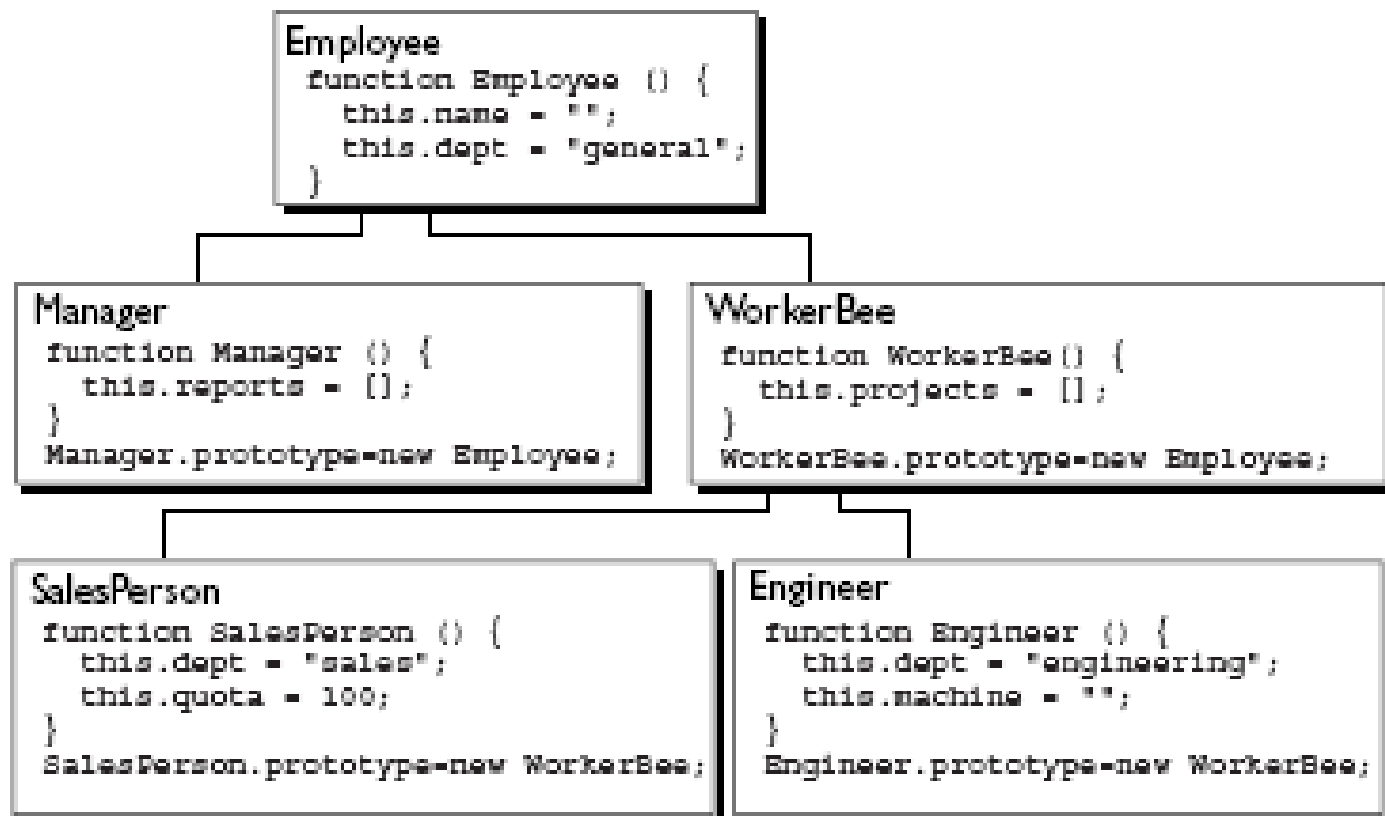
- JavaScript implements inheritance by allowing you to associate a prototypical object with any constructor function.



JavaScript Objects

Inheritance

- When you create a new Manager, it inherits the name and dept properties from the Employee object.




JavaScript Objects

Inheritance

- The **new** operator creates a new generic object and passes this new object as the value of the **this** keyword to the constructor function.
- When you ask for the value of a property, JavaScript first checks to see if the value exists in that object.
 - If it does, that value is returned.
 - If the value is not there locally, JavaScript **checks the prototype chain** (using the `__proto__` property).
 - If an object in the prototype chain has a value for the property, that value is returned.
 - If no such property is found, JavaScript says the object does not have the property.

JavaScript Objects Inheritance



```
function display(obj)
{ for (var a in obj)
  {    document.write(a);
    if (typeof(obj[a])=="object" && obj[a].length!=0)
    {    document.write("."); display(obj[a]);}
    else    document.writeln(' = ' + obj[a]);
  }
}

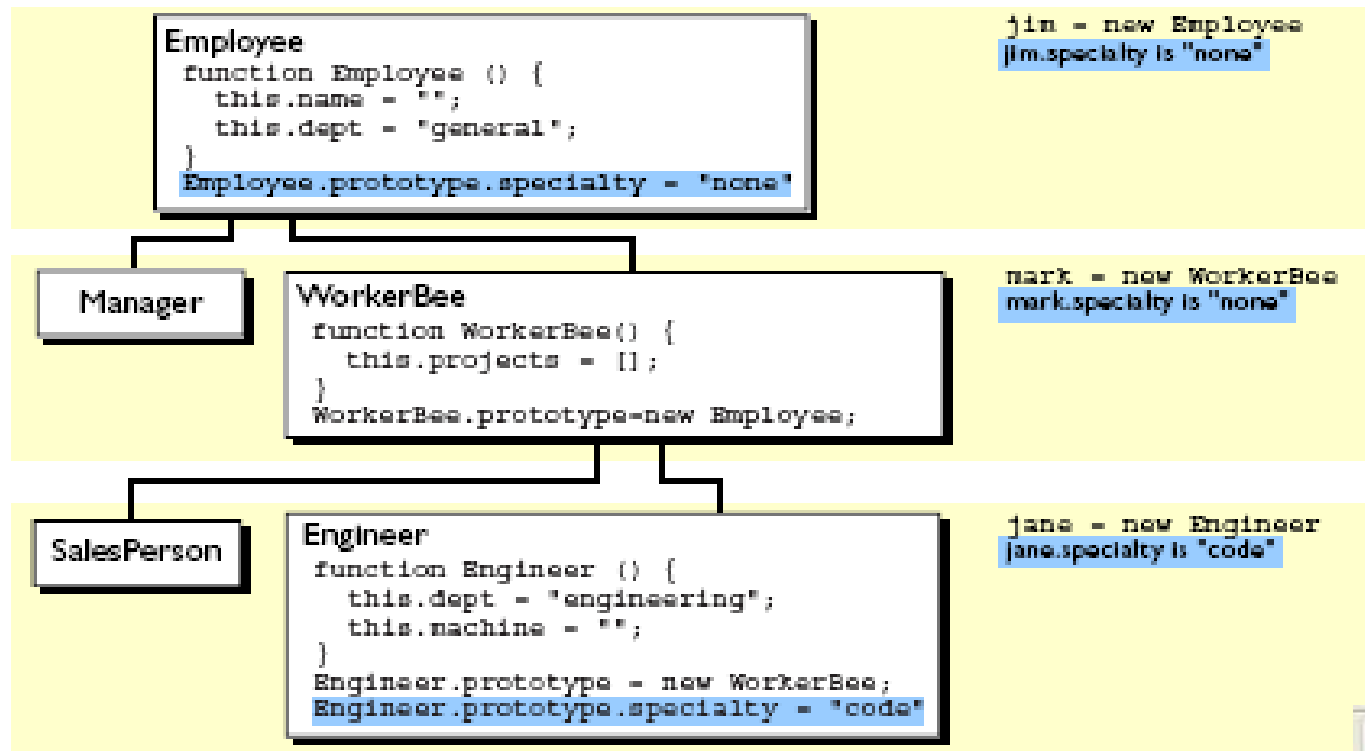
salesP = new SalesPerson;
display(salesP);
document.writeln();
eng = new Engineer;
display(eng);
```



JavaScript Objects

Inheritance – adding properties

- If you add a **new property to an object** that is being used as the **prototype for a constructor function**, you add that property to all objects that inherit properties from the prototype.



JavaScript Objects

Inheritance – adding properties

Constructors with arguments

Default values

```
Employee
function Employee (name, dept) {
  this.name = name || "";
  this.dept = dept || "general";
}
```

```
jim = new Employee("Jones, Jim", "marketing")
jim.name is "Jones, Jim"
jim.dept is "marketing"
```

Manager

WorkerBee

```
function WorkerBee(projs) {
  this.projects = projs || [];
}
WorkerBee.prototype = new Employee;
```

```
mark = new WorkerBee(["javascript"])
mark.name is ""
mark.dept is "general"
mark.projects is ["javascript"]
```

SalesPerson

Engineer

```
function Engineer(mach) {
  this.dept = "engineering";
  this.machine = mach || "";
}
Engineer.prototype = new WorkerBee;
```

```
jane = new Engineer("belau")
jane.name is ""
jane.dept is "engineering"
jane.projects is [ ]
jane.machine is "belau"
```

JavaScript Objects

Inheritance – adding properties

■ Constructors with arguments

Employee

```
function Employee (name, dept) {
  this.name = name || "";
  this.dept = dept || "general";
}
```

```
jim = new Employee("Jones, Jim", "marketing");
jim.name is "Jones,Jim"
jim.dept is "marketing"
```

Manager

WorkerBee

```
function WorkerBee(name, dept, proj) {
  this.base = Employee;
  this.base(name, dept);
  this.projects = proj || [];
}
WorkerBee.prototype = new Employee;
```

```
mark = new WorkerBee("Smith, Mark", "training",
  ["javascript"]);
mark.name is "Smith,Mark"
mark.dept is "training"
mark.projects is ["javascript"]
```

SalesPerson

Engineer

```
function Engineer (name, proj, mach) {
  this.base = WorkerBee;
  this.base(name, "engineering", proj);
  this.machine = mach || "";
}
Engineer.prototype = new WorkerBee;
```

```
jane = new Engineer ("Doe, Jane",
  ["navigator", "javascript"], "belau");
jane.name is "Doe, Jane"
jane.dept is "engineering"
jane.projects is ["navigator", "javascript"]
jane.machine is "belau"
```



JavaScript Objects

Inheritance – adding properties

- Note: The name of the base property is not special. You can use any legal property name; base is simply evocative of its purpose.
- Note: call the WorkerBee constructor from inside the Engineer constructor does not set up inheritance appropriately for Engineer objects.
 - Calling the WorkerBee constructor ensures that an Engineer object starts out with the properties specified in all constructor functions that are called. However, if you later add properties to the Employee or WorkerBee prototypes, those properties are not inherited by the Engineer object.

JavaScript Objects

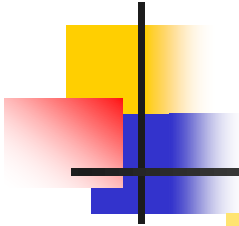
Inheritance – adding properties

- For example, assume you have the following statements:

```
function Engineer (name, projs, mach) {  
  this.base = WorkerBee; this.base(name, "engineering", projs);  
  this.machine = mach || "";}  
  
jane = new Engineer("Doe, Jane", ["navigator", "javascript"], "belau");  
Employee.prototype.specialty = "none";
```
- The jane object does not inherit the specialty property.

Engineer.prototype = new WorkerBee;
jane = new Engineer("Doe, Jane", ["navigator", "javascript"], "belau");
Employee.prototype.specialty = "none";

- **Now, the value is inherited**



Built-in Objects





Objects

- **Predefined built-in objects**

Array, Boolean, Date, Function, Math, Number, RegExp, and String

- **Predefined client-side objects**

- When you load a document in Navigator, it creates a number of JavaScript objects with property values based on the HTML in the document and other pertinent information.
- These objects exist in a hierarchy that reflects the structure of the HTML page itself.
- Each page has always the objects: **Navigator, Window, Document, Location, History.**

Built-in Objects

Array

- An array is an ordered set of values that you refer to with a name and an index.
- Array elements may belong to different types
- The Array object has methods for manipulating arrays in various ways, such as joining, reversing, and sorting them.
 - It has a property for determining the array length and other properties for use with regular expressions.

Built-in Objects

Array

■ Create an Array

Three forms

- `var myCars=new Array();` // (add an optional integer
`myCars[0]="Saab";` // argument to control array's size)
`myCars[1]="Volvo";`
`myCars[2]="BMW";`
- `var myCars=new Array("Saab","Volvo","BMW");` //condensed
- `var myCars=["Saab","Volvo","BMW"];` //literal array
 - It is also possible to avoid to specify all the values of the elements of the array
`var myCars=["Saab", ,"BMW"];`

Built-in Objects

Array

- Access an Array

```
document.write(myCars[0]);
```

- Modify values in an Array

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

- An array can dynamically be extended

```
myCars[6]="Fiat";
```

The size of the array is extended to 7. The elements with indexes 3, 4 and 5 are undefined.

Built-in Objects

Array

- Array properties

Property	Description
constructor	Returns the function that created the Array object's prototype
length	Sets or returns the number of elements in an array
prototype	Allows you to add properties and methods to an object

Built-in Objects

Array

Method	Description
concat()	Joins two or more arrays, and returns a copy of the joined arrays <code>array.concat(array2, array3, ..., arrayX)</code>
join()	Joins all elements of an array into a string <code>array.join(separator)</code> If the separator is omitted, the elements are separated with a comma
pop()	Removes the last element of an array, and returns that element <code>array.pop()</code>
push()	Adds new elements to the end of an array, and returns the new length <code>array.push(element1, element2, ..., elementX)</code>

Built-in Objects

Array

Method	Description
reverse()	Reverses the order of the elements in an array <code>array.reverse()</code>
shift()	Removes the first element of an array, and returns that element <code>array.shift()</code>
slice()	Selects a part of an array, and returns the new array <code>array.slice(start, end)</code> <i>start</i> Required. An integer that specifies where to start the selection. You can also use negative numbers to select from the end of an array <i>end</i> Optional. An integer that specifies where to end the selection. If omitted, <code>slice()</code> selects all elements from the start position and to the end of the array

Built-in Objects

Array

Method	Description
sort()	Sorts the elements of an array
	<i>array.sort(sortfunc)</i>
	<i>sortfunc</i> Optional. A function that defines the sort order Default: sorts the elements alphabetically and ascending. However, numbers will not be sorted correctly (40 comes before 5). To sort numbers, you must add a function that compare numbers
splice()	Adds/Removes elements from an array
	<i>array.splice(index,howmany,element1,.....,elementX)</i>
	<i>index</i> Required. An integer that specifies at what position to add/remove elements
	<i>howmany</i> Required. The number of elements to be removed. If set to 0, no elements will be removed
	<i>element1, ..., elementX</i> Optional. The new element(s) to be added to the array

Built-in Objects

Array

Method	Description
toString()	Converts an array to a string, and returns the result <i>array.toString()</i>
unshift()	Adds new elements to the beginning of an array, and returns the new length <i>array.unshift(element1,element2, ..., elementX)</i> <i>element1,element2, ..., elementX</i> Required. The element(s) to add to the beginning of the array
valueOf()	Returns the primitive values of an array <i>array.valueOf()</i>

Built-in Objects

Array

■ Sort numbers

```
<script type="text/javascript">
function sortNumber(a,b)
{
    return a - b;          //(numerically and ascending)
    //return b - a;  (numerically and descending)
}
var n = [10, 5, 40, 25, 100, 1];
document.write(n.sort(sortNumber));
</script>
```

Built-in Objects

Array

■ Example

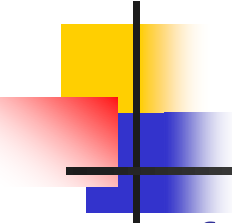
```
<script type="text/javascript"><!--  
var myvector= new Array("S","D","A","B");  
print(myvector,"(initialise)");
```

```
function add()  
{  myvector[myvector.length] = document.forms[0].text1.value;  
  print(myvector, "(add)");  
}
```

```
function concatenate()  
{  var vector = myvector.join("+");  
  document.writeln(vector+ "<br>(concatenate)");  
}
```

Built-in Objects

Array



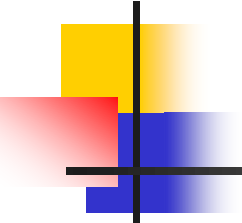
```
function invert()  
{ myvector.reverse();  
  print(myvector, "(invert)");  
}
```

```
function shift()  
{ myvector.shift();  
  print(myvector, "(translate)");  
}
```

```
function sort()  
{ myvector.sort();  
  print(myvector, "(sort)");  
}
```

Built-in Objects

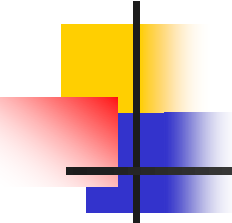
Array



```
function print(vector, comment)
{ for (var i = 0; i < vector.length; i++)
    document.writeln(vector[i]);
    document.writeln("<br>" + comment);
}
-->
</script>
</head>
```


Built-in Objects

Array



```
<body>
<form action="#">
<p>Add a new element: <input type="text" name="text1">
<input type="button" value="ADD" onclick="add()"><br>
<input type="button" value="JOIN('+)'
    onclick="concatenate()"><br>
<input type="button" value="INVERT" onclick="invert()"><br>
<input type="button" value="SHIFT" onclick="shift()"><br>
<input type="button" value="SORT" onclick="sort()"><br>
</p>
</form>
</body>
```



Built-in Objects

Boolean

- The Boolean object is a wrapper around the primitive Boolean data type.

`booleanObjectName = new Boolean(value)`

- If the Boolean object has no initial value or if it is 0, -0, null, "", false, undefined, or NaN, the object is set to false. Otherwise it is true (even with the string "false")!
- It is used to convert a non-boolean value into a boolean value.

Method	Description
<code>toString()</code>	Converts a Boolean value to a string, and returns the result
<code>valueOf()</code>	Returns the primitive value of a Boolean object

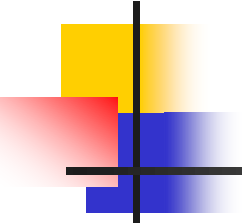
Built-in Objects

Date

- JavaScript stores dates as the number of milliseconds since January 1, 1970, 00:00:00
- Four forms to create a Date object
 - `new Date ()` // Date object with current date and time
 - `new Date (milliseconds)` // milliseconds since 1970/01/01
 - `new Date (dateString)` // A string representing a date in the following form: "Month day, year hours:minutes:seconds." For example, `Xmas95 = new Date("December 25, 1995 13:30:00")`. If you omit hours, minutes, or seconds, the value will be set to zero.
 - `new Date (year, month, day, hours, minutes, seconds, milliseconds)` // parameters are integer values
- Most parameters above are optional. Not specifying them, causes 0 to be passed in.

Built-in Objects

Date



Method	Description
<code>getDate()</code>	Returns the day of the month (from 1-31)
<code>getDay()</code>	Returns the day of the week (from 0-6)
<code>getFullYear()</code>	Returns the year (four digits)
<code>getHours()</code>	Returns the hour (from 0-23)
<code>getMilliseconds()</code>	Returns the milliseconds (from 0-999)
<code>getMinutes()</code>	Returns the minutes (from 0-59)
<code>getMonth()</code>	Returns the month (from 0-11)
<code>getSeconds()</code>	Returns the seconds (from 0-59)
<code>getTime()</code>	Returns the number of milliseconds since midnight Jan 1, 1970

Built-in Objects

Date

Method	Description
<code>getUTCDate()</code>	Returns the day of the month, according to universal time (from 1-31). UTC (Coordinated Universal Time) is an atomic timescale that approximates UT1.
<code>getUTCDay()</code>	Returns the day of the week, according to universal time (from 0-6)
<code>getUTCFullYear()</code>	Returns the year, according to universal time (four digits)
<code>getUTCHours()</code>	Returns the hour, according to universal time (from 0-23)
<code>getUTCMilliseconds()</code>	Returns the milliseconds, according to universal time (from 0-999)
<code>getUTCMinutes()</code>	Returns the minutes, according to universal time (from 0-59)
<code>getUTCMonth()</code>	Returns the month, according to universal time (from 0-11)
<code>getUTCSeconds()</code>	Returns the seconds, according to universal time (from 0-59)

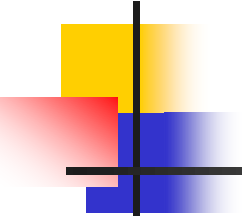
Built-in Objects

Date

Method	Description
parse()	Parses a date string and returns the number of milliseconds since midnight of January 1, 1970
	Date.parse(datestring)
	Datestring Required. A string representing a date
setDate()	Sets the day of the month (from 1-31)
setFullYear()	Sets the year (four digits)
setHours()	Sets the hour (from 0-23)
setMilliseconds()	Sets the milliseconds (from 0-999)
setMinutes()	Set the minutes (from 0-59)
setMonth()	Sets the month (from 0-11)
setSeconds()	Sets the seconds (from 0-59)
setTime()	Sets a date and time by adding or subtracting a specified number of milliseconds to/from midnight January 1, 1970

Built-in Objects

Date



Method	Description
setUTCDate()	Sets the day of the month, according to universal time (from 1-31)
setUTCFullYear()	Sets the year, according to universal time (four digits)
setUTCHours()	Sets the hour, according to universal time (from 0-23)
setUTCMilliseconds()	Sets the milliseconds, according to universal time (from 0-999)
setUTCMinutes()	Set the minutes, according to universal time (from 0-59)
setUTCMonth()	Sets the month, according to universal time (from 0-11)
setUTCSeconds()	Set the seconds, according to universal time (from 0-59)

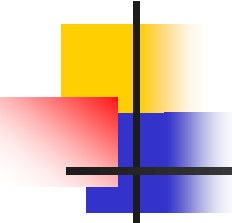
Built-in Objects

Date

Method	Description
<code>toDateString()</code>	Converts the date portion of a Date object into a readable string
<code>toLocaleDateString()</code>	Returns the date portion of a Date object as a string, using locale conventions
<code>toLocaleTimeString()</code>	Returns the time portion of a Date object as a string, using locale conventions
<code>toLocaleString()</code>	Converts a Date object to a string, using locale conventions
<code>toString()</code>	Converts a Date object to a string
<code>getTimeString()</code>	Converts the time portion of a Date object to a string
<code>toUTCString()</code>	Converts a Date object to a string, according to universal time
<code>UTC()</code>	Returns the number of milliseconds in a date string since midnight of January 1, 1970, according to universal time
<code>valueOf()</code>	Returns the primitive value of a Date object

Built-in Objects

Date



```
<script type="text/javascript">  
    var d=new Date();  
    document.write("Original form: ");  
    document.write(d + "<br>");  
    document.write("Formatted form: ");  
    document.write(d.toLocaleDateString());  
</script>
```

Output

Original form: Mon May 17 16:46:50 UTC+0200 2010

Formatted form: lunedì 17 maggio 2010



Built-in Objects

Math

- The Math object **allows performing mathematical tasks.**
- The Math object includes several mathematical constants and methods.
- Math is not a constructor. All properties/methods of Math can be called by using Math as an object, without creating it.
- Note that **all trigonometric methods of Math take arguments in radians.**

```
var pi_value=Math.PI;  
var sqrt_value=Math.sqrt(16);
```

Built-in Objects

Math

- Note: Case-Sensitive

Property	Description
E	Returns Euler's number (approx. 2.718)
LN2	Returns the natural logarithm of 2 (approx. 0.693)
LN10	Returns the natural logarithm of 10 (approx. 2.302)
LOG2E	Returns the base-2 logarithm of E (approx. 1.442)
LOG10E	Returns the base-10 logarithm of E (approx. 0.434)
PI	Returns PI (approx. 3.14159)
SQRT1_2	Returns the square root of 1/2 (approx. 0.707)
SQRT2	Returns the square root of 2 (approx. 1.414)

Built-in Objects

Math

Method	Description
<code>abs(x)</code>	Returns the absolute value of x
<code>acos(x)</code>	Returns the arccosine of x , in radians
<code>asin(x)</code>	Returns the arcsine of x , in radians
<code>atan(x)</code>	Returns the arctangent of x as a numeric value between $-\pi/2$ and $\pi/2$ radians
<code>atan2(y,x)</code>	Returns the arctangent of the quotient of its arguments
<code>ceil(x)</code>	Returns x , rounded upwards to the nearest integer
<code>cos(x)</code>	Returns the cosine of x (x is in radians)
<code>exp(x)</code>	Returns the value of E^x

Built-in Objects

Math

Method	Description
<code>floor(x)</code>	Returns x, rounded downwards to the nearest integer
<code>log(x)</code>	Returns the natural logarithm (base E) of x
<code>max(x,y,z,...,n)</code>	Returns the number with the highest value
<code>min(x,y,z,...,n)</code>	Returns the number with the lowest value
<code>pow(x,y)</code>	Returns the value of x to the power of y
<code>random()</code>	Returns a random number between 0 and 1
<code>round(x)</code>	Rounds x to the nearest integer
<code>sin(x)</code>	Returns the sine of x (x is in radians)
<code>sqrt(x)</code>	Returns the square root of x
<code>tan(x)</code>	Returns the tangent of an angle

Built-in Objects

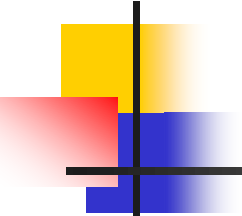
Number

- The **Number** object is an object wrapper for primitive numeric values.
- The Number object has properties for **numerical constants**, such as maximum value, not-a-number, and infinity. You cannot change the values of these properties and you use them as follows:
 biggestNum = Number.MAX_VALUE
 smallestNum = Number.MIN_VALUE
- Typically, you create a Number object when you want to add some properties to them through the prototype property

```
var num = new Number(value);  
Number.prototype.newproperty=value
```

Built-in Objects

Number



Property	Description
constructor	Returns the function that created the Number object's prototype
MAX_VALUE	Returns the largest number possible in JavaScript
MIN_VALUE	Returns the smallest number possible in JavaScript
NEGATIVE_INFINITY	Represents negative infinity (returned on overflow)
POSITIVE_INFINITY	Represents infinity (returned on overflow)
prototype	Allows you to add properties and methods to an object

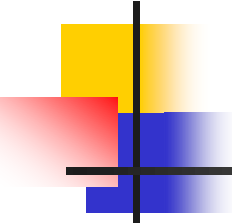
Built-in Objects

Number

Method	Description
toExponential(x)	Converts a number into an exponential notation
	<i>number.toExponential(x)</i>
	x Optional. An integer between 0 and 20 representing the number of digits in the notation after the decimal point. If omitted, it is set to as many digits as necessary to represent the value
toFixed(x)	Formats a number with x numbers of digits after the decimal point
	<i>number.toFixed(x)</i>
	x Optional. The number of digits after the decimal point. Default is 0 (no digits after the decimal point)
toPrecision(x)	Formats a number to x length
	<i>number.toPrecision(x)</i>
	x Optional. The number of digits. If omitted, it returns the entire number (without any formatting)
toString()	Converts a Number object to a string
valueOf()	Returns the primitive value of a Number object

Built-in Objects

Number



```
<script type="text/javascript">  
    var num = new Number(13.3714);  
    document.write(Number.MAX_VALUE+"<br>");  
    document.write(Number.MIN_VALUE+"<br>");  
    document.write(num.toExponential(4)+"<br>");  
    document.write(num.toFixed(2)+"<br>");  
    document.write(num.toPrecision(3)+"<br>");  
    document.write(num.toString()+"<br>");  
    document.write(num.valueOf());  
</script>
```



Built-in Objects

String

- The String object is a wrapper around the string primitive data type. **Do not confuse a string literal with the String object.**
- String objects are created with `new String()`.
`s1 = "Hi" //creates a string literal value`
`s2 = new String("Hi") //creates a String object`
- **You can call any of the methods of the String object on a string literal value**
 - JavaScript automatically converts the string literal to a temporary String object, calls the method, then discards the temporary String object.

Built-in Objects

String

- You can also use the `String.length` property with a string literal.
- A String object has two types of methods:
 - those that return a variation on the string itself, such as `substring` and `toUpperCase`,
 - those that return an HTML-formatted version of the string, such as `bold` and `link`.

Built-in Objects

String

Method	Description
<code>charAt()</code>	Returns the character at the specified index
<code>charCodeAt()</code>	Returns the Unicode of the character at the specified index
<code>concat()</code>	Joins two or more strings, and returns a copy of the joined strings
	<code>string.concat(string2, string3, ..., stringX)</code>
<code>fromCharCode()</code>	Converts Unicode values to characters
<code>indexOf()</code>	Returns the position of the first found occurrence of a specified value in a string
<code>lastIndexOf()</code>	Returns the position of the last found occurrence of a specified value in a string (-1 if the value to search for never occurs)
	<code>string.lastIndexOf(searchstring, start)</code>
	<code>searchstring</code> Required. The string to search for Start Optional. The start position in the string to start the search. If omitted, the search starts from position 0

Built-in Objects

String

Method	Description
match()	Searches for a match between a regular expression and a string, and returns the matches
	<i>string.match(regex)</i>
	<i>regex</i> Required. A regular expression.
replace()	Searches for a match between a substring (or regular expression) and a string, and replaces the matched substring with a new substring
	<i>string.replace(regex/substr,newstring)</i>
	<i>regex/substr</i> Required. A substring or a regular expression. <i>newstring</i> Required. The string to replace the found value in parameter 1
search()	Searches for a match between a regular expression and a string, and returns the position of the match

Built-in Objects

String

Method	Description
slice()	<p>Extracts a part of a string and returns a new string</p> <p><i>string.slice(begin,end)</i></p> <p><i>begin</i> Required. The index where to begin the extraction. First character is at index 0</p> <p><i>end</i> Optional. Where to end the extraction. If omitted, slice() selects all characters from the begin position to the end of the string</p>
split()	<p>Splits a string into an array of substrings</p> <p><i>string.split(separator, limit)</i></p> <p><i>separator</i> Optional. Specifies the character to use for splitting the string. If omitted, the entire string will be returned</p> <p><i>limit</i> Optional. An integer that specifies the number of splits</p>

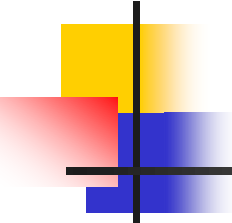
Built-in Objects

String

Method	Description
<code>substr()</code>	Extracts the characters from a string, beginning at a specified start position, and through the specified number of character
<code>substring()</code>	Extracts the characters from a string, between two specified indices
<code>toLowerCase()</code>	Converts a string to lowercase letters
<code>toUpperCase()</code>	Converts a string to uppercase letters
<code>valueOf()</code>	Returns the primitive value of a String object

Built-in Objects

String



```
<script type="text/javascript">
<!-- HTML comment
var str="Hello world!";
document.write(str + "<br>");
document.write(str.substring(1) + "<br>");
document.write(str.substring(3,7) + "<br>");
document.write(str.split('o',2) + "<br>");
document.write(str.toUpperCase() + "<br>");
document.write(str.toLowerCase());
-->
</script>
```



JavaScript

RegExp

- A **regular expression** is an object that describes a pattern of characters.
 - A **simple pattern** can be one single character.
 - A **more complicated pattern** can consist of more characters, and can be used for parsing, format checking, substitution and more.
- **Regular expressions** are used to perform powerful pattern-matching and "search-and-replace" functions on text.

```
var txt=new RegExp(pattern,modifiers);  
or more simply  
var txt=/pattern/modifiers;
```

pattern specifies the pattern of an expression

modifiers specify if a search should be global, case-sensitive, etc.

Built-in Objects

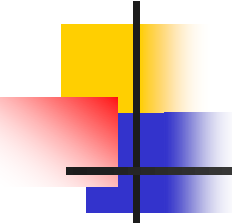
RegExp

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)
m	Perform multiline matching

Method	Description
exec()	Tests for a match in a string. Returns the first match regexp.exec(str) regexp(str)
test()	Tests for a match in a string. Returns true or false regexp.test(str)

Built-in Objects

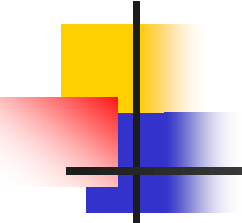
RegExp



Expression	Description
[abc]	Find any character between the brackets
[^abc]	Find any character not between the brackets
[0-9]	Find any digit from 0 to 9
[a-z]	Find any character from lowercase a to lowercase z
[A-Z]	Find any character from uppercase A to uppercase Z
[a-Z]	Find any character from lowercase a to uppercase Z
adgk	Find the sequence of characters
(red blue green)	Find any of the alternatives specified

Built-in Objects

RegExp



Metachar	Description
.	Find a single character, except newline or line terminator
\w	Find a word character
\W	Find a non-word character
\d	Find a digit
\D	Find a non-digit character
\s	Find a whitespace character
\S	Find a non-whitespace character
\b	Find a match at the beginning/end of a word
\B	Find a match not at the beginning/end of a word

Built-in Objects

RegExp

Metachar	Description
<code>\0</code>	Find a NUL character
<code>\n</code>	Find a new line character
<code>\f</code>	Find a form feed character
<code>\r</code>	Find a carriage return character
<code>\t</code>	Find a tab character
<code>\v</code>	Find a vertical tab character
<code>\xxx</code>	Find the character specified by an octal number xxx
<code>\xdd</code>	Find the character specified by a hexadecimal number dd
<code>\uxxxx</code>	Find the Unicode character specified by a hexadecimal number xxxx

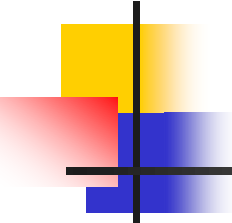
Built-in Objects

RegExp

Quantifier	Description
n^+	Matches any string that contains at least one n
n^*	Matches any string that contains zero or more occurrences of n
$n^?$	Matches any string that contains zero or one occurrences of n
$n\{X\}$	Matches any string that contains a sequence of X n 's
$n\{X,Y\}$	Matches any string that contains a sequence of X or Y n 's
$n\{X, \}$	Matches any string that contains a sequence of at least X n 's
$n\$$	Matches any string with n at the end of it
n	Matches any string with n at the beginning of it
$?=n$	Matches any string that is followed by a specific string n
$?!n$	Matches any string that is not followed by a specific string n

Built-in Objects

RegExp



```
<script type="text/javascript">  
  var str="100, 1000 or 10000?";  
  var patt1=/\d{3,4}/g;  
  document.write(str.match(patt1));  
</script>
```

Output:

100,1000,1000



Built-in Objects

Global Properties and Functions

- The JavaScript global properties and functions can be used with all the built-in JavaScript objects.

Property	Description
Infinity	A numeric value that represents positive/negative infinity
NaN	"Not-a-Number" value
undefined	Indicates that a variable has not been assigned a value

Function	Description
decodeURI()	Decodes a URI
decodeURIComponent()	Decodes a URI component
encodeURI()	Encodes a URI
encodeURIComponent()	Encodes a URI component

Built-in Objects

Global Properties and Functions

Function	Description
<code>escape()</code>	Encodes a string
<code>eval()</code>	Evaluates a string and executes it as if it was script code. First, <code>eval()</code> determines if the argument is a valid string, then <code>eval()</code> parses the string looking for JavaScript code. If it finds any JavaScript code, it will be executed.
<code>isFinite()</code>	Determines whether a value is a finite, legal number
<code>isNaN()</code>	Determines whether a value is an illegal number
<code>Number()</code>	Converts an object's value to a number
<code>parseFloat()</code>	Parses a string and returns a floating point number
<code>parseInt()</code>	Parses a string and returns an integer
<code>String()</code>	Converts an object's value to a string
<code>unescape()</code>	Decodes an encoded string



Built-in Objects

Global Properties and Functions

```
<script type="text/javascript">  
    eval("x=10;y=20;document.write(x*y)");  
    document.write("<br>" + eval("2+2"));  
    document.write("<br>" + eval(x+17));  
</script>
```

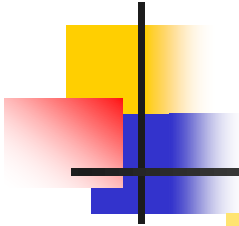
Output:

200

4

27



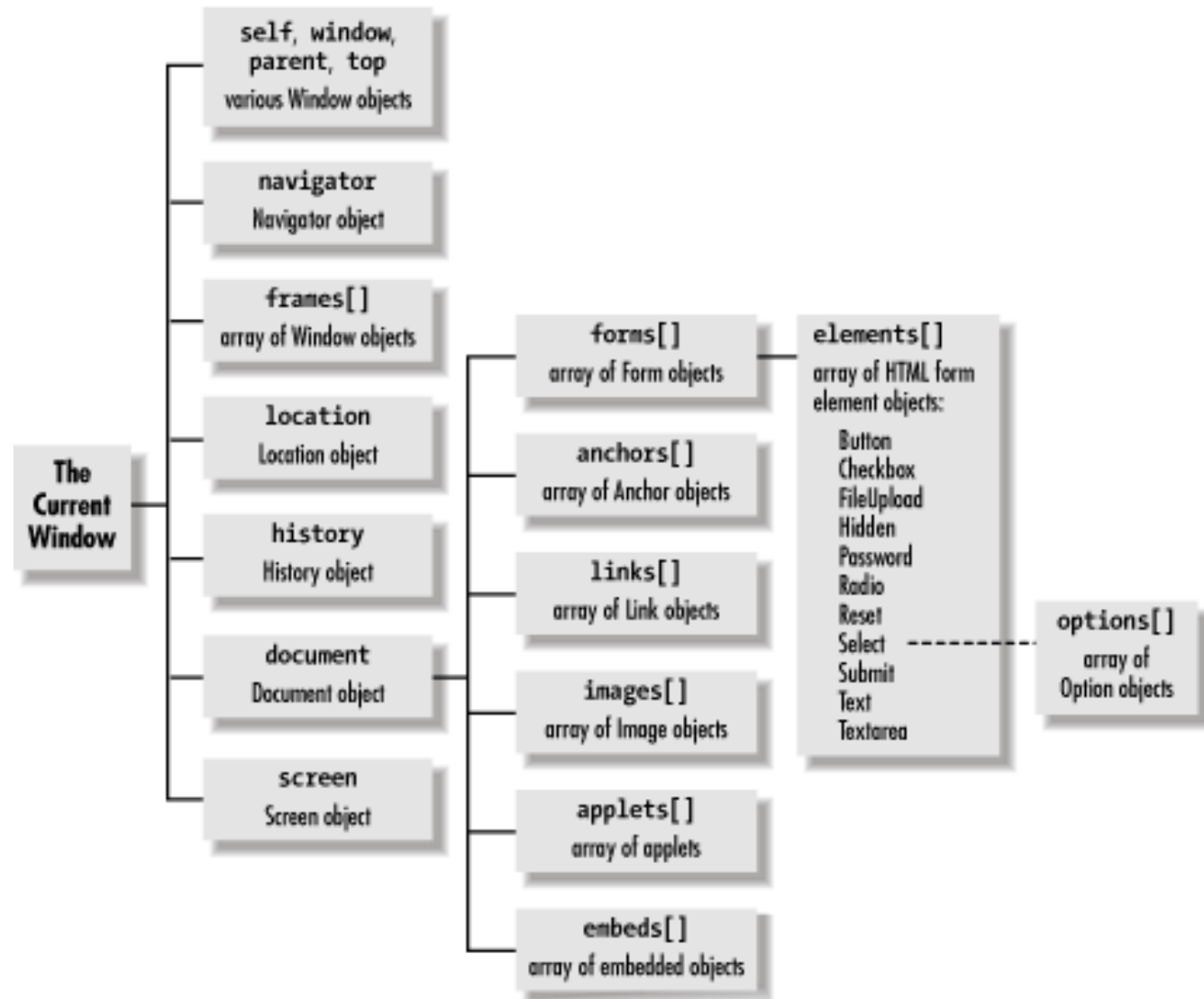


Client-side Objects



Client-side objects

When you load a document in a Browser, it creates a number of JavaScript Objects.



Client-side objects

Objects in a Page

Every page has the following objects

- **navigator**: has properties for the name and version of the Navigator being used, for the MIME types supported by the client, and for the plugins installed on the client.
- **window**: the top-level object; has properties that apply to the entire window. There is also a *window* object for each "child window" in a frames document.
- **document**: contains properties based on the content of the document, such as title, background color, links and forms.
- **location**: has properties based on the current URL.
- **history**: contains properties representing URLs the client has previously requested.

Client-side objects

Controlling objects

By controlling Objects, we can

- **Open new browser windows**, and determine their size and whether they should have scroll bars, or location windows
- **Change the content of multiple frames at once**, even rewriting the HTML in a frame without downloading a new file
- **Add or remove text from forms** including text areas and select boxes
- **Control the background color of your Web pages** using JavaScript
- etc...

Client-side objects

Window Object

- Every browser window that is currently open will have a corresponding **window Object**.
- If a document contain frames (<frame> or <iframe> tags), **the browser creates one window object for the HTML document, and one additional window object for each frame.**
- All the other Objects are children of one of the window Objects, except for navigator and screen.

Client-side objects

Window Object

- Every window **has a history of the previous pages** that have been displayed in that window, which are represented by the various properties of the history object.
- **JavaScript maintains an idea of the current window**, so that almost all references to sub-objects of the current window do not need to refer to it explicitly.
 - This is why all of our output has been done using `document.write()` rather than `window.document.write()`.

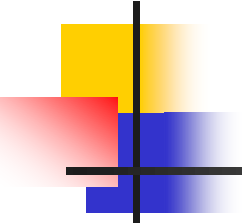
Client-side objects

Window Object Properties

Property	Description
closed	Returns a Boolean value indicating whether a window has been closed or not
defaultStatus	Sets or returns the default text in the statusbar of a window
document	Returns the Document object for the window
frames	Returns an array of all the frames (including iframes) in the current window
history	Returns the History object for the window
innerHeight	Sets or returns the inner height of a window's content area
innerWidth	Sets or returns the inner width of a window's content area
length	Returns the number of frames (including iframes) in a window

Client-side objects

Window Object Properties



Property	Description
location	Returns the Location object for the window
name	Sets or returns the name of a window
navigator	Returns the Navigator object for the window
opener	Returns a reference to the window that created the window
outerHeight	Sets or returns the outer height of a window, including toolbars/scrollbars
outerWidth	Sets or returns the outer width of a window, including toolbars/scrollbars
pageXOffset	Returns the pixels the current document has been scrolled (horizontally) from the upper left corner of the window
pageYOffset	Returns the pixels the current document has been scrolled (vertically) from the upper left corner of the window

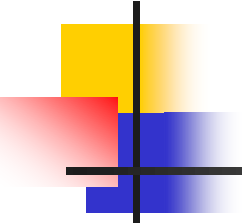
Client-side objects

Window Object Properties

Property	Description
parent	Returns the parent window of the current window
screen	Returns the Screen object for the window
screenLeft	Returns the x coordinate of the upper-left hand corner of the window (Explorer – Opera)
screenTop	Returns the y coordinate of the upper-left hand corner of the window (Explorer – Opera)
screenX	Returns the x coordinate of the upper-left hand corner of the window (Firefox – Chrome)
screenY	Returns the y coordinate of the upper-left hand corner of the window (Firefox – Chrome)
self	Returns the current window
status	Sets the text in the statusbar of a window
top	Returns the topmost browser window

Client-side objects

Window Object Properties



Method	Description
<code>alert()</code>	Displays an alert box with a message and an OK button
<code>blur()</code>	Removes focus from the current window
<code>clearInterval()</code>	Clears a timer set with <code>setInterval()</code>
<code>clearTimeout()</code>	Clears a timer set with <code>setTimeout()</code>
<code>close()</code>	Closes the current window
<code>confirm()</code>	Displays a dialog box with a message and an OK and a Cancel button
<code>focus()</code>	Sets focus to the current window

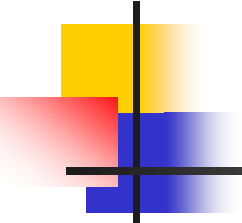
Client-side objects

Window Object Methods

Method	Description
<code>moveBy()</code>	Moves a window relative to its current position
<code>moveTo()</code>	Moves a window to the specified position
<code>open()</code>	Opens a new browser window
<code>print()</code>	Prints the content of the current window
<code>prompt()</code>	Displays a dialog box that prompts the visitor for input
<code>resizeBy()</code>	Resizes the window by the specified pixels
<code>resizeTo()</code>	Resizes the window to the specified width and height

Client-side objects

Window Object Methods



Method	Description
<code>scrollBy()</code>	Scrolls the content by the specified number of pixels
<code>scrollTo()</code>	Scrolls the content to the specified coordinates
<code>setInterval()</code>	Calls a function or evaluates an expression at specified intervals (in milliseconds)
<code>setTimeout()</code>	Calls a function or evaluates an expression after a specified number of milliseconds

Client-side objects

Window Object Methods

```
<title>Example</title>
<script type="text/javascript">
  var myWindow;
  function openWin()
  { myWindow=window.open(",",'width=200,height=100');
    myWindow.document.write("This is 'myWindow'");
  }

  function closeWin()
  { myWindow.close(); }

  function moveWin()
  { myWindow.moveBy(250,250);
    myWindow.focus();}
</script>
</head>
```

Client-side objects

Window Object Methods

```
<body>
<div>
  <input type="button" value="Open 'myWindow'"
  onclick="openWin()" >
  <br><br>
  <input type="button" value="Move 'myWindow'"
  onclick="moveWin()" >
  <br><br>
  <input type="button" value="Close 'myWindow'"
  onclick="closeWin()" >
</div>
</body>
</html>
```



Client-side objects

Navigator Object

- The **navigator object** contains information about the browser.
- The JavaScript runtime engine on the client automatically creates the navigator object.
- Use the navigator object to determine **which version** of the Navigator your users have, **what MIME types** the user's Navigator can handle, and **what plugins** the user has installed.
- **All of the properties of the navigator object are read-only.**

Client-side objects

Navigator Object

Property	Description
appName	Returns the code name of the browser
appVersion	Returns the name of the browser
appVersion	Returns the version information of the browser
battery	Returns information about the battery charging status.
cookieEnabled	Determines whether cookies are enabled in the browser
platform	Returns for which platform the browser is compiled
userAgent	Returns the user-agent header sent by the browser to the server

Client-side objects

Navigator Object (example 1)

```
<script type="text/javascript">
document.write("Code Name: " + navigator.appCodeName +
"<br>");
document.write("Name: " + navigator.appName + "<br>");
document.write("Platform: " + navigator.platform + "<br>");
document.write("User Agent: " + navigator.userAgent + "<br>");
</script>
```

Output:

Code Name: Mozilla

Name: Microsoft Internet Explorer

Platform: Win32

User Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1;
Trident/4.0; GTB6.5; .NET CLR 1.1.4322; .NET CLR 2.0.50727;
.NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)

Client-side objects

Navigator Object (example 2)

```
\\myscript5.js  
var infobrowser = "<h1>Information on your browser</h1><p>";  
for (var proprietyName in navigator)  
    infobrowser += "<strong>" + proprietyName + ":</strong>" +  
    navigator[proprietyName] + "<br>";  
infobrowser += "</p>";  
document.writeln(infobrowser);  
if (navigator.appName.indexOf("Netscape") != -1)  
    window.alert("Your browser is Netscape");  
else if (navigator.appName.indexOf("Microsoft Internet Explorer") !=  
    -1)  
    window.alert("Your browser is Internet Explorer");
```



Client-side objects

Navigator Object (example 2)

Information on your browser

appCodeName:Mozilla

appName:Microsoft Internet Explorer

appMinorVersion:0

cpuClass:x86

platform:Win32

plugins:[object]

opsProfile:null

userProfile:null

systemLanguage:en-us

userLanguage:it

appVersion:4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; GTB6.5; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)

userAgent:Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; GTB6.5; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)

onLine:true

cookieEnabled:true

mimeTypes:

Client-side objects

Screen Object

- Contains read-only properties describing the display screen and colours.

Property	Description
availHeight	Returns the height of the screen (excluding the Windows Taskbar)
availWidth	Returns the width of the screen (excluding the Windows Taskbar)
colorDepth	Returns the bit depth of the color palette for displaying images (the \log_2 of the number of colors)
height	Returns the total height of the screen
pixelDepth	Returns the color resolution (in bits per pixel) of the screen
width	Returns the total width of the screen

Client-side objects

Screen Object (example)

```
//myscript6.js
document.writeln("<h1>Screen Information</h1><p>");
printScreenInfo('width'); printScreenInfo('height');
printScreenInfo('colorDepth'); printScreenInfo('availWidth');
printScreenInfo('availHeight');
function printScreenInfo(name) {
document.writeln("<strong>" + name + " :</strong>" +
  screen[name] + "<br>");
}
```

Output:

width:1680
height:1050
colorDepth:32
availWidth:1680
availHeight:1016



Client-side objects

History Object (properties)

- The **history object** contains the URLs visited by the user (within a browser window).
- The history object is part of the window object and is accessed through the **window.history** property

length – contains the current length of the history list

- The history Object does not contain any values that reflect the actual URLs in the history list.
- Therefore, you cannot, for example, perform some action that reads the value of "URL number 3 in the history list"
- The history Object is designed for navigating the history list.

Client-side objects

History Object (Methods)

- **back()** – moves the user to the URL one place previous in the history list (previous to the current position).
- **forward()** - moves the user one URL forward, relative to current position, in the history list
- **go(offset)** - accepts an integer parameter, positive or negative, as offset.
 - If the parameter is a positive integer, the program will move the user that many places forward in the history list.
 - If the parameter is negative, it'll move the user that many places backward (previous to the current position) in the history list.
 - The current position is always place zero.
- **go(substring)** - searches for the newest history list URL that contains the specified string somewhere within its URL string
 - `history.go("email.htm");`

Client-side objects

History Object (Example)

<body>

<div>

Go to the page:

<button onclick="history.go(-1);">Back</button>

<button onclick="history.go(0);">Current</button>

<button onclick="history.go(+1);">Forward</button>

</div>



Client-side objects

Location Object (properties)

The location Object contains information on the current URL.

- **href** - contains the string value of the entire URL
 - **window.open(location.href, "windowName", "feature1,feature2, ...");**
open a new window, which connects to the same URL as the current window (so you can look at another portion of the same page at the same time you're looking at the current portion of the page)
 - **location.href = "http://www.someISP.com/~me/myPage.htm";**
Launch a new page without opening a new window (you leave your current page, including other JavaScript programs).

Client-side objects

Location Object (properties)

- **host** - holds only the hostname and port of the current page's URL
 - Example:
`http://www.someISP.com:80/~me/myPage.htm`
`location.host` -> " www.someISP.com:80"
 - Note: Assigning a value to `location.host` will generate an error because it is nonsensical. Whereas you can assign an entire URL to `location.href`, which would then connect to that URL. You cannot connect to just a host.
- **port** - contains the value of the port number in the URL

Client-side objects

Location Object (properties)

- **hostname** – returns the hostname portion of the URL.
- **pathname** - is the portion of the URL that describes the location of the Web document on the host computer.

- Example:

`http://www.someISP.com/~me/myPage.htm`

`location.pathname -> /~me/myPage.htm`

Assigning a value to this property,

`location.pathname = "~me/otherdocs/newdoc.htm";`

will cause the Web browser to load that document into the current window.

Client-side objects

Location Object (properties)

- **protocol** – contains the leftmost portion of the URL.
 - By checking the `location.protocol` property, the JavaScript program can determine if the page currently resides in was delivered by HTTP, FTP, or NEWS.
- **hash** – contains the value following the hash mark
 - Some URLs contain special hash mark values following the pathname.
`http://www.someISP.com/~me/myPage.htm#item1`
The hash mark (#) specifies the name of an anchor to jump to in the Web page.
`location.hash` -> `item1`
You can use the `location.hash` property in Event Handlers to bring the user to specific locations within the current page.

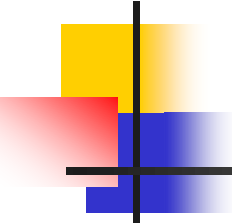
Client-side objects

Location Object (properties)

- **search** – contains the value following the question mark.
 - Some URLs contain search parameters following the pathname, denoted with a question mark (?).
 - A form entry is probably the most common use for a search parameter.
`http://www.someISP.com/~me/myProgram?formData`
`location.search -> formData`
- **reload()** – reloads the document that is currently displayed in the window of the location Object.
- **assign(newURL)** – loads a new document.
- **replace(newURL)** – replaces the current document with a new one.

Client-side objects

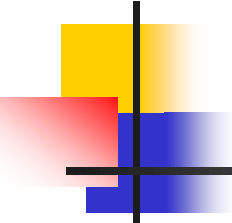
Location Object (example)



```
<html>
<head>
<title> Location Object</title>
<script type="text/javascript" src="./myscript7.js">
</script>
</head>
<body>
<form action="#" method="get">
<p><strong>Insert your name:</strong><br>
<input type="text" name="Name">
<input type="button" onclick= "send()" value="SEND"></p>
</form>
</body>
</html>
```


Client-side objects

Location Object (example)



```
//myscript7.js
display('URL',location.href);
display('Protocol',location.protocol);
display('Host name', location.hostname);
display('Local Address', location.hash);
display('Port', location.port);
display('Path', location.pathname);

function send() {
location.search+"&Name=" + document.forms[0].Name.value;}

function display(propriety, value) {
document.writeln("<p><strong>" + propriety + ":</strong>" + value + "
</p>");}
```



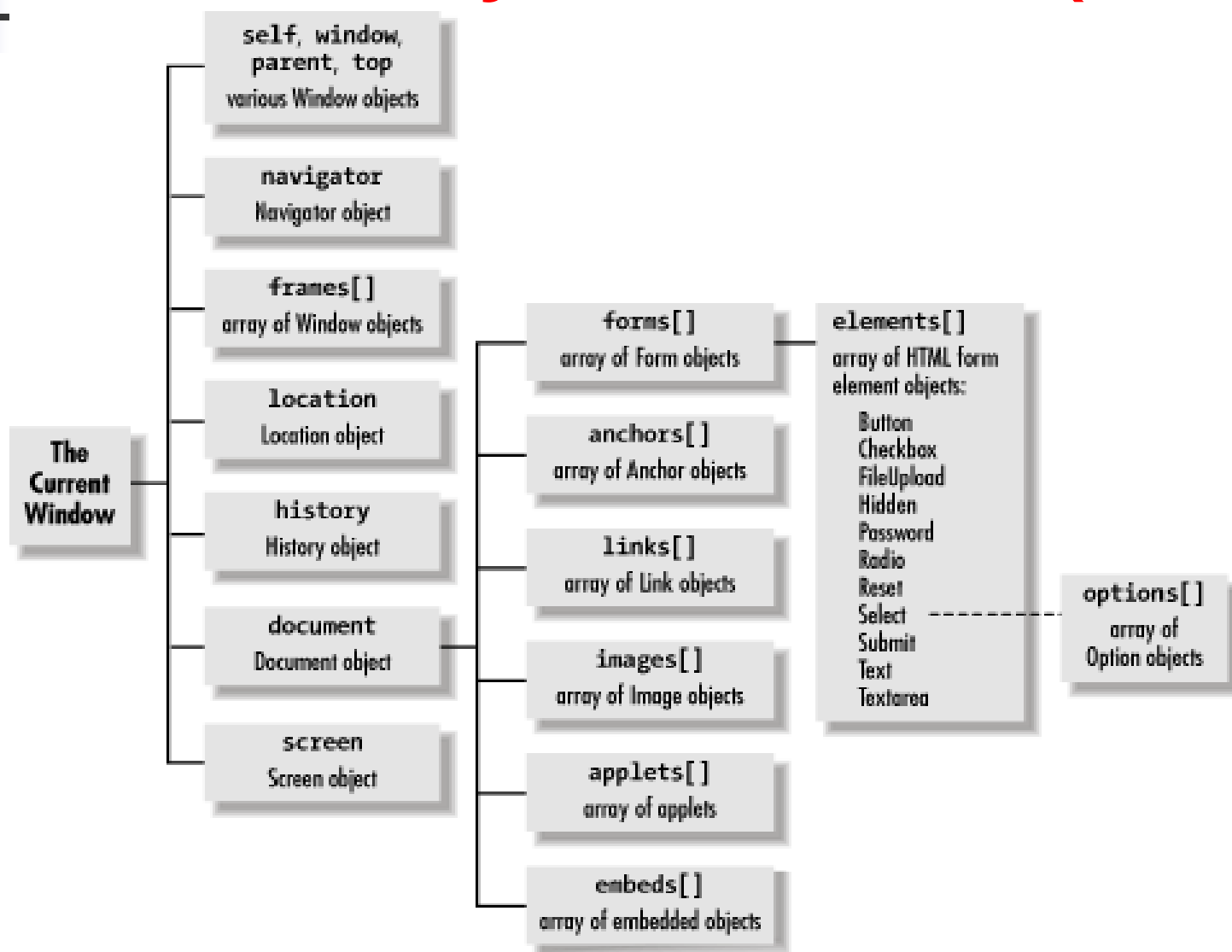
Client-side objects

Document Object

- Each HTML document loaded into a browser window becomes a Document object.
- The Document object provides access to all HTML elements in a page, from within a script.
- Tip: The Document object is also part of the Window object, and can be accessed through the window.document property
- A document is the file of HTML codes that describe a given page.
 - A page is what appears within the browser window. So, every window is associated with a document object.

Client-side objects

Document Object Collections (DOM 0)



Client-side objects

Document Object (Properties)

Property	Description
activeElement	Returns the currently focused element.
body	Returns the <body> element of the current document.
cookie	Returns a semicolon-separated list of the cookies for that document or sets a single cookie.
defaultView	Returns a reference to the window object.
dir	Gets/sets directionality (rtl/ltr) of the document.
domain	Specifies the domain name of the server that served a document
head	Returns the <head> element of the document.
lastModified	Returns the date on which the document was last modified.
location	Returns the URI of the current document.



Client-side objects

Document Object (Properties)

Property	Description
plugins	Returns a list of the available plugins.
readyState	Returns loading status of the document.
referrer	Returns the URI of the page that linked to this page.
title	Returns the title of the current document.
URL	Returns a string containing the URL of the current document.

- Note: document.URL property may be modified through URL redirection. Thus, location.href contains the requested URL while document.URL specifies the actual URL where it was found.



Client-side objects

Document Object (Methods)

Method	Description
<code>close()</code>	Closes the output stream previously opened with <code>document.open()</code>
<code>open()</code>	Opens an output stream to collect the output from <code>document.write()</code> or <code>document.writeln()</code>
<code>write()</code>	Writes HTML expressions or JavaScript code to a document
<code>writeln()</code>	Same as <code>write()</code> , but adds a newline character after each statement

Client-side objects

Document Object (Methods)

- Note: How is it possible to change the presentation style of the elements?
- Use the "style" property.
- Note: the names of the properties do not have hyphens and are in camelCase notation.
- For instance:
 - background-color -> backgroundColor
 - border-right-style -> borderRightStyle
- Example
 - `element.style.backgroundColor = 'yellow'`

Client-side objects

Document Object (Methods)

- **write(str)** – outputs text to the client window's document
 - document.write() "converts" everything to HTML
 - the browser then interprets the HTML and then renders the HTML
- **writeln(str)** – outputs text to the client window's document (appends a newline character to the end of the output)
 - str - strings of text are surrounded by double (or single) quotes

Examples:

```
document.write("<h1>Thank you for ordering!</h1>");
```

```
document.write(example);
```

where example is a string variable

Client-side objects

Document Object (Methods)

- **Note:** write() method actually takes a variable number of arguments, rather than just one.
 - If more than one argument is given, each of the arguments is interpreted as a string and written in turn.

document.**write**("This is Part 1 ", "and this is Part 2 ", "and this is Part 3");

- **Note:** writeln() appends a newline character to the end of the output.
 - Keep in mind that these methods output their parameters as HTML. HTML ignores newline characters when it comes to outputting to the screen. HTML does not insert line breaks in screen output unless you specify a line break using either
 or <p> tags or text resides between <pre> and </pre> tags.

Client-side objects

Document Object (Example)



```
<body> <p>
  <a href="#"
    onclick="openW('yellow')">open yellow window</a><br>
  <a href="#"
    onclick="openW('red')">open red window</a><br>
  <a href="#"
    onclick="openW('blue')">open blue window</a>
</p><p>
  <a href="#"
    onclick="openDoc('yellow')">open yellow page</a><br>
  <a href="#"
    onclick="openDoc('red')">open red page</a><br>
  <a href="#"
    onclick="openDoc('blue')">open blue page</a>
</p> </body>
```

Client-side objects

Document Object (Example)

```
document.writeln("<p>You are on " + document.URL + "<p>");  
function openW(color) {  
    win = window.open("",color,"width=400,height=250");  
    HTMLcode = '<html><head><meta      charset="utf-  
8"><title>Page ' + color + '</title></head>' +  
    '<body><p>This is a ' + color + ' page </p></body></html>';  
    win.document.writeln(HTMLcode);  
    win.document.body.style.background = color;  
    win.document.body.style.color = "white";  
}
```

Client-side objects

Document Object (Example)

```
function openDoc(color) {  
    doc = document.open("text/html","replace");  
    HTMLcode      =      '<html><head><meta      charset="utf-  
8"><title>Page '+ color+ '</title></head>' +  
    '<body><p>This is a ' + color + ' page </p></body></html>';  
    doc.writeln(HTMLcode);  
    doc.body.style.backgroundColor = color;  
    doc.body.style.color = "green";  
    doc.close();  
}
```

Client-side objects

Document Object (Methods)

Double and Single Quotes

- When writing **HTML** it is **general practice** to use **double quotes** for tag attributes, e.g.:

```

```

- When writing **JavaScript** it is **general practice** to use **single quotes** for string literals:

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

```
document.write('<p>');
```

```
</script>
```

In general these quoting styles then complement one another when outputting HTML using JavaScript

Client-side objects

How are objects named? (DOM 0)

- You can access only to a subset of all elements contained in a document, through the document object.
- The name of each Object is prefaced by the names of all the Objects that contain it, in order, separated by dots.
- For example, the **window** Object contains the **document** Object, and the **document** Object contains a **form** Object, and the **form** Object contains **text** input fields, **buttons**, **select** Objects, etc... —
`window.document.forms[0].elements[3]`

Client-side objects

How are objects named? (DOM 0)

Leave off the word "window."

- All Client-side Object names officially start with the highest Object name, "window". But since all names start with "window", JavaScript allows you to leave that off.
 - `document.forms[0].elements[3]`

Client-side objects

How are objects named? (DOM 0)

Form elements

- The numerous Object types included in the Forms Array are collectively referred to as "elements" and they can all be referred to by means of the "Elements Array" (e.g.: `elements[1]`).
- Example: **`document.forms[0].elements[1]`**

Client-side objects

How to refer to properties or methods?

- **Note: JavaScript is case sensitive**
 - `document.myform.Check1` is not the same "Object" as `document.myform.check1`
- The form Object has child objects named `button1` and `text1`, corresponding to the button and text field in the form.
- These objects have their own properties based on their HTML attribute values, for example,
 - `document.myform.button1.value` is "Press Me"
 - `document.myform.elements[2].name` is "Button1"
 - `document.myform.elements[0].name` is "text1"

Client-side objects

How to refer to properties or methods?

- Three different ways to specify the value of an object:
 - `document.myform.text1.value` is "blahblah"
NOTE: using just the object names
 - `document.myform.elements[0].value` is "blahblah"
NOTE: using an object name & array notation (to specify the object instead of the object name)
 - `document.forms[0].elements[0].value` is "blahblah"
NOTE: using just array notation

Client-side objects

How to refer to properties or methods?

```
<title>A Simple Document</title>
</head>
<body>
<p><a name="top" id="top">This is the top of the page</a><p>
<hr>
<form method="post" action="%20mailto:nobody@dev.null">
<p>Enter your name: <input type="text" name="me" size="70">
<input type="Submit" value="OK">
<input type="Reset" value="Oops"></p>
</form>
<hr>
<p>Click here to go to the <a href="#top">top</a> of the
  page</p>
</body>
</html>
```

Client-side objects

How to refer to properties or methods?

- The code creates an HTML page with an anchor at the top of the page and a link to that anchor at the bottom.
- In between is a simple form that allows the user to enter his name.
- We can also access the other HTML elements of this document using the following properties:
 - anchors
 - forms
 - links

Client-side objects

How to refer to properties or methods?

document.title

<title>A very simple HTML page</title>

document.anchors[0]

This is the top of the page

document.anchors[0].name -> top

document.forms[0]

<form method= "post" action="mailto:nobody@dev.null">

document.forms[0].method -> post

document.forms[0].action -> mailto:nobody@dev.null

Client-side objects

How to refer to properties or methods?

document.forms[0].elements[0]

`<input type="text" name="me" size="70">`

document.forms[0].elements[0].name -> me

document.forms[0].elements[0].type -> text

document.forms[0].elements[1]

`<input type="Submit" value="OK">`

document.forms[0].elements[1].value -> OK

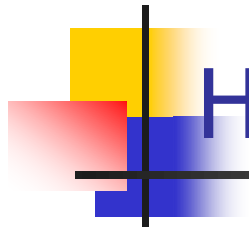
document.forms[0].elements[1].type -> Submit

document.forms[0].elements[2]

`<input type="Reset" value="Oops">`

document.forms[0].elements[2].value -> Oops

document.forms[0].elements[2].type -> Reset



Client-side objects

How to refer to properties or methods?

document.links[0]

`top`

document.links[0].href

`file:///F:/Examples/example1.htm#top`



Client-side objects

Document

- Each object in the document has specific properties and methods.
- The access to each element using the approach described so far is not flexible, is limited and does not allow changing dynamically the structure of the document.
- The ability to change a Web page dynamically with a scripting language is made possible by the **Document Object Model (DOM)**, which can connect any element on the screen to a JavaScript function.

Document Object Model

DOM

- "The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."
- The DOM is separated into 3 different parts / levels:
 - Core DOM - standard model for any structured document
 - XML DOM - standard model for XML documents
 - HTML DOM - standard model for HTML documents
- The DOM defines the objects and properties of all document elements, and the methods (interface) to access them.

Document Object Model DOM

<!DOCTYPE html>

<html>

<head>

<title>Sample page</title>

</head>

<body>

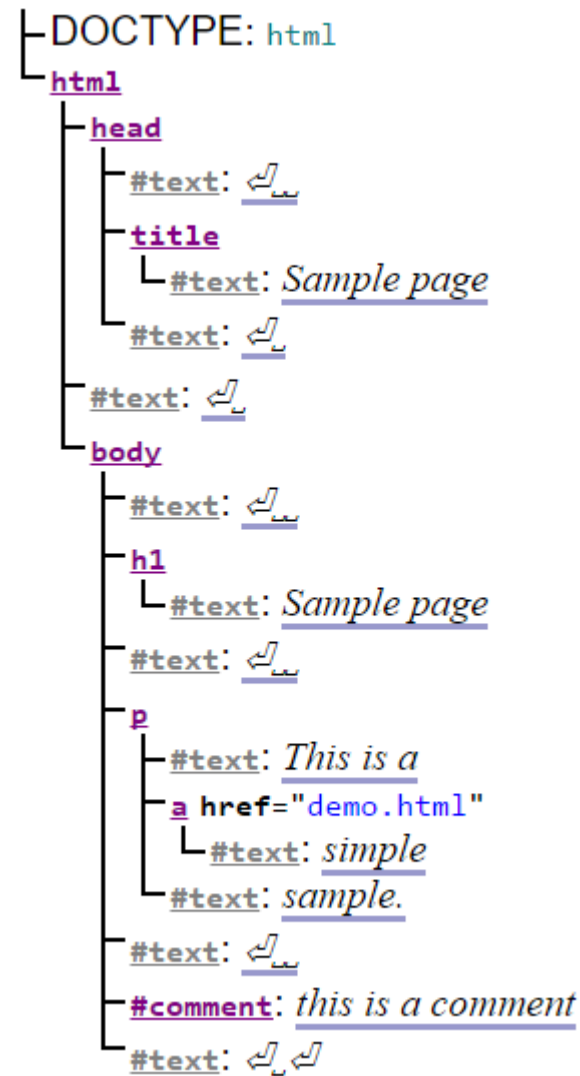
<h1>Sample page</h1>

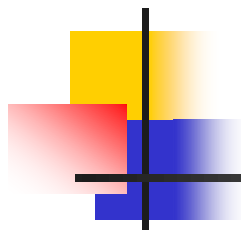
<p>This is a simple
sample.</p>

<!-- this is a comment -->

</body>

</html>





DOM

The HTML DOM is:

- A standard object model for accessing and manipulating HTML documents HTML
 - A standard programming interface for HTML
 - Platform- and language-independent
 - A W3C standard

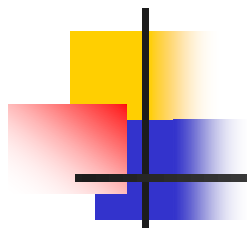
- In other words: The HTML DOM is a standard for how to get, change, add, or delete HTML elements.





DOM

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



DOM

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

- The **root node** in the HTML above is **<html>**. All other nodes in the document are contained within **<html>**.
- The **<html>** node has **two child nodes**: **<head>** and **<body>**.
- The **<head>** node holds a **<title>** node. The **<body>** node holds a **<h1>** and **<p>** nodes.



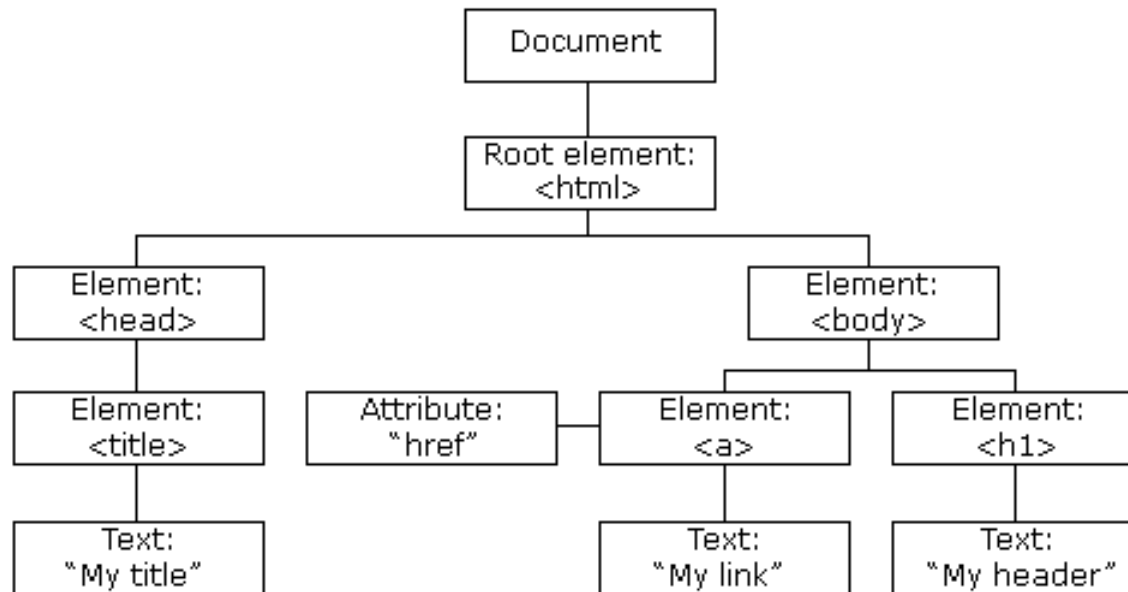


DOM

- A common error in DOM processing is to expect an element node to contain text. However, **the text of an element node is stored in a text node.**
- In this example: `<title>DOM Tutorial</title>`, the element node `<title>` holds a text node with the value "DOM Tutorial".
 - "DOM Tutorial" is not the value of the `<title>` element!

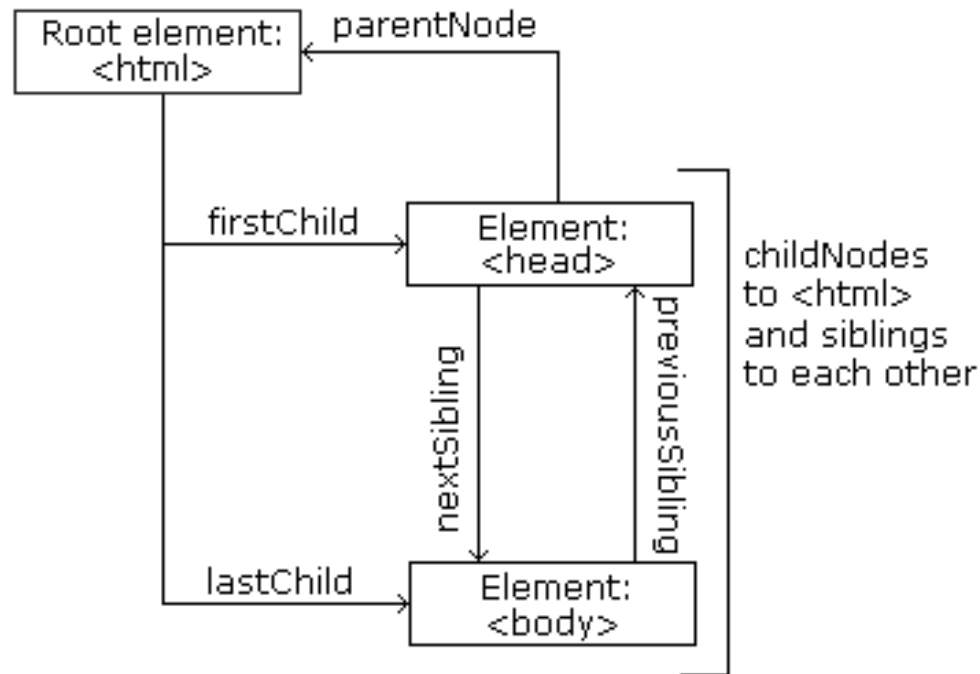
DOM Tree

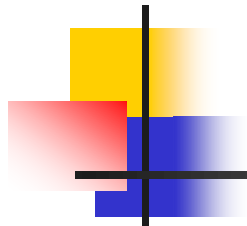
- The HTML DOM views a HTML document as a tree-structure. The tree structure is called a **node-tree**.
- All nodes can be accessed through the tree.



DOM Tree

- **Parent nodes have children.** Children on the same level are called **siblings** (brothers or sisters).
- Every node, except the root, has exactly one parent node





DOM Tree

- `<head>` element: first child of the `<html>` element
 - `<body>` element: last child of the `<html>` element
 - `<h1>` element: first child of the `<body>` element
 - `<p>` element: last child of the `<body>` element
-
- The programming interface of the DOM is defined by standard properties and methods.

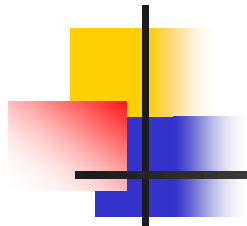




DOM (Level 3) Tree

- Some attributes:
 - **x.innerHTML** - the text value of x (not a part of the W3C DOM specification, but supported by all major browsers)
YOU MUST NOT USE INNERHTML
 - **x.childNodes** - the child nodes of x
 - **x.parentNode** - the parent of x
 - **x.firstChild** – the first child of x
 - **x.lastChild** – the last child of x
 - **x.localName** – the local part of the qualified name of x
 - **x.namespaceURI** – the namespace URI of this node of x
 - **x.nodeName** - the name of x
 - **x.nodeValue** - the value of x
 - **x.nodeType** – the type of x
- Note: In the list above, x is a node object (HTML element).





DOM (Level 3) Tree

- Some attributes:
 - **x.nextSibling** – next brother of x
 - **x.previousSibling** – previous brother of x
 - **x.textContent** – returns the text content of x and its descendants
 - **x.attributes** - the attributes nodes of x
- Note: In the list above, x is a node object (HTML element).





DOM (Level 3) Tree

- `myNode.nodeType == Node.ELEMENT_NODE`
myNode is an object Element
- `myNode.nodeType == Node.ATTRIBUTE_NODE`
Node is an attribute

Main types of nodes:

const unsigned **short** ELEMENT_NODE = 1

const unsigned **short** ATTRIBUTE_NODE = 2

const unsigned **short** TEXT_NODE = 3

const unsigned **short** ENTITY_NODE = 6 (XML documents)

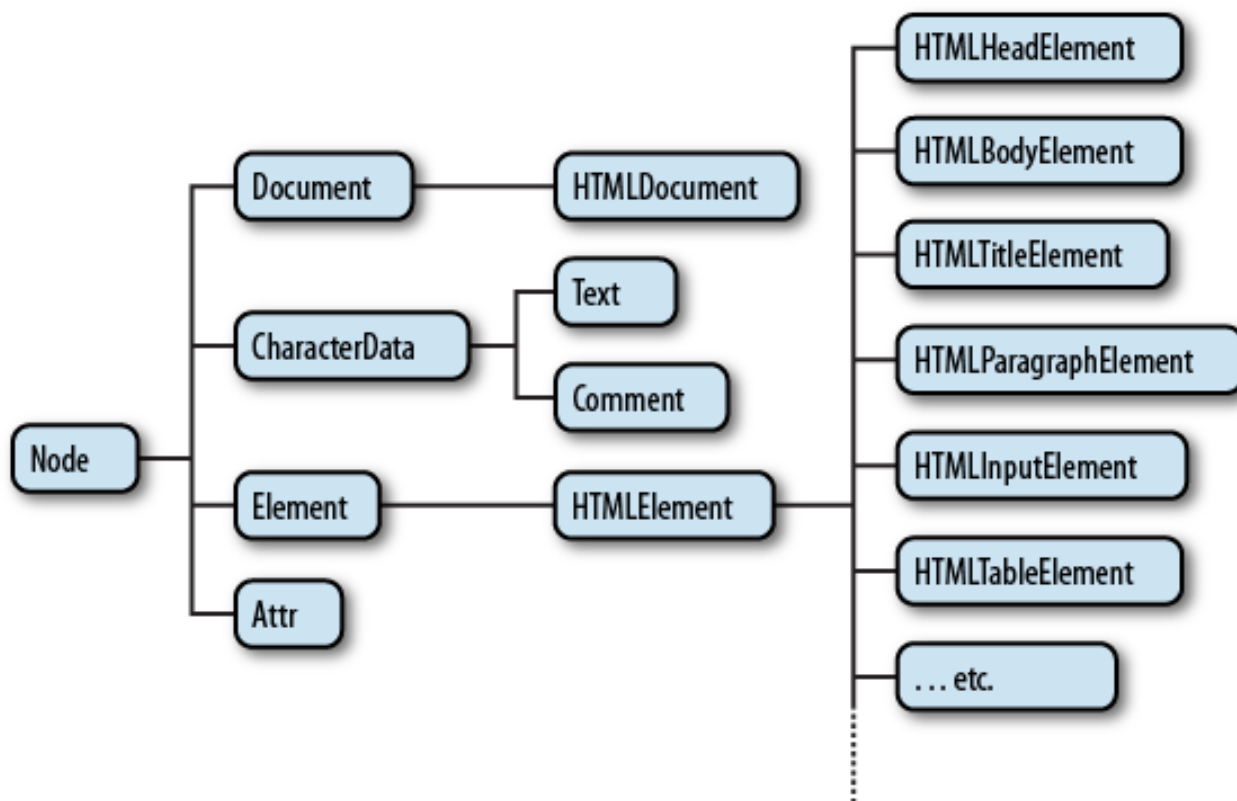
const unsigned **short** PROCESSING_INSTRUCTION_NODE = 7

const unsigned **short** COMMENT_NODE = 8

const unsigned **short** DOCUMENT_NODE = 9

DOM (Level 3) Tree

As specified by the DOM API, each HTML element is represented by an object, hence the term "Object Model".





DOM (Level 3) Tree (Methods)

- **x.getElementById(id)** - gets the element with a specified id
- **x.getElementsByTagName(name)** - gets all elements with a specified tag name
- **x.appendChild(node)** - adds a child node to the end of the list of children of x
- **x.compareDocumentPosition(node)** – compares x with the node passed as a parameter, with regard to their position in the document and according to the document order. Returns a short integer which denotes how the node is positioned relatively to x.
- **newx = x.cloneNode(deep)** – clones x to create a new object: if deep == true, all the tree with x as root is cloned; otherwise only node x
- **x.removeChild(node)** - removes a child node from x and returns it



DOM (Level 3) Tree (Methods)

- **x.replaceChild(newC,oldC)** - replaces oldC with newC and returns oldC
- **x.hasChildNodes()** – returns true if the object has sons; false otherwise
- **x.contains(node)** - returns a Boolean value indicating whether a node is a descendant of x or not
- **x.isEqualNode(node)** – tests whether x is equal to node
- **x.lookupNamespaceURI()** - takes a prefix and returns the namespace URI associated with it on the given node if found (and null if not).



DOM (Level 3) Tree (Methods)

Read and set the attributes of elements

- **x.getAttribute(string_name)** – returns the value of the attribute string_attribute
- **x.setAttribute(string_name, string_value)** – set the value of the attribute string_name to string_value

```
function showAttributes() {  
    var result;  
    var elems = document.getElementsByTagName("input");  
    for (var i=0;i<elems.length;i++) {  
        result = "";  
        for (var a in elems[i]) {  
            aValue = elems[i].getAttribute(a);  
            if (a=='type')    result += "- " + a + ": " + aValue + "\n";  
        }  
        window.alert("Attributes of " + elems[i].tagName + ":\n" + result);  
    }  
}
```



DOM Tree (get or modify the content of an element)

- Attention: The easiest way to get or modify the content of an element is by using the innerHTML property, but it is not W3C.

Two solutions:

- First Solution: Use the childNodes and NodeValue properties

```
<body>
```

```
<p id="intro">Hello World!</p>
```

```
...
```

```
function change()
```

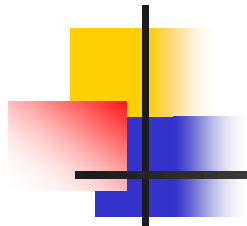
```
{ txt=document.getElementById("intro").childNodes[0].nodeValue;  
  document.getElementById("intro").childNodes[0].nodeValue="hi";  
  document.write("<p>The text from the intro paragraph: " + txt +  
    "</p>");  
}
```

DOM Tree (get or modify the content of an element)

- Second Solution: use the firstChild and NodeValue properties

function change()

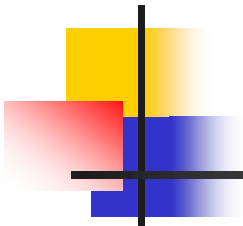
```
{  
    txt=document.getElementById("intro").firstChild.nodeValue;  
    document.getElementById("intro").firstChild.nodeValue="hi";  
    document.write("<p>The text from the intro paragraph: " + txt  
    + "<\p>");  
}
```



DOM Tree (Example)

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset="utf-8">  
    <title> Object Model Example </title>  
  </head>
```



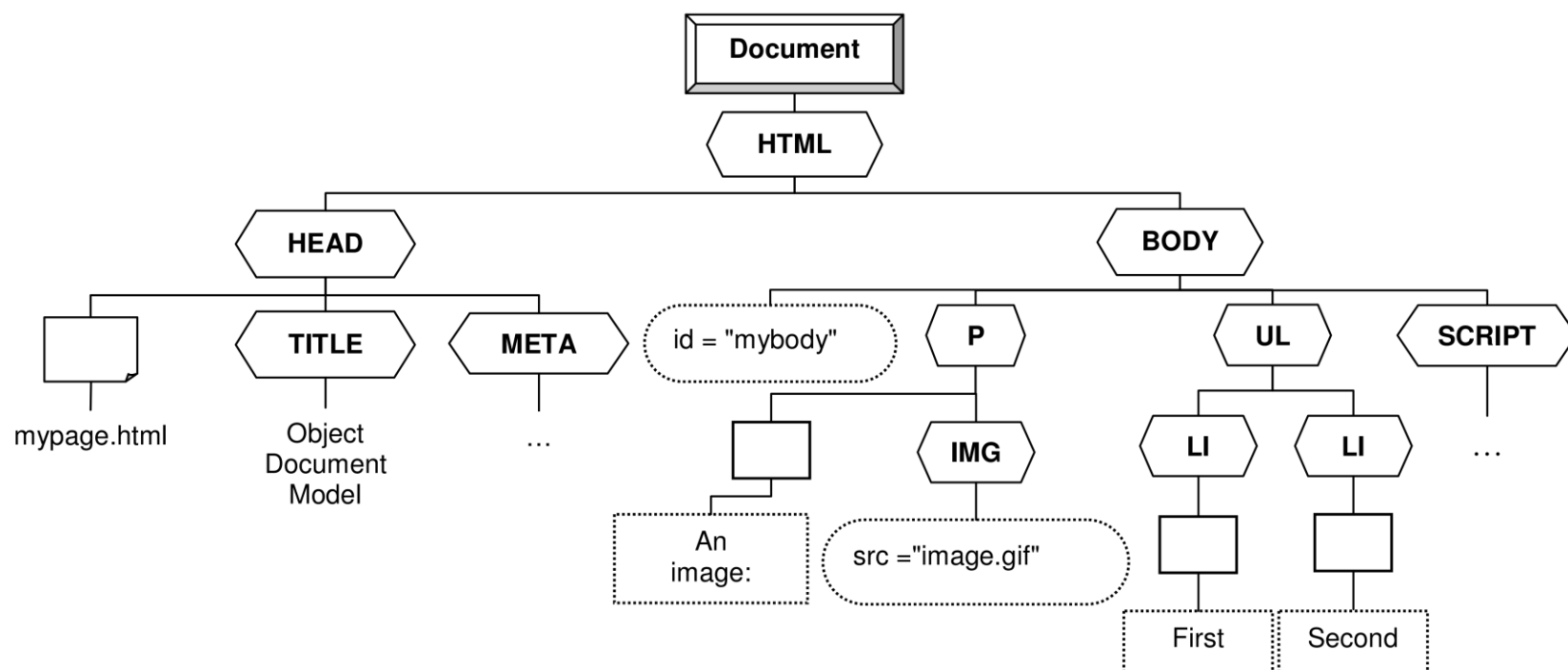


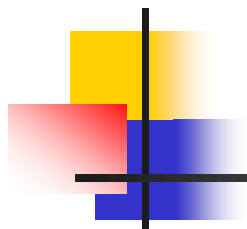
DOM Tree (Example)

```
<body id="mybody">  
  <p id="mypar1">  
    An image:  
  </p>  
  <ul>  
    <li>First  
    </li>  
    <li>Second  
    </li>  
  </ul>  
  <script type="text/javascript" src="./myscript1.js"> </script>  
</body>  
</html>
```



DOM Tree (Example)





DOM Tree

Access to a node

You can access a node in three ways:

1. By using the `getElementById()` method
2. By using the `getElementsByTagName()` method
3. By navigating the node tree, using the node relationships



DOM Tree

Access to a node

1. Use of the **getElementById method** (myscript1.js)
var text =
 document.getElementById("mypar1").firstChild.nodeValue;
window.alert("The paragraph has the text '" + text + "'");
var body = document.getElementById("mybody");
var textNode = body.childNodes[0].firstChild;
 //Explorer (old versions)
if (!textNode) window.alert("Mozilla Firefox");
else window.alert("Microsoft Explorer");
text = textNode ? textNode.nodeValue :
body.childNodes[1].firstChild.nodeValue; //Mozilla
window.alert("The paragraph has the text '" + text + "'");



DOM Tree

Access to a node

Object Model Example

An image:

DOM Inspector
(Mozilla Firefox)

Firefox (and the new versions of Explorer) add always a child node of text, also if the text is not present

DOM Tree

Access to a node

2. Using the `getElementsByTagName()` method (file `myscript2.js`)

```
var elems = document.getElementsByTagName("p");  
var text = elems[0].firstChild.nodeValue;  
window.alert("The paragraph has the text '" + text + "'");
```



DOM Tree

Access to a node

3. Navigating the DOM tree (example 1 - file myscript3.js)

```
function visit(node,obj)
{ if (node.hasChildNodes())
  { for (var i=0; i<node.childNodes.length; i++)
    { obj.txt+="CHILD " + i + " of " + node.nodeName +
      " - .#" + node.childNodes[i].nodeType+"." +
      node.childNodes[i].nodeName + "=" +
      node.childNodes[i].nodeValue + "<br>";
      visit(node.childNodes[i],obj);    }}}
var node=document.documentElement;
var obj=new Object;
obj.txt="";
visit(node,obj);
document.write(obj.txt);
```

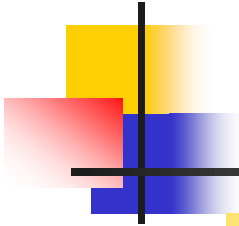
DOM Tree

Access to a node

3. Navigating the DOM tree (example 2 - file myscript4.js)

```
function visit(node,obj)
{ if (node==null) return;
  if (node.firstChild)
  {   obj.txt+=node.firstChild.nodeType + "." +
      node.firstChild.nodeName   + "=" +
      node.firstChild.nodeValue + "<br>";
      visit(node.firstChild,obj);
  }
  if (node.nextSibling)
  { obj.txt+=node.nextSibling.nodeType + "." +
      node.nextSibling.nodeName + "=" +
      node.nextSibling.nodeValue + "<br>";
      visit(node.nextSibling,obj);}
}
```





Events



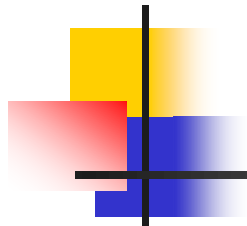
Events

- Events are actions that can be detected by JavaScript.
- Every element on a web page has certain events which can trigger a JavaScript handler.
- The events are defined in the HTML tags
- Note: events are normally used in combination with functions, and the function will not be executed before the event occurs!
 - For example, we can use the onClick event of a button element to indicate that a function will run when a user clicks on the button



Event Object (Definition)

- The event object gives you information about an event that has occurred.
- The **Event object** represents the state of an event, such as the element in which the event occurred, the state of the keyboard keys, the location of the mouse, and the state of the mouse buttons.
- Events are normally used in combination with functions (**Event handlers**), and the function will not be executed before the event occurs!
- The **event object** contains properties that describe a **JavaScript event**, and is passed as an argument to **an event handler** when the event occurs.

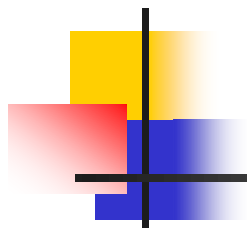


Event Object (Definition)

Example

- mouse-down event. The event object contains
 - the type of event (in this case MouseDown),
 - the x and y position of the cursor at the time of the event,
 - a number representing the mouse button used,
 - a field containing the modifier keys (Control, Alt, Meta, or Shift) that were depressed at the time of the event.
- The properties used within the event object vary from one type of event to another.
 - This variation is provided in the descriptions of individual event handlers.





Event Object

To Associate Events with Code

- **First approach** – Events as HTML element attributes
 - The event is an attribute of an HTML element
`<tag eventName = "JavaScript code or call to handler">`
eventName starts with **on**
 - Example:
`<body onLoad = "myHandler()"`
`<body onLoad = "instruction1;...; instructionN;">`

Note: it is not possible to associate any event with any element since some events have no meaning or cannot arise for some element.



Event Object

To Associate Events with Code

- **First Approach** (example)

```
<body>
```

```
  <div style="position:absolute; left:300px; top:200px;"  
    id="but">
```

```
    <button onmouseover="moveMouse()">Click</button>
```

```
  </div>
```

```
    <script      type="text/javascript"      src="./myscript10.js">  
    </script>
```

```
</body>
```

Event Object

To Associate Events with Code

- **First Approach** (example)

```
var but = document.getElementById('but');  
var diff = [150, 0, -150, 0];  
var i = 0;  
function moveMouse()  
{  
  but.style.top= (parseInt(but.style.top)+ diff[(i+3)%4]) + "px";  
  but.style.left=(parseInt(but.style.left)+ diff[(i++)%4])+ "px";  
}
```

Event Object

To Associate Events with Code

- Second Approach – to set the value of the property corresponding to the event to a specific function

```
<div style="position:absolute; left:300px; top:200px;"  
  id="but">  <button >Click</button>  
</div>
```

```
<script type="text/javascript" src="./myscript11.js"> </script>
```

```
// myscript11.js
```

```
but.onmouseover=moveMouse; //Event handlers are function references
```

```
function moveMouse()
```

```
{
```

```
  but.style.top= (parseInt(but.style.top)+ diff[(i+3)%4]) + "px";
```

```
  but.style.left=(parseInt(but.style.left)+ diff[(i++)%4])+ "px";
```

```
}
```

Note: no parentheses

Event Object

To Associate Events with Code

- **Third Approach – Specifying an event handler with a Function object.**

Function objects created with the Function constructor are evaluated each time they are used.

```
function dettaglia(elemento, evento) {  
  var t = elemento.form.area_testo;  
  var nome_elemento = elemento.name; var valore = " ";  
  if ((elemento.type == "select-one") || (elemento.type == "select-  
multiple")){ for (var i = 0; i < elemento.options.length; i++)  
    if (elemento.options[i].selected)  
      valore += elemento.options[i].value + " "; }  
  else if (elemento.type == "textarea") valore = "...";  
  else valore = elemento.value;  
  var mess = evento + ": " + nome_elemento + ' (' + valore + ')\n';  
  t.value += mess;
```

Event Object

To Associate Events with Code

- Third Approach – Specifying an event handler with a Function object.

```
// La funzione aggiungi_gestori installa un insieme di gestori di  
// eventi su ogni elemento di un form f, senza controllare se  
// l'elemento supporta tutti questi tipi di eventi
```

```
function aggiungi_gestori(f) {  
  var gestore_click = new Function("dettaglia(this, 'Click')");  
  var gestore_change = new Function("dettaglia(this, 'Change')");  
  var gestore_focus = new Function("dettaglia(this, 'Focus')");  
  var gestore_blur = new Function("dettaglia(this, 'Blur')");  
  var gestore_select = new Function("dettaglia(this, 'Select')");  
  var gestore_dbclick = new Function("dettaglia(this, 'dblClick')");
```

Event Object

To Associate Events with Code

- Third Approach – Specifying an event handler with a Function object.

```
for(var i = 0; i < f.elements.length; i++) {  
    var e = f.elements[i];  
    e.onclick    = gestore_click;  
    e.onchange   = gestore_change;  
    e.onfocus   = gestore_focus;  
    e.onblur     = gestore_blur;  
    e.onselect   = gestore_select;  
    e.ondblclick = gestore_dblclick;  
}
```

Event Object

To Associate Events with Code

- **Fourth approach** – `addEventListener(type, listener[, useCapture])`
- The `EventTarget.addEventListener()` method registers the specified listener on the `EventTarget` it is called on. The event target may be an `Element` in a document, the `Document` itself, a `Window`, or any other object that supports events
 - **Type** - A string representing the event type to listen for
 - **Listener** - the object that receives a notification when an event of the specified type occurs
 - **useCapture** - If true, `useCapture` indicates that the user wishes to initiate capture. After initiating capture, all events of the specified type will be dispatched to the registered listener before being dispatched to any `EventTarget` beneath it in the DOM tree.

Event Object

To Associate Events with Code

- **Fourth approach** – `addEventListener(type, listener[, useCapture])`

```
// Function to change the content of t2
function modifyText() {
    var t2 = document.getElementById("t2");
    if (t2.firstChild.nodeValue == "three") {
        t2.firstChild.nodeValue = "two";
    } else {
        t2.firstChild.nodeValue = "three";
    }
}
// add event listener to table
var el = document.getElementById("outside");
el.addEventListener("click", modifyText, false);
```




Events

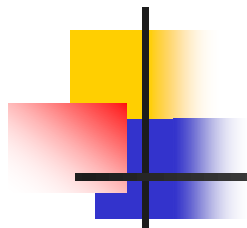
■ Main events

Attribute	The event occurs when...	IE	F	O	W3C
onblur	An element loses focus	3	1	9	Yes
onchange	The content of a field changes	3	1	9	Yes
onclick	Mouse clicks an object	3	1	9	Yes
ondblclick	Mouse double-clicks an object	4	1	9	Yes
onerror	An error occurs when loading a document or an image	4	1	9	Yes
onfocus	An element gets focus	3	1	9	Yes
onkeydown	A keyboard key is pressed	3	1	No	Yes



Events

Attribute	The event occurs when...	IE	F	O	W3C
onkeypress	A keyboard key is pressed or held down	3	1	9	Yes
onkeyup	A keyboard key is released	3	1	9	Yes
onmousedown	A mouse button is pressed	4	1	9	Yes
onmousemove	The mouse is moved	3	1	9	Yes
onmouseout	The mouse is moved off an element	4	1	9	Yes
onmouseover	The mouse is moved over an element	3	1	9	Yes
onmouseup	A mouse button is released	4	1	9	Yes
onresize	A window or frame is resized	4	1	9	Yes
onselect	Text is selected	3	1	9	Yes
onunload	The user exits the page	3	1	9	Yes



Events

- The events are applicable to specific elements
- For instance,
 - onBlur is applicable to only object window and all the elements of a form
 - onChange is applicable to only the fields <input>, <textarea> and <select> of a form
 - onMouseOut is applicable to elements links and areas
 - onResize is applicable to document, frame and window

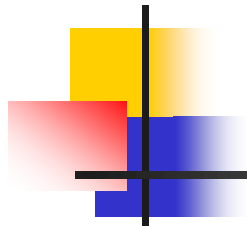




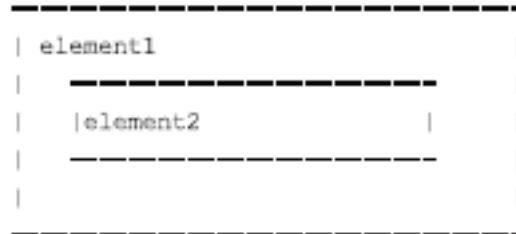
Events

```
<script type="text/javascript">
function setStyle(x)
{ document.getElementById(x).style.background="yellow"; }
function resetStyle(x)
{ document.getElementById(x).style.background="white";}
</script>
</head>
<body>
<div>
  First name: <input type="text" onfocus="setStyle(this.id)"
onblur="resetStyle(this.id)" id="fname"><br>
  Last name: <input type="text" onfocus="setStyle(this.id)"
onblur="resetStyle(this.id)" id="lname"></div>
</body>
```





Event order

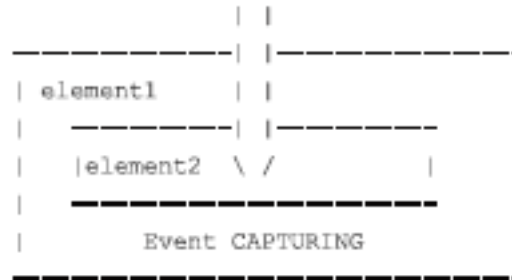


■ Two models

- Capturing – the event on element1 takes place first (Netscape)
- Bubbling – the event on element2 takes precedence (Microsoft)



Event order



- Event capturing
 - The event handler of element1 fires first, the event handler of element2 fires last

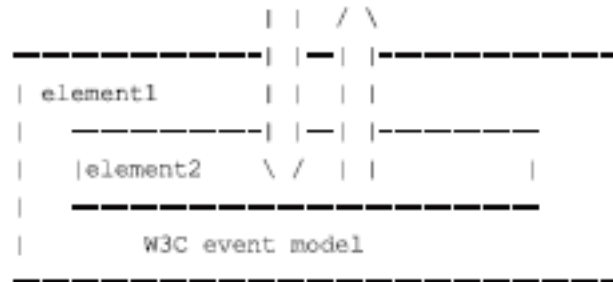
Event order



■ Event bubbling

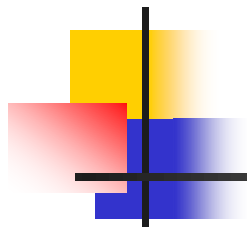
- The event handler of element2 fires first, the event handler of element1 fires last

Event order



■ W3C model

- W3C has decided to take a middle position in this struggle. Any event taking place in the W3C event model is first captured until it reaches the target element and then bubbles up again



Event Handler

- The Web developer can choose whether to register an event handler in the capturing or in the bubbling phase.
 - This is done through the `addEventListener()`
 - If its last argument is true the event handler is set for the capturing phase
 - If it is false the event handler is set for the bubbling phase





Event Handler

```
element1.addEventListener('click',doSomething2,true)
element2.addEventListener('click',doSomething,false)
```

If the user clicks on element2 the following happens:

1. The `click` event starts in the capturing phase. The event looks if any ancestor element of element2 has a `onclick` event handler for the capturing phase.
2. The event finds one on element1. `doSomething2()` is executed.
3. The event travels down to the target itself, no more event handlers for the capturing phase are found. The event moves to its bubbling phase and executes `doSomething()`, which is registered to element2 for the bubbling phase.
4. The event travels upwards again and checks if any ancestor element of the target has an event handler for the bubbling phase. This is not the case, so nothing happens.



Event Handler

```
element1.addEventListener('click',doSomething2,false)  
element2.addEventListener('click',doSomething,false)
```

Now if the user clicks on element2 the following happens:

-
1. The `click` event starts in the capturing phase. The event looks if any ancestor element of element2 has a `onclick` event handler for the capturing phase and doesn't find any.
 2. The event travels down to the target itself. The event moves to its bubbling phase and executes `doSomething()`, which is registered to element2 for the bubbling phase.
 3. The event travels upwards again and checks if any ancestor element of the target has an event handler for the bubbling phase.
 4. The event finds one on element1. Now `doSomething2()` is





Event Handler

- If you would like to stop the propagation
 - IE – `window.event.cancelBubble = true`
 - W3C model – `a.stopPropagation();`

```
function doSomething(e)
{
    if (!e) var e = window.event;
    e.cancelBubble = true;
    if (e.stopPropagation) e.stopPropagation();
}
```



Event Handler

■ Current Target

- During the capturing and bubbling phases the target does not change: it always remains a reference to element 2

```
element1.onclick = doSomething;  
element2.onclick = doSomething;
```

- How do you know which HTML element is currently handling the event?
- Target/srcElement don't give a clue, they always refer to element2
- In W3C has been added currentTarget (non in IE).
- You can however use the this keyword.

Event Object Properties

- When an event occurs, the browsers make an Event object available to the handlers
- The object provides information regarding the event
- Differences between browsers
 - Typically, the Event object is passed to the handler except for Internet Explorer
 - In IE, the Event object is made available as a property of the object Window, that is, a global variable
 - Some properties have different names
 - For instance, to find the reference to the element where the event is arisen:
 - target (in FireFox)
 - srcElement (in IE)

How to write a code independent of the browser for handling events

- To access the event object

- ```
function handler(e) {
 e=(!e) ? window.event : e;
```

If the browser is IE, the parameter `e` is not initialized. Thus, `e` has to be explicitly initialized to contain the Event object

- To refer to the object where the event is arisen

- ```
obj=(e.target != null) ? e.target : e.srcElement
```

Event Object Properties

Property	Description	IE	F	O	W3C
altKey	Returns whether or not the "ALT" key was pressed when an event was triggered	6	1	9	Yes
button	Returns which mouse button was clicked when an event was triggered	6	1	9	Yes
clientX	Returns the horizontal coordinate of the mouse pointer when an event was triggered	6	1	9	Yes
clientY	Returns the vertical coordinate of the mouse pointer when an event was triggered	6	1	9	Yes
ctrlKey	Returns whether or not the "CTRL" key was pressed when an event was triggered	6	1	9	Yes
metaKey	Returns whether or not the "meta" key was pressed when an event was triggered	6	1	9	Yes
relatedTarget	Returns the element related to the element that triggered the event	No	1	9	Yes
screenX	Returns the horizontal coordinate of the mouse pointer when an event was triggered	6	1	9	Yes
screenY	Returns the vertical coordinate of the mouse pointer when an event was triggered	6	1	9	Yes
shiftKey	Returns whether or not the "SHIFT" key was pressed when an event was triggered	6	1	9	Yes

Event Object Properties

Property	Description
bubbles	Returns a Boolean value that indicates whether or not an event is a bubbling event
cancelable	Returns a Boolean value that indicates whether or not an event can have its default action prevented
currentTarget	Returns the element whose event listeners triggered the event
eventPhase	Returns which phase of the event flow is currently being evaluated
target	Returns the element that triggered the event
timeStamp	Returns the time stamp, in milliseconds, from the epoch (system start or event trigger)
type	Returns the name of the event
isTrusted	Indicates whether or not the event was initiated by the browser or by a script

Event Object Events

Event Handlers can be used in three different ways to trigger a function

1. Link Events

Places an Event Handler as an attribute within an `` tag

For example:

```
<div>
```

```
<a href="#" onClick="alert('Ooo, do it again!');">
```

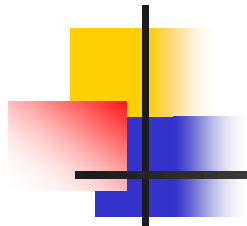
```
    Click on me!</a>
```

```
<a href="javascript:void()" onClick="alert('Ooo, do  
    it again!');"> Click on me! </a>
```

```
<a href="javascript:alert('Ooo, do it again!')" >  
    Click on me!</a>
```

```
</div>
```





Event Object (Events)

Note: No `<script>` tag.

- Anything that appears in the quotes of an `onClick` or an `onMouseOver` is automatically interpreted as JavaScript.
- `href="#"` tells the browser to look for the anchor `#`, but there is no anchor `"#"`, so the browser goes to the top of the page since it could not find the anchor.
- `<a href="javascript:void(' ')"` tells the browser not to go anywhere –
 - it "deadens" the link when you click on it.
 - `href="javascript:"` is the way to call a function when a link (hyperlink or an `HREFed` image) is clicked.

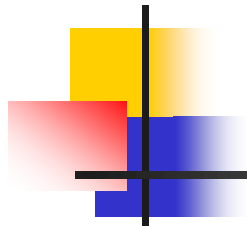




Event Object (Events)

2. Actions within forms

```
function checkField(fld){  
    if (fld.checkValidity() && fld.value>100)  
    {  
        alert("Correct number");  
        fld.style.backgroundColor="white";  
    }  
    else  
    { alert("Please enter a number greater than 100");  
      fld.value=null;  
      fld.style.backgroundColor="red";  
      setTimeout(function()  
      {document.getElementById('idname').focus();},1000);  
    }  
}
```



Event Object (Events)

```
function setStyle(fld)
{fld.style.backgroundColor="yellow"}
</script>
</head>
```

```
<body>
  <form id="frm1" action="form_action.asp">
    <p>Insert a Number: <input type="input" required
      pattern="^[0-9]+$" id="idname"
      onfocus="setStyle(this)"
      onchange="checkField(this)"> </p>
  </form>
</body>
</html>
```

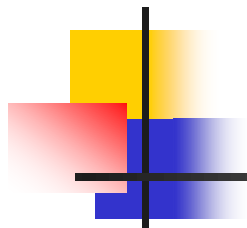




Event Object (Events)

3. Body onLoad and unload

- These Event Handlers are defined in the <body> or <frameset> tag of an HTML file and **are invoked when the document or frameset are fully loaded or unloaded.**
- If you set a flag within the onLoad Event Handler, other Event Handlers can test this flag to see if they can safely run, with the knowledge that the document is fully loaded and all objects are defined.
- Used **to check the visitor's browser** type and version, and load the proper version of the web page
- Used **to deal with cookies** that should be set when a user enters or leaves a page
- **onload** - allows launching a particular JavaScript function upon the completion of initially loading the document
- **onunload** - is triggered when the user "unloads" or exits the document



Event Object (Example)

```
<head>
<script type="text/javascript">
var loaded = "false";
function doit() {
    alert('Everything is "loaded" and loaded = ' + loaded);
}
function bye(){alert("Bye bye");}
</script>
</head>
```





Event Object (Example)

```
<body onload="loaded='true';" onunload="bye()">  
<form id="frm1" action="form_action.asp">  
<p>  
  <input type="button" value="Press me"  
    onClick="if (window.loaded) doit();">  
</p>  
</form>  
</body>
```



Examples (keydown event)

Move a figure on the page

```
<html>
  <head>
    <title> Events </title>
    <meta http-equiv="Content-Type"
      content="text/html; charset=us-ascii">
  </head>
  <body onload="set()">
    <div id="mydiv"
      style="position:absolute; left:450px; top:300px;">
      
    </div>
    <script type="text/javascript"
      src="./myscript12.js">
  </script>
</body>
</html>
```

Examples (keydown event)

Move a figure on the page

```
var elem = document.getElementById('mydiv');
function set() {document.onkeydown = onKeyHandler;}

function onKeyHandler(e) {
  e = (!e) ? window.event : e;           //Explorer -> !e
  var key = (e.which != null) ? e.which : e.keyCode; //Firefox -> e.which
  switch (key){
    case 37: move(-3, 0); break;          // left
    case 38: move(0, -3); break;          // up
    case 39: move(3, 0); break;           // right
    case 40: move(0, 3); break;           // down
    default: window.alert('stop'); }
}

function move(dx,dy) {
  elem.style.left = parseInt(elem.style.left) + dx + "px";
  elem.style.top  = parseInt(elem.style.top) + dy + "px"; }
```





Timer Events

setInterval("function()", delay)

Calls a function repeatedly, with a fixed time delay between each call to that function. The setInterval() method will continue calling the function until clearInterval() is called, or the window is closed. delay is expressed in millisec.

setTimeout("function()", delay)

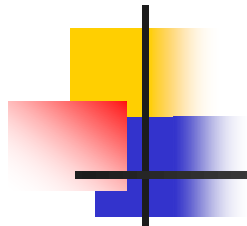
Calls a function after a specified number "delay" of milliseconds.

clearInterval(id_of_setinterval)

Clears a timer set with the setInterval() method. The ID value returned by setInterval() is used as the parameter for the clearInterval() method.

clearTimeout(id_of_settimeout)

Clears a timer set with the setTimeout() method. The ID value returned by setTimeout() is used as the parameter for the clearTimeout() method.

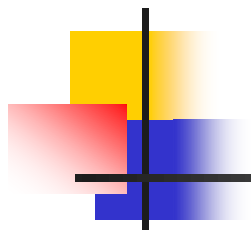


Timer Events

```
<!-- Head -->
<title>setInterval/clearInterval example</title>
<script type="text/javascript">
var intervalID;
function changeColor()
{   intervalID = setInterval(flashText, 1000);  }

function flashText()
{ var elem = document.getElementById("my_box");
  if (elem.style.color == 'red')
  {   elem.style.color = 'blue';  }
  else
  {   elem.style.color = 'red';  }
}
```



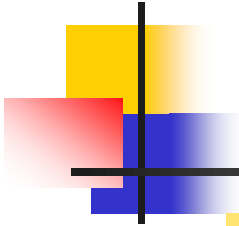


Timer Events

```
function stopTextColor()  
{ clearInterval(intervalID);}  
</script>  
</head>
```

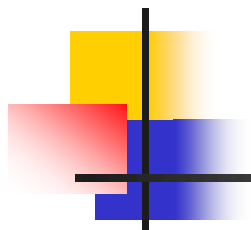
```
<body onload="changeColor();">  
<div id="my_box">  
<p>Hello World</p>  
<button onclick="stopTextColor();">Stop</button>  
</div>  
</body>  
</html>
```





Examples

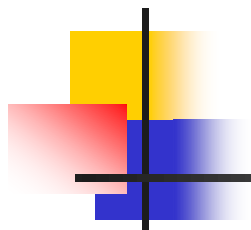




Dynamic Effects

- Dynamic effects are obtained by combining events and properties of the different objects
- Examples
 - Rollover – change of the images when the user passes over the images with the cursor of the mouse
 - Brief Animations
 - Sliding text
 - Gradual change of the background colour
 - Layer management





Rollover

```
<body>
  <div>
    
    
  </div>
  <script type="text/javascript" src="./myscript13.js"></script>
</body>
```





Rollover

```
//myscript13.js
var img1 = new Array(2); //
img1[0] = new Image(); img1[1] = new Image();
var img2 = new Array(2); img2[0]= new Image();
img2[1]= new Image();
img1[0].src="a.jpg"; img1[1].src="b.gif"; //default images
img2[0].src="b.gif"; img2[1].src="a.jpg";//rollover images

function modify(i,type) {
    if (type == "over")
        document.images[i].src = img2[i].src; //mouseover
    else    document.images[i].src = img1[i].src;//mouseout
}
```





Sliding Text

```
<style type="text/css">
#slidText { color : red; background:black}
</style>
</head>
<body onload="setInterval('slide()',100);">
  <form action='#'>
    <p> <input type="text"
      size="30"
      name="mytext"
      id = "slidText"
      style="border: solid;"> </p> </form>
    <script type="text/javascript" src="./myscript12.js"></script>
  </body>
```



Sliding Text

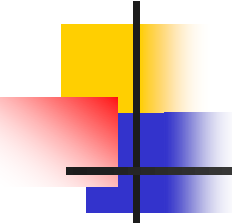
```
//myscript14.js  
var text = "Example of Sliding Text .....";  
function slide() {  
  var firstchar = text.charAt(0);  
  text = text.slice(1,text.length) + firstchar;  
  document.forms[0].mytext.value = text}
```



Managing Layers

- Layers allow designers to (partially) superimpose an content on another content
- The element HTML div permits managing the layers using the following properties;
 - **position**: identifies the position of the element in the page
 - Values: absolute, fixed and relative
 - **left, right, top, bottom**: identify the position of the left-top or the right-bottom margin with respect to the reference container.
 - **height and width**: specify the width and height of the layer in pixels
 - **z-index**: specifies the layer (positive integer) on the z-axis
 - **visibility**: determines whether the element is visible or not.
 - Values: hidden and visible

Managing Layers Example



```
<link rel="stylesheet" href="mystyleJS.css" type="text/css"
      media="screen">
</head>
<body>
  <h1> Properties of the levels </h1>
  <div id="div1" style= "position: absolute; left: 300px; top:150px;
    width:300px; height:150px; z-index: 1"> Red </div>
  <div id="div2" style= "position: absolute; left: 350px; top:200px;
    width:300px; height:150px; z-index: 2"> Blu </div>
  <div id="div3" style= "position: absolute; left: 400px; top:250px;
    width:300px; height:150px; z-index: 3"> Green </div>
```

Managing Layers Example



```
<div id="control">
```

```
<form action="#" name="mymodule" id="mymodule">
```

```
<p>
```

1) Insert the block:


```
<select name="idiv">
```

```
<option value="div1" selected> red </option>
```

```
<option value="div2"> blu </option>
```

```
<option value="div3"> green </option>
```

```
</select><br>
```

2) Insert the distance from the top:


```
<input type="text" size="5" name="top" value="150"><br>
```

```
//...
```

```
<input type="button" value="set" onclick="set()">
```

```
//...
```

```
<script type="text/javascript" src="./myscript16.js">
```

Managing Layers

Example (myscript15.js)

```
var ERROR = false;
function read(i) { value = 0;
  with (document.mymodule) {
    switch (i) {
      case 1: return idiv.value; break;
      case 2: value = top.value; break;
      case 3: value = left.value; break;
      case 4: value = width.value; break;
      case 5: value = height.value; break;
      case 6: value = level.value; }
    }
  value = parseInt(value);
  if ((value < 0) || isNaN(value))
  { window.alert("Error in the field n." + i);
    ERROR = true; } return value; }
```

Managing Layers

Example (myscript15.js)

```
function set() { var i = read(1);
                 var x = read(2);
                 var y = read(3);
                 var w = read(4);
                 var h = read(5);
                 var z = read(6);
                 if (!ERROR) { var mydiv = document.getElementById(i);
                               mydiv.style.top = x + "px";
                               mydiv.style.left = y + "px";
                               mydiv.style.width = w + "px";
                               mydiv.style.height = h + "px";
                               mydiv.style.zIndex = z + ""; }
                 ERROR = false;}
```


Managing Layers

Example (mystyleJS.js)

```
div {
    border:        outset;
    font-size:     xx-large;
    font-weight:   bolder;
    overflow:      hidden;}
#div1 { background-color: red;}
#div2 { background-color: blue; }
#div3 { background-color: green; }
#control {
    background-color: azure; z-index:        0;
    width:            15em; padding:         5px;
    font-size:        medium; font-weight:   normal;
}
```

Validation of data inserted into a Form

- JavaScript allows verifying the data inserted into a form directly on the client-side
- Using handlers of appropriate events (onClick and onBlur), the data can be analysed and messages can be sent to the user
 - onClick – when the form is completed and sent to the server
 - onBlur – when the focus is passed on another element
- Typically, the verification is performed only on the input text fields

Validation of data inserted into a Form

- Two examples
 - Validation of data
 - Assessment of the answers to a quiz

Validation

Name (*):

Surname (*):

Age:

City, :

Zip Code (*):

Quiz

1) What is the keyword in Javascript for defining a function?

- ☐ var
☐ function
☐ script

2) What is not a valid comment in Javascript?

- ☐ *...*\n
☐ */.../*
☐ //...
☐ //...//

VERIFY

Validation of data inserted into a Form

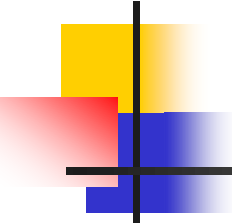
Validation of the data inserted by using the previous form

```
var fields = document.getElementsByTagName("input");  
var shouldBeALPHANUMERICAL = [0, 1, 4];  //(1)  
var shouldBeNUMERICAL      = [2, 4];  
var shouldBeALPHABETICAL   = [0, 1, 3];  
var shouldBe5NUMBERS       = [4];  
var MESSAGES = [ "The following field does not have valid symbols: ",  
                  "The following field is not exclusively numerical: ",  
                  "The following field must have five digits: ",  
                  "The following field must have only letters: ",  
                  "...",  
                  "OK"];
```

Validation of data inserted into a Form

```
var existALPHANUMERICAL = /\w/;
var existNONALPHANUMERICAL = /\W/;
var existNONNUMERICAL = /\D/;
var existNUMERICAL = /\d/;
var exist5NUMERICAL = /^\\d{5}$/;
function error(idmess, field)
{ window.alert(MESSAGES[idmess] + field.name);
  field.focus(); field.select();
}
```

Validation of data inserted into a Form



```
function isTrue(COND, ELEM, BOOL, MESS) {  
  for (var i=0; i<ELEM.length; i++) {  
    var j = ELEM[i];  
    field = fields[j];  
    if (COND.test(field.value) == BOOL) {  
      error(MESS, field);  
      return true;  
    }  
  }  
  return false;  
}
```

Validation of data inserted into a Form

```
function validate() {  
  if (isTrue(existALPHANUMERICAL, shouldBeALPHANUMERICAL, false, 0))  
    return;  
  if (isTrue(existNONALPHANUMERICAL, shouldBeALPHANUMERICAL, true, 0))  
    return;  
  if (isTrue(existNONNUMERICAL, shouldBeNUMERICAL, true, 1))  
    return;  
  if (isTrue(exist5NUMERICAL, shouldBe5NUMBERS, false, 2)) return;  
  if (isTrue(existNUMERICAL, shouldBeALPHABETICAL, true, 3)) return;  
  if (isTrue(existNONALPHANUMERICAL, shouldBeALPHABETICAL, true, 3))  
    return;  
  window.alert(MESSAGES[MESSAGES.length-1]);  
}
```





Assessment of the answers to a quiz

```
<form action="#" name="mymodule" id="mymodule">
<script type="text/javascript">
  <!--
  for (var i=0; i<Queries.length; i++)
  {
    document.writeln("<div>" + (i+1) + ") "+ Queries[i] + "<br>");
    for (var j=0; j<Options[i].length; j++)
    document.writeln("<input type='radio' name='q' + i + '>' +
                      Options[i][j] + "<br>");
    document.writeln("<hr></div>");
  }
  -->
</script>
<p>
<input type="button" value="VERIFY" onclick="check()"></p>
```




Assessment of the answers to a quiz

```
//myscript18.js
```

```
var Queries = [  
  "What is the keyword in Javascript for defining a function?",  
  "What is not a valid comment in Javascript?",  
  "What is the handler for the 'loss of active state' event?",  
  "Which operator can be used to instance an object?",  
  "To periodically call a function you use the method:"];  
  
var Options = [ ["var","function","script"],  
  ["\\*...*\\","*/.../*","//...","//...//"],  
  ["onFocus","onBlur","onClick"],  
  ["create","new","add"],  
  ["window.setInterval","window.setTimeout","date.setTime"]];
```



Assessment of the answers to a quiz

```
//myscript18.js
```

```
var Answers = [ 0,1,0, 0,1,0,0, 0,1,0, 0,1,0, 1,0,0];
```

```
function check() { var score = 0;
```

```
    var answers =
```

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

```
    for (var i = 0; i < answers.length-1; i++)
```

```
        if (answers[i].checked)
```

```
            if (Answers[i]==1) score++;
```

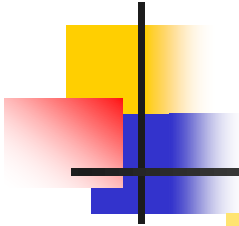
```
            else score--;
```

```
    window.alert("Your score is: " + score);
```

```
    document.mymodule.reset();
```

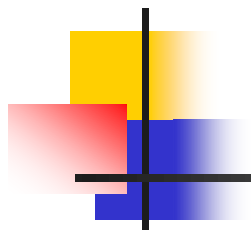
```
}
```





Web Storage





Cookie

- A **cookie is a piece of text** stored by a user's web browser.
- A cookie can be used for
 - authentication,
 - storing site preferences,
 - shopping cart contents,
 - the identifier for a server-based session.
- A **cookie consists of one or more name-value pairs containing bits of information**, which may be encrypted for information privacy and data security purposes.
- The cookie is sent as an HTTP header by a web server to a web browser and then sent back unchanged by the browser each time it accesses that server.





Cookie

- Cookies may be set by the server with or without an expiration date.
 - Cookies without an expiration date exist until the browser terminates
 - Cookies with an expiration date may be stored by the browser until the expiration date passes.
 - Users may also manually delete cookies in order to save space or to avoid privacy issues.
- Cookies may be used to remember the information about the user who has visited a website in order to show relevant content in the future.
 - For example a web server may send a cookie containing the username last used to log in to a web site so that it may be filled in for future visits.



Cookie

```
function setCookie(name, value, expires, path, domain,
    secure )
{ // set time, it's in milliseconds
var today = new Date();
if ( expires ) { expires = expires * 1000 * 60 * 60 * 24;}
var expires_date = new Date( today.getTime() + expires);
document.cookie = name + "=" +escape( value ) +
( ( expires ) ? ";expires=" + expires_date.toGMTString() : "" )
+ ( ( path ) ? ";path=" + path : "" ) +
( ( domain ) ? ";domain=" + domain : "" ) +
( ( secure ) ? ";secure" : "" );
}
```



Cookie

Parameters:

- **name** – name of the cookie
- **value** – value of the cookie
- **expire** – the time the cookie expires
- **path** – path on the server in which the cookie will be available on. If set to '/', the cookie will be available within the entire domain.
- **domain** - The domain that the cookie is available to. To make the cookie available on all subdomains of example.com (including example.com itself) then you'd set it to '.example.com'.
- **secure** - Indicates that the cookie should only be transmitted over a secure HTTPS connection from the client. When set to **TRUE**, the cookie will only be set if a secure connection exists.



Cookie

```
function getCookie( check_name ) {  
    // first we'll split this cookie up into name/value pairs  
    // note: document.cookie only returns name=value, not the other  
    // components  
    var a_all_cookies = document.cookie.split( ';' );  
    var a_temp_cookie = "";  
    var cookie_name = "";  
    var cookie_value = "";  
    var b_cookie_found = false; // set boolean t/f default f  
  
    for ( i = 0; i < a_all_cookies.length; i++ )  
    {  
        // now we'll split apart each name=value pair  
        a_temp_cookie = a_all_cookies[i].split( '=' );  
  
        // and trim left/right whitespace while we're at it  
        cookie_name = a_temp_cookie[0].replace(/^\s+|\s+$/g, "");
```




Cookie

```
// if the extracted name matches passed check_name
if ( cookie_name == check_name )
{
    b_cookie_found = true;
    // we need to handle case where cookie has no value
    // but exists (no = sign, that is):
    if ( a_temp_cookie.length > 1 )
    { cookie_value =
unescape(a_temp_cookie[1].replace(/^\s+|\s+$/g, " ) );
    }
// note: if cookie is initialized but no value, null is returned
    return cookie_value;
}
a_temp_cookie = null;  cookie_name = "";
}
```



Cookie

```
if ( !b_cookie_found )
{ return null;}
}

function deleteCookie( name, path, domain ) {
name=prompt('Please enter the cookie to remove:', "");
if ( getCookie( name ) ) document.cookie = name + "=" +
( ( path ) ? ";path=" + path : "" ) +
( ( domain ) ? ";domain=" + domain : "" ) +
";expires=Thu, 01-Jan-1970 00:00:01 GMT";
}

function deleteC()
{ name=prompt('Please enter the cookie to remove:', "");
deleteCookie(name, '/', "");
}
```



Cookie

```
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,100,'/');   }
  }
}

....
<body onLoad="checkCookie()">
<p><a href="javascript:deleteC()">
  Click to delete the cookie</a></p>
</body>
```



Session and Local Storage

- Cookies are extremely limited in size. Generally, only about 4KB.
- Cookies are transmitted back and forth from server to browser on every request scoped to that cookie.
 - security risks
 - network bandwidth consumption
- The same results could be achieved without involving a network or remote server and storing up to a few megabytes.
 - **sessionStorage** – values survive either across page loads in a single window or tab
 - **localStorage** – values survive across browser restarts

Session and Local Storage Browser Support

```
function checkStorageSupport() {  
  //sessionStorage  
  if (window.sessionStorage) {  
    alert('This browser supports sessionStorage');  
  } else {  
    alert('This browser does NOT support sessionStorage');  
  }  
  //localStorage  
  if (window.localStorage) {  
    alert('This browser supports localStorage');  
  } else {  
    alert('This browser does NOT support localStorage');  
  }  
}
```

Session and Local Storage Browser Support

- Setting a value

```
window.sessionStorage.setItem('myFirstKey', 'myFirstValue');
```

- Retrieve a value

```
window.sessionStorage.getItem('myFirstKey')
```

- `setItem` and `getItem` calls can be avoided entirely by simply setting and retrieving values corresponding to the key-value pair directly on the `sessionStorage` object.

```
window.sessionStorage.myFirstKey = 'myFirstValue';  
window.sessionStorage.myFirstKey;
```

Session and Local Storage

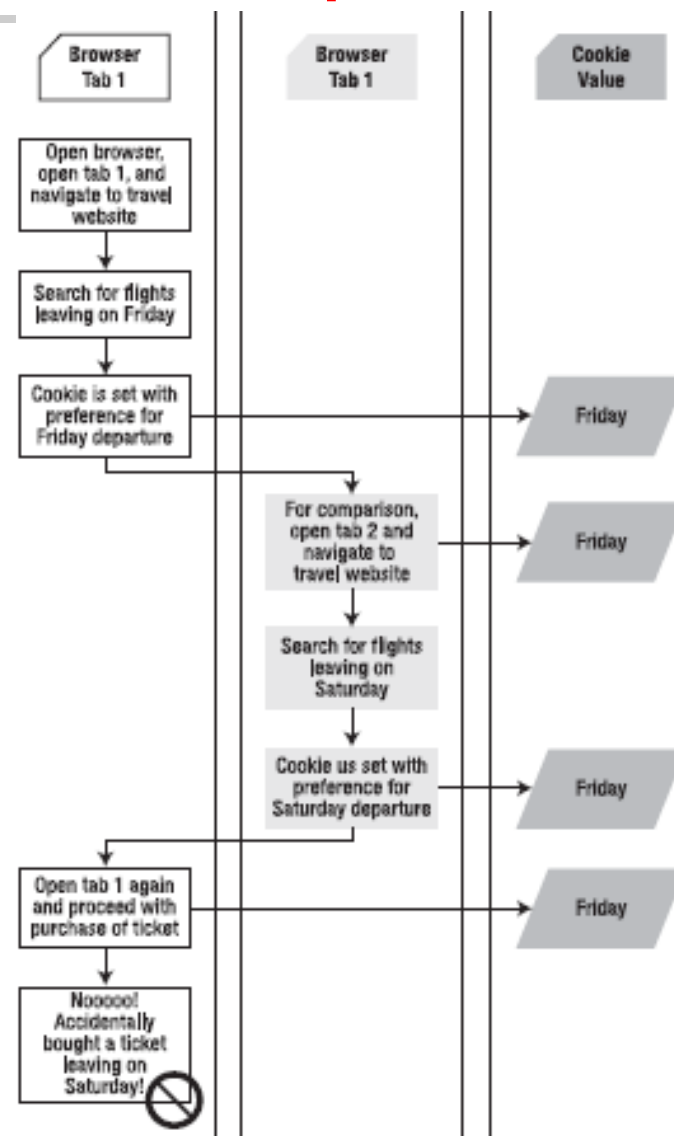
How long do the values persist?

- **Objects in sessionStorage**
 - These objects will persist as long as the browser window (or tab) is not closed.
 - Session storage is perfect for short-lived processes that would normally be represented in wizards or dialogs.
 - Solves a problem that has plagued many web-applications: scoping of values.

Session and Local Storage

How long do the values persist?

- The figure shows the problem of value scoping with cookies
- Solution: sessionStorage allows temporary values like a departure date to be saved across pages that access the application but not leak into other windows where the user is also browsing for flights. Therefore, those preferences will be isolated to each window where the corresponding flights are booked.





Local Versus Session Storage

- **localStorage** – values survive across browser restarts

sessionStorage	localStorage
Values persist only as long as the window or tab in which they were stored.	Values persist beyond window and browser lifetimes.
Values are only visible within the window or tab that created them.	Values are shared across every window or tab running at the same origin.

- The storage interface
interface Storage {
 readonly attribute unsigned long length;
 getter DOMString key(in unsigned long index);
 getter any getItem(in DOMString key);
 setter creator void setItem(in DOMString key, in any data);
 deleter void removeItem(in DOMString key);
 void clear();
};



Storage

- **length** – specifies how many key-value pairs are currently stored in the storage object.
- **key(index)** – allows retrieval of a given key.
- **getItem(key)** – is one way to retrieve the value based on a given key.
- **setItem(key, value)** – will put a value into storage under the specified key name, or replace an existing value if one already exists under that key name.
- **removeItem(key)** – If a value is currently in storage under the specified key, this call will remove it. If no item was stored under that key, no action is taken.
- **clear()** – removes all values from the storage list.

Communicating Web Storage Updates

- To register to receive the storage events of a window's origin, simply register an event listener, for example:

```
window.addEventListener("storage", displayStorageEvent, true);
```

- The StorageEvent Interface

```
interface StorageEvent : Event {  
  readonly attribute DOMString key;  
  readonly attribute any oldValue;  
  readonly attribute any newValue;  
  readonly attribute DOMString url;  
  readonly attribute Storage storageArea;  
};
```



Storage Event

- **key** – contains the key value that was updated or removed in the storage.
- **oldValue** – contains the previous value corresponding to the key before it was updated, and the **newValue** contains the value after the change.
- **url** – points to the origin where the storage event occurred.
- **storageArea** – provides a convenient reference to the `sessionStorage` or `localStorage` where the value was changed. This gives the handler an easy way to query the storage for current values or make changes based on other storage changes.



Display Content of a Storage Event

```
// display the contents of a storage event
function displayStorageEvent(e) {
  var logged = "key:" + e.key + ", newValue:" + e.newValue + ",
  oldValue:" +
  e.oldValue + ", url:" + e.url + ", storageArea:" + e.storageArea;
  alert(logged);
}

// add a storage event listener for this origin
window.addEventListener("storage", displayStorageEvent, true);
```



Example

```
<body>
  Key: <input type="text" id="key"><br>
  Value: <input type="text" id="value"><br>
  <input type="button" id="add" value="Add to Storage">&nbsp;
  <input type="button" id="remove" value="Remove from
Storage">&nbsp;
  <input type="button" id="clear" value="Clear Storage"><br>
  Contents of Local Storage:<br>
  <div id="content"></div>
</body>
```

Note that the example does not work correctly in Chrome if the page runs from the file protocol (<file:///>).