Client-side programming with JavaScript

Laura Farinetti
Dipartimento di Automatica e Informatica
Politecnico di Torino
laura.farinetti@polito.it

Summary

- Introduction
- Language syntax
- Functions
- Objects
- Events
- The HTML Document Object Model (DOM)
- Examples

What and why JavaScript?

- JavaScript is a lightweight, interpreted programming language with object-oriented capabilities that allows you to build interactivity into otherwise static HTML pages
  - JavaScript made its first appearance in Netscape 2.0 in 1995 with the name “LiveScript”
  - Later standardized by ECMA (www.ecma.ch): ECMAScript
- JavaScript is one of the 3 languages all web developers must learn
  - HTML to define the content of web pages
  - CSS to specify the layout of web pages
  - JavaScript to program the behavior of web pages

What can JavaScript do?

- JavaScript can change HTML content
- JavaScript can change HTML attributes
- JavaScript can change HTML styles (CSS)
- JavaScript can validate data

- http://www.w3schools.com/js/js_intro.asp

JavaScripts

- A JavaScript consists of JavaScript statements placed within the <script>... </script> HTML tags in a web page
- The <script> tag containing JavaScript code can be placed anywhere in a web page
  - In the head or the body section

prova.html

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

Where to embed JavaScript code?

- In the head section
  - Scripts to be executed when they are called, or when an event is triggered, go in the head section
  - When you place a script in the head section, you will ensure that the script is loaded before anyone uses it
- In the body section
  - Scripts to be executed when the page loads go in the body section
  - When you place a script in the body section it generates the content of the page

JavaScript functions and events

- Functions are usually defined in the head section
- Functions can be executed when an event occurs, e.g. when the user clicks a button

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

Example

- JavaScript can change HTML content

```
<html>
<head>
<script>
function myFunction() {
    document.getElementById("demo").innerHTML =
       "... e vivo a Torino."; }
</script>
</head>

<body>
<h1>JavaScript</h1>
<p id="demo">Mi chiamo Andrea Rossi ...</p>
<button type="button" onclick="myFunction()">Prova</button>
</body>
</html>
```

External JavaScripts

- Scripts can be placed in external files too
  - Useful when the same code is used in many different web pages
  - Can be called in <head> or <body>
- JavaScript files: extension .js

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

JavaScript display possibilities

- JavaScript can “display” data in different ways
  - Writing into an alert box: window.alert()
  - Writing into the HTML output: document.write()
  - Writing into an HTML element: innerHTML
  - Writing into the browser console: console.log()

  - http://www.w3schools.com/js/js_output.asp

JavaScript display possibilities

- Using document.write() after an HTML document is fully loaded deletes all existing HTML
  - document.write() is useful only for testing purposes

```
<!DOCTYPE html>
<html>
<body>

<h1>Esempio</h1>
<p>Quanto fa 5 + 6 ?</p>

<button type="button"
   onclick="document.write(5 + 6)">Prova</button>

</body>
</html>
```

JavaScript display possibilities

- To access an HTML element, JavaScript can use the document.getElementById(id) method
- The id attribute defines the HTML element
- The innerHTML property defines the HTML content

```
<!DOCTYPE html>
<html>
<body>

<h1>Esempio</h1>
<p>Quanto fa 5 + 6 ?</p>
<p id="demo"></p>
<button type="button"
   onclick="document.getElementById('demo').innerHTML
           = 5 + 6;">Prova</button>
</body>
</html>
```

JavaScript display possibilities

- Example, with the predefined function Date()

```
<!DOCTYPE html>
<html>
<body>
<h1>Esempio</h1>
<button type="button" onclick=
   "document.getElementById('demo').innerHTML = Date()">
   Premi qui per sapere data e ora</button>
<p id="demo"></p>
</body>
</html>
```

What can JavaScript do?

- Generate dialog boxes
- Redirect a page
- Open new browser windows (pop-ups)
- Intercept mouse events
  - Clicks on links, buttons, ...
  - Mouse-overs, …
- Read user input in forms
- Modify HTML pages
  - Add/remove content
  - Change images
  - Modify form controls

What to know…

- JS variables and expressions
- JS language constructs (if, while, ...)
- JS objects
  - The most important built-in objects
- Interaction with the user
  - Mouse, keyboard
- Interaction with the browser
  - Windows, pages
- Interaction with the page: the Document Object Model

JavaScript syntax

- Similar to C language (and Ruby too)
  - Choice, loops and other constructs are the same
  - Blocks are delimited by { }
  - Most operators are identical
  - Variables are different, however, …
- JavaScript is a case-sensitive language
- Semi-colons (at the end of a line) can be omitted
- Comments:

```
<script>
// This is a comment. It is similar to comments in C++
/*
 * This is a multiline comment in JavaScript
 * It is very similar to comments in C Programming
 */
</script>
```

JavaScript data types and variables

- Three primitive data types
  - Numbers (123, 120.50, …) – no distinction between integers are real numbers
  - Strings of text ("This text string", …)
  - Booleans (true or false)
- A composite data type known as “object”
- In JavaScript all variables must be declared before their use
- Data types are converted as needed

```
<script>
  var money;
  var x;
  var y = 10;
  var z = "Hello!";
  var one, two, three;
  var d = new Date();  //object
</script>
```

Main Javascript operators

- Numeric operators
  - + - * / %
- Increment operators
  - ++ --
- Assignment operators
  - = += -= *= /= %=
- String operator
  - + (concatenation)
- Comparison operators
  - == (same value) === (same value and same type)
  - != > < >= <=
- Boolean and Logic operators
  - && (logical "and") || (logical "or") ! (logical "not")

Choice statements

```
if (condition)
{
   ...code...
}
```

```
if (condition)
{
   ...code if true...
}
else
{
   ...code if false...
}
```

```
if (condition1)
{
   ...code if 1 true...
}
else if (condition2)
{
   ...code if 2 true...
}
else
{
   ...if both false...
}
```

Choice statements

```
switch(n)
{
  case 1:
    code block 1
    break

  case 2:
    code block 2
    break

  default:
    code to be executed if n is
    different from case 1 and 2
}
```

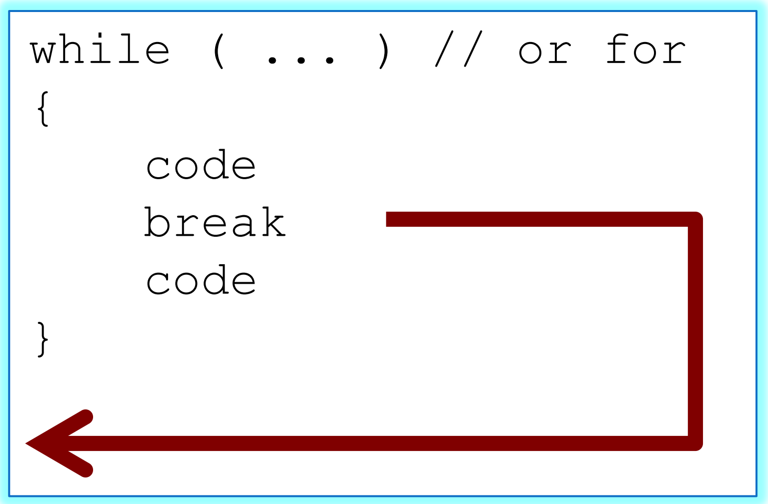
Loop statements

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

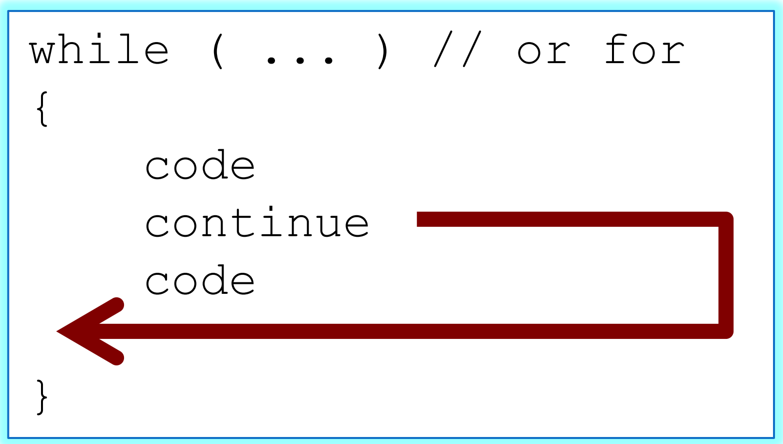
```
while ( condition_is_true )
{
    code to be executed
}
```

```
do {
    code to be executed
} while ( condition_is_true )
```

Loop statements



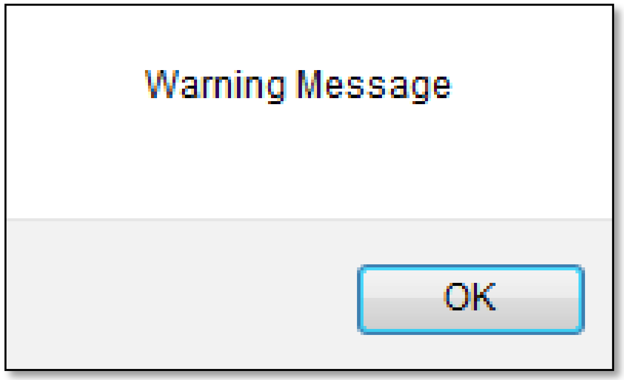
```
while ( ... ) // or for
{
     code
     break
     code
}
```



```
while ( ... ) // or for
{
     code
     continue
     code

}
```

Basic interaction methods

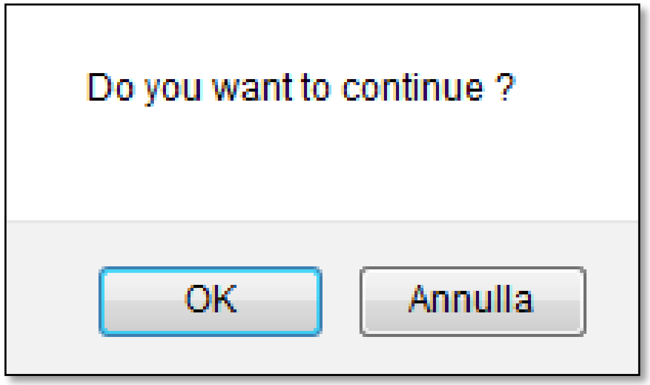
- Alert dialog box
  - OK to confirm
- Mostly used to give a warning message to the users


Warning Message

```
<head>
<script type="text/javascript">
<!--
   alert("Warning Message");
//-->
</script>
</head>
```

Basic interaction methods

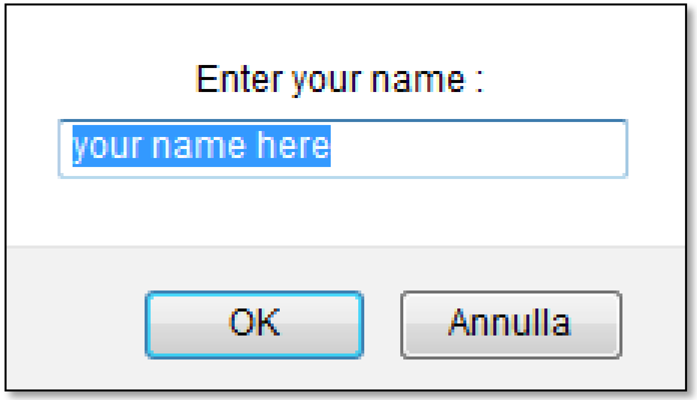
- Confirmation dialog box
  - OK, cancel
  - True if user clicks on OK
- Mostly used to take user's consent on any option


Do you want to continue ?

```
<script type="text/javascript">
 var retVal = confirm("Do you want to continue ?");
  if( retVal == true ){
    alert("User wants to continue!");
  }else{
    alert("User does not want to continue!");
  }
</script>
```

Basic interaction methods

- Prompt dialog box
  - Returns a string with the text written by the user
  - Returns null if user clicks on Cancel
- Used to get user input


Enter your name :

your name here

```
<script type="text/javascript">
<!--
   var retVal = prompt("Enter your name : ",
     "your name here");
   alert("Hello " +  retVal );
//-->
</script>
```

Functions

- Function definition

```
function functionname(var1,var2,...,varX)
{
      some code
}
```

  - No parameters:

```
function functionname()
{
      some code
}
```

- A function may return a value to its caller by executing the return statement
  - return value ;
  - The value may be of any type (boolean, numeric, string, ...)

Example

```
<html>
<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>
</body>
</html>
```

Objects in JavaScript

- An object is a complex data type characterized by
- A current value
  - Sometimes the internal value is “hidden”
- A set of properties
  - Various values that be read, associated in some way to the object value
  - Some values that may be written, that modify in some way the object value
- A set of methods
  - Operations (with parameters) that can be asked to the object

Example

```
<html>
<head>
<title>User-defined objects</title>
<script type="text/javascript">
  var book = new Object();   // Create the object
  book.subject = "Perl"; // Assign properties to the object
  book.author  = "Mohtashim";
</script>
</head>
<body>
<script type="text/javascript">
  document.write("Book name is: " + book.subject + "<br>");
  document.write("Book author is: " + book.author + "<br>");
</script>
</body>
</html>
```

JavaScript native objects

- JavaScript has several built-in objects
  - Accessible anywhere in a program
  - Work the same way in any browser running in any operating system
- List of native objects
  - JavaScript Number Object
  - JavaScript Boolean Object
  - JavaScript String Object
  - JavaScript Array Object
  - JavaScript Date Object
  - JavaScript Math Object
  - JavaScript RegExp Object

The String object

- Strings are used to store and manipulate sequences of characters
- The only property is
  - .length (the number of characters in the string)
- Many general methods
  - .charAt(), .concat(), .indexOf(), .localeCompare(), .match(), .replace(), .search(), .slice(), .split(), .substr(), .substring(), .toLowerCase(), .toUpperCase(), .toString(), .valueOf(), …
- Many methods specific for writing HTML

String methods for HTML formatting

- Methods that returns a copy of the string wrapped inside the appropriate HTML tag
  - Warning: not standard methods, may not work as expected in all browsers
- List of main methods
  - .big(), .small(), .italic(), .bold(), .fixed(), .sub(), .sup()
  - .fontcolor(c), .fontsize(s)
  - .anchor(“name”), .link(“url”)

```
<script>
    var str = "Hello World!";
    document.write(str);
    document.write("<br />");
    str = str.fontcolor("red");
    document.write(str + "<br/>");
    str = str.fontsize(7);
    document.write(str);
</script>
```

http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_str_style

Example

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to create an HTML link around a string.</p>

<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
    var txt = document.getElementById("demo").innerHTML;
    txt2 = txt.link("chap10.html");
       document.getElementById("demo").innerHTML = txt2;
}
</script>

<p id="demo">Chapter 10</p>

</body>
</html>
```

JavaScript event model

- Events are "things" that happen in a web page
- Some examples:
  - A web page has finished loading
  - An input field was changed
  - A button was clicked
- Events' categories
  - User interaction (click, move mouse, …)
  - Browser actions (load page, …)
- In Javascript, an event handler can be attached to most events
  - Any Javascript function
  - The Javascript interpreter calls the function anytime the event is generated

JavaScript event model

- Some common events
  - onclick, onchange, onmouseover, onmouseout, onkeydown, onload, onsubmit, onreset, onselect, …
- All events
  - http://www.w3schools.com/jsref/dom_obj_event.asp
- Examples:

```
<head>
<script>
function mymessage() {
    alert("This message was triggered from the onload event"); }
</script>
</head>
<body onload="mymessage()">
</body>
```

JavaScript event model

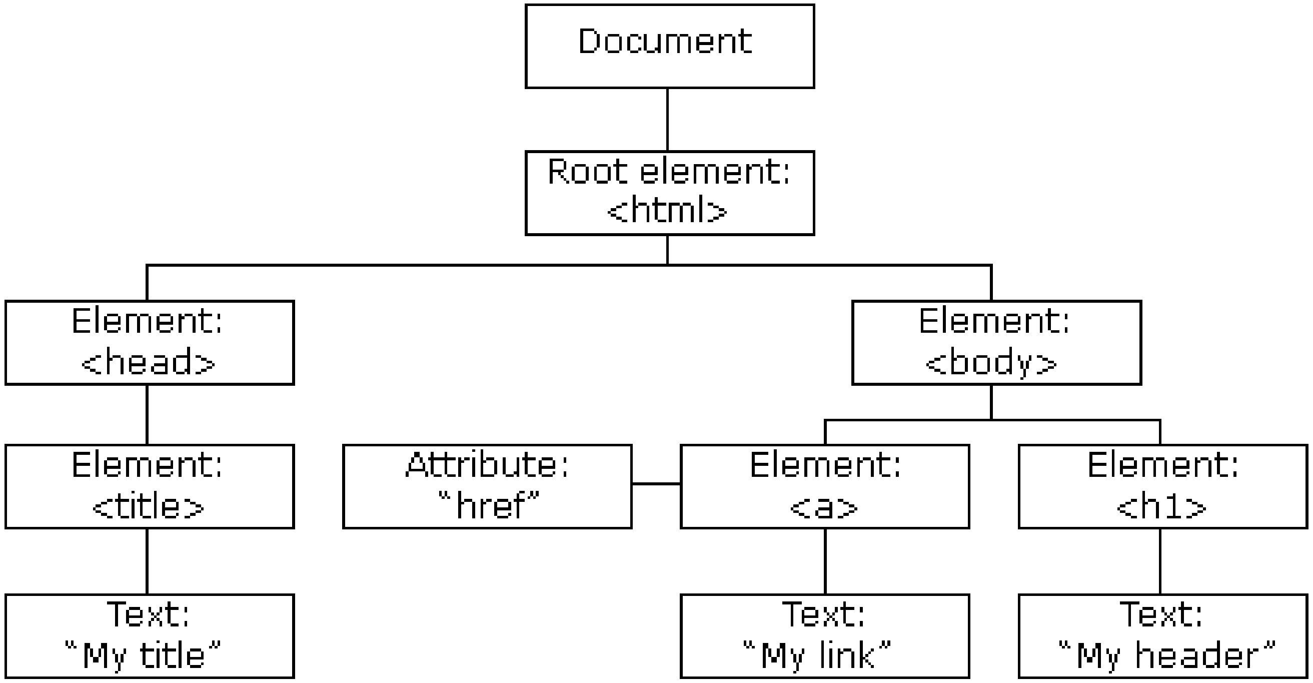
- Examples:

```
<body>
<button onclick="getElementById('demo').innerHTML=Date()">What
time is it?</button>
<p id="demo"></p>
</body>
```

```
<head>
<script>
function myFunc() {
    var x = document.getElementById("fname");
    x.value = x.value.toUpperCase(); }
</script>
</head>
<body>
Enter your name: <input type="text" id="fname" ochange="myFunc()">
<p>When you leave the input field, a function is triggered which
transforms the input text to upper case.</p>
</body>
```

HTML Document Object Model (DOM)

- Defines a standard way for accessing and manipulating HTML documents
- Objects are organized in a hierarchy: an HTML document is a tree-structure (a node tree), with elements, attributes, and text



HTML Document Object Model (DOM)

- The object model enables JavaScript to create dynamic HTML
  - Can change all the HTML elements in the page
  - Can change all the HTML attributes in the page
  - Can change all the CSS styles in the page
  - Can remove existing HTML elements and attributes
  - Can add new HTML elements and attributes
  - Can react to all existing HTML events in the page
  - Can create new HTML events in the page

- Tool: DOM inspector (for most browser)

The DOM structure

- The entire document is a <u>document</u> node
    - If you want to access objects in an HTML page, you always start with accessing the document object
- Every HTML tag is an <u>element</u> node
- The texts contained in the HTML elements are <u>text</u> nodes
- Every HTML attribute is an <u>attribute</u> node
- Comments are <u>comment</u> nodes
- Nodes have a hierarchical relationship to each other

With Dom you can …

- Find HTML elements
- Change HTML content
- Change the value of an attribute
- Change the HTML style
- Add or delete HTML elements

Find HTML elements

- Find an elements by its id
  - var x = document.getElementById("intro");
- Find elements by tag name
  - var x = document.getElementsByTagName("p");
  - Returns all the elements with that tag
- Find elements by class name
  - var x = document.getElementsByClassName("intro");
  - Returns all the elements of that class

- Example: finds the element with id="main", and then finds all <p> elements inside "main"

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

Change HTML content

- Easiest way: the innerHTML property

```
<!DOCTYPE html>
<html>
<body>

<h1 id="header">Old Header</h1>

<script>
var element = document.getElementById("header");
element.innerHTML = "New Header";
</script>

<p>"Old Header" was changed to "New Header"</p>

</body>
</html>
```

Change the value of an attribute

- Syntax
  - document.getElementById(id).attribute=new value
- Example

```
<!DOCTYPE html>
<html>
<body>
<img id="image" src="smiley.gif" width="160"
height="120">
<script>
document.getElementById("image").src = "landscape.jpg";
</script>
<p>The original image was smiley.gif, but the script
changed it to landscape.jpg</p>
</body>
</html>
<html>
```

http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_image

Change HTML style

- Syntax
  - document.getElementById(id).style.property=new style
- Example

```
<!DOCTYPE html>
<html>
<body>
<p id="p1">Hello World!</p>
<p id="p2">Hello World!</p>
<script>
document.getElementById("p2").style.color = "blue";
document.getElementById("p2").style.fontFamily = "Arial";
document.getElementById("p2").style.fontSize = "larger";
</script>
<p>The paragraph above was changed by a script.</p>
</body>
</html>
```

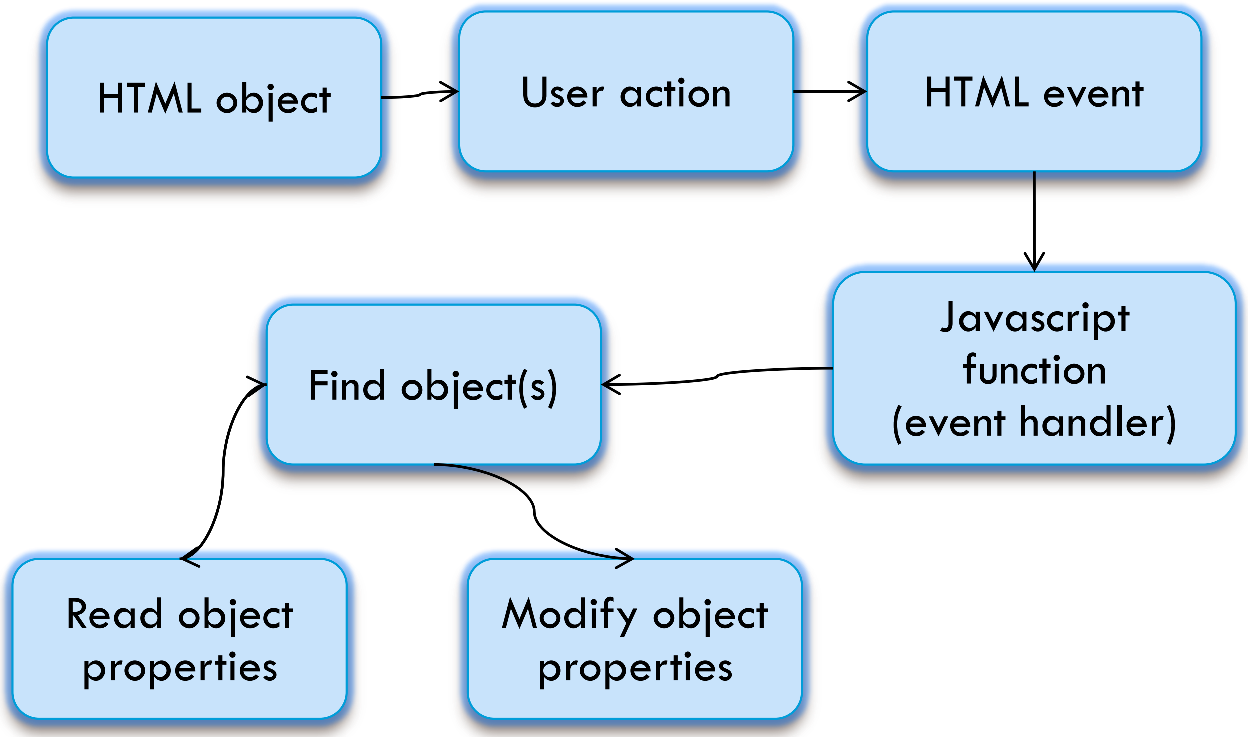
Add and delete elements

- Create an HTML element
  - document.createElement()
- Remove an HTML element
  - document.removeChild()
- Add an HTML element
  - document.appendChild()
- Replace an HTML element
  - document.replaceChild()

- Need to know node relationships, and how to navigate in the DOM tree

DOM and events

- The HTML DOM allows to execute code when an event occurs
- Control sequence



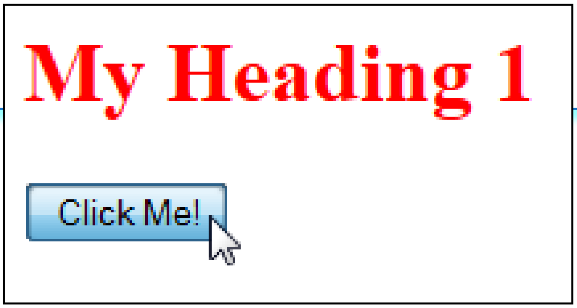
Reacting to events

- JavaScript code can be executed when an event occurs
- Simplest way: to assign events to HTML elements you can use event attributes (e.g onclick, onload, onchange, …)
  - event=JavaScript code
  - Javascript code can be an expression or a function call
- Alternative: write event handlers
  - addEventListener() method
  - Advantages: you can add many event handlers of the same type to one element, i.e two "click" events.
- the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.

Reacting to events

- Example with JavaScript expression

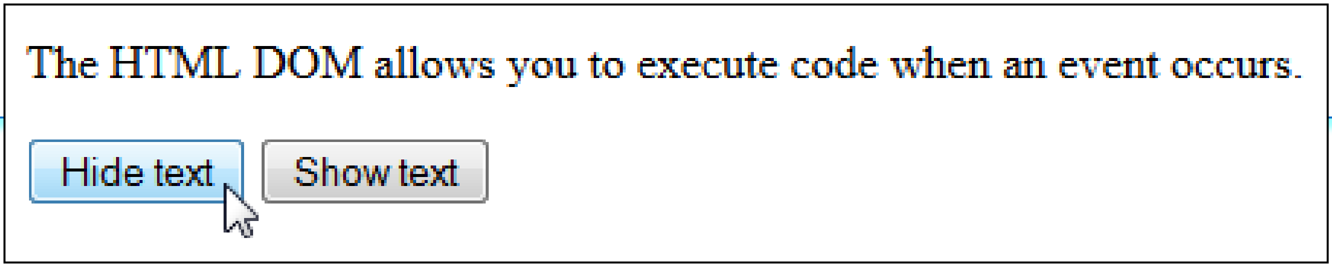
```
<!DOCTYPE html>
<html>
<body>

<h1 id="id1">My Heading 1</h1>

<button type="button"
onclick="document.getElementById('id1').style.color = 'red'">
Click Me!</button>

</body>
</html>
```


My Heading 1

Reacting to events

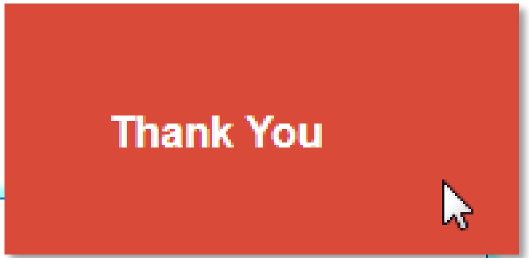
- Example with JavaScript expression

```
<!DOCTYPE html>
<html>
<body>

<p id="p1">
The HTML DOM allows you to execute code when an event occurs.
</p>

<input type="button" value="Hide text"
onclick="document.getElementById('p1').style.visibility='hidden'">
<input type="button" value="Show text"
onclick="document.getElementById('p1').style.visibility='visible'">

</body>
</html>
```


The HTML DOM allows you to execute code when an event occurs.

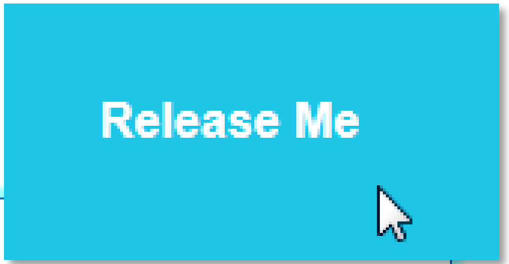
More examples …

- Example with function call



```
<!DOCTYPE html>
<html>
<body>
<div onmouseover="mOver(this)" onmouseout="mOut(this)"
style="background-color:#D94A38;width:120px;height:20px;
padding:40px;color:white;font-family:Arial;font-weight:bold;">
Mouse Over Me</div>
<script>
function mOver(obj) {
    obj.innerHTML = "Thank You"
}
function mOut(obj) {
    obj.innerHTML = "Mouse Over Me"
}
</script>
</body>
</html>
```

More examples …



```
<!DOCTYPE html>
<html>
<body>

<div onmousedown="mDown(this)" onmouseup="mUp(this)"
style="background-color:#D94A38;width:120px;height:20px;
padding:40px;color:white;font-family:Arial;font-weight:bold;">
Click Me</div>
<script>
function mDown(obj) {
    obj.style.backgroundColor = "#1ec5e5";
    obj.innerHTML = "Release Me";
}
function mUp(obj) {
    obj.style.backgroundColor="#D94A38";
    obj.innerHTML="Thank You";
}
</script>
</body>
</html>
```

Adding events handlers

- The addEventListener() method attaches an event handler to the specified element
  - element.addEventListener(event, function, useCapture)
  - The first parameter is the type of the event (like “click” or “mousedown”) – note: “click”, not “onclick”
  - The second parameter is the function to be called when the event occurs
  - The third parameter (optional) is a boolean value specifying whether to use event bubbling or event capturing
- You can add more than one event handler to the same element

Adding events handlers

- Advantages
  - The addEventListener() method attaches an event handler to an element without overwriting existing event handlers
  - You can add many event handlers of the same type to one element, i.e two “click” events
  - You can add event listeners to any DOM object not only HTML elements, i.e the window object
  - The addEventListener() method makes it easier to control how the event reacts to bubbling
  - When using the addEventListener() method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup
  - You can easily remove an event listener by using the removeEventListener() method

Event handler example

- Single event handler

```
<!DOCTYPE html>
<html>
<body>
<p>This example uses the addEventListener() method to attach a click
event to a button.</p>
<button id="myBtn">Try it</button>
<script>
document.getElementById("myBtn").addEventListener("click", function()
  {
    alert("Hello World!");
  });
</script>
</body>
</html>
```

Event handler example

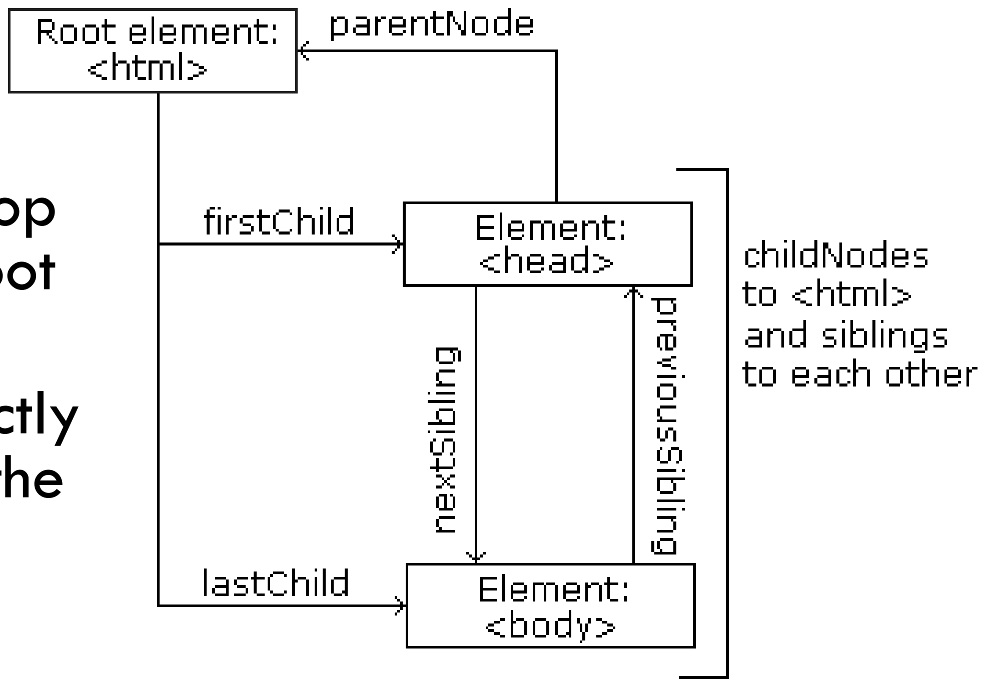
- Many event handlers

```
<p>This example uses the addEventListener() method to add many events
on the same button.</p>
<button id="myBtn">Try it</button>
<p id="demo"></p>

<script>
var x = document.getElementById("myBtn");
x.addEventListener("mouseover", myFunction);
x.addEventListener("click", mySecondFunction);
x.addEventListener("mouseout", myThirdFunction);

function myFunction() {
  document.getElementById("demo").innerHTML += "Moused over!<br>"; }
function mySecondFunction() {
  document.getElementById("demo").innerHTML += "Clicked!<br>"; }
function myThirdFunction() {
  document.getElementById("demo").innerHTML += "Moused out!<br>"; }
</script>
```

Event propagation

- Two ways of event propagation in the HTML DOM: bubbling and capturing
- If you have a <p> element inside a <div> element, and the user clicks on the <p> element, which element's "click" event should be handled first?
  - Bubbling (default, useCapture=false): the inner most element's event is handled first and then the outer one
  - Capturing (useCapture=true): the outer most element's event is handled first and then the inner one
- Example
  - http://www.w3schools.com/js/tryit.asp?filename=tryjs_addeventlistener_usecapture

DOM and node relationships



- In a node tree, the top node is called the root (or root node)
- Every node has exactly one parent, except the root (which has no parent)
- A node can have a number of children
- Siblings (brothers or sisters) are nodes with the same parent

Example

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

- From this HTML code:
  - <html> is the root node
  - <html> has no parents
  - <html> is the parent of <head> and <body>
  - <head> is the first child of <html>
  - <body> is the last child of <html>

  - <head> has one child: <title>
  - <title> has one child (a text node): “DOM Tutorial”
  - <body> has two children: <h1> and <p>
  - <h1> has one child: “DOM Lesson one”
  - <p> has one child: “Hello world!”
  - <h1> and <p> are siblings

Navigation among nodes

- Node properties useful to navigate between nodes
  - parentNode
  - childNodes[nodenumber]
  - firstChild
  - lastChild
  - nextSibling
  - previousSibling
- Other node properties
  - nodeName
  - nodeValue
  - nodeType

http://www.w3schools.com/js/js_htmldom_navigation.asp

Example

- Collects the node value of an <h1> element and copies it into a <p> element

```
<!DOCTYPE html>
<html>
<body>
<h1 id="intro">My First Page</h1>
<p id="demo">Hello World!</p>

<script>
var myText = document.getElementById("intro").childNodes[0].nodeValue;
document.getElementById("demo").innerHTML = myText;
</script>

</body>
</html>
```

```
<script>
myText = document.getElementById("intro").firstChild.nodeValue;
document.getElementById("demo").innerHTML = myText;
</script>
```

Create new elements

- To add a new element to the HTML DOM, you must create the element (element node) first, and then append it to an existing element
- In the following example, that creates a new paragraph in an existing <div>, the steps are:
  - create a new <p> element
  - create a text node
  - append the text node to the <p> element
  - append the new element to the <div> element

Create new elements

- Example

```
<!DOCTYPE html>
<html>
<body>

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);
var element = document.getElementById("div1");
element.appendChild(para);
</script>

</body>
</html>
```

Create new elements

- If you don't want to append the new element as the last child
  - insertBefore() method

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);

var element = document.getElementById("div1");
var child = document.getElementById("p1");
element.insertBefore(para,child);
</script>
```

Remove existing elements

- You must know the parent of the element
  - removeChild() method

```
<!DOCTYPE html>
<html>
<body>
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var parent = document.getElementById("div1");
var child = document.getElementById("p1");
parent.removeChild(child);
</script>

</body>
</html>
```

Replace existing elements

◦ replaceChild() method

```
<!DOCTYPE html>
<html>
<body>
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var parent = document.getElementById("div1");
var child = document.getElementById("p1");
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);
parent.replaceChild(para,child);
</script>
</body>
</html>
```

References

- JavaScript and HTML DOM Reference
  - http://www.w3schools.com/jsref/default.asp
- JavaScript tutorials
  - http://www.w3schools.com/js/
  - http://www.html.it/guide/guida-javascript-di-base/
  - http://www.codecademy.com/tracks/javascript
  - http://www.tutorialspoint.com/javascript/

Licenza d'uso

- Queste diapositive sono distribuite con licenza Creative Commons "Attribuzione - Non commerciale - Condividi allo stesso modo 2.5 Italia (CC BY-NC-SA 2.5)"
- Sei libero:
  - di riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare quest'opera
  - di modificare quest'opera
- Alle seguenti condizioni:
  - **Attribuzione** — Devi attribuire la paternità dell'opera agli autori originali e in modo tale da non suggerire che essi avallino te o il modo in cui tu usi l'opera.
  - **Non commerciale** — Non puoi usare quest'opera per fini commerciali.
  - **Condividi allo stesso modo** — Se alteri o trasformi quest'opera, o se la usi per crearne un'altra, puoi distribuire l'opera risultante solo con una licenza identica o equivalente a questa.
- http://creativecommons.org/licenses/by-nc-sa/2.5/it/