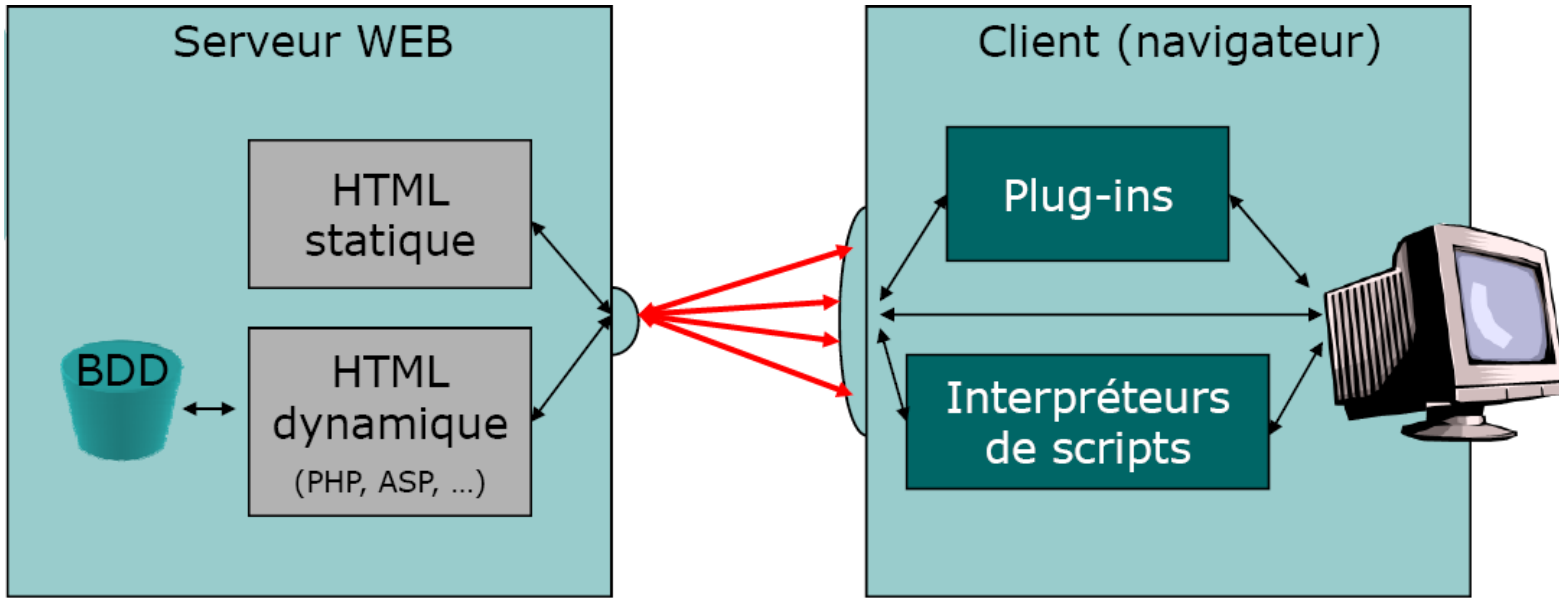
JAVASCRIPT

Dritan Nace

Concepts covered in this course

- JavaScript
  - General aspects
  - JavaScript programming basics
  - JavaScript for the web.
- DHTML
  - DOM versus JavaScript
- JavaScript : an event-oriented langage
  - Event manager,
- JavaScript and CSS
- AJAX

Context



- **JavaScript** is a script programming language mainly used in interactive web pages, client side.
- It is integrated into HTML pages; it is an interpreted language.

JavaScript

- **JavaScript** (object oriented script language) is a script programming language mainly used in interactive web pages.
- The language was created in 1995 by Brendan Eich (then called LiveScript) on behalf of Netscape.
  - Subsequently, the Netscape-Java pairing gave birth to Javascript.
- A very similar language, Jscript, was proposed by Microsoft.
- The basic syntax is intentionally similar to Java and C + + to reduce the amount of new concepts required to learn the language.
- ECMA proposed the "ECMAScript“ standard, the latest standard (ISO): ECMA 262 (5th edition - December 2009). JavaScript, currently at version 1.8.1, is an implementation of ECMA-262.

JavaScript

- A somewhat rudimentary object oriented language (known as a prototype)
- It is a relatively un-typed language:
  - The same variable can have several type successively.
- It offers basic types,
  - a syntax which is similar to the programming languages like C and Java,
- For experienced programmers, it also works as an object-oriented language and as a procedural language.
- The code can be read by anyone;
- The links are dynamic, therefore the reference of the objects is not verified during loading.

JavaScript : an object-oriented language

- An object is an encapsulation of variables called properties and functions called methods.
- The properties are boxes containing primitive values, methods and / or objects.
- The primitive values can be taken in the following primitive types: Undefined, Null, Boolean, Number or String.
- A method is simply a function associated to an object.
- JavaScript does not support the concept of usual class object-oriented languages .
  - The objects are not instances of classes, but are each equipped with constructors to generate their properties, including a prototype property that allows users to generate custom objects inheritors.

Primitive types

- JavaScript recognises the following primitive types:
  - numbers: 42 ou 3.1415
  - booleans: true or false
  - strings: "Bonjour !"
  - null
- There is no distinction between integers and real values.

- Real numbers use "." for the decimal point and *e* or E for the exponent.
- Strings
  - " or ’ can be used to demarcate a string.
  - Special characters can be used inside :
    - \ b for a backspace,
    - \ f for a page break,
    - \ n for a line break,
    - \ r for a line return,
    - \ t for a tabulation.

The primitive types

- JavaScript is weakly typed : the variables are not associated to a particular type.
  - Example :
    - var i = 4.25
    - i = "Hello"

  In an expression involving numbers and strings, numbers are converted to a string.
  - message = " There are" + 79 + "students in SR" + 03

Range of variables

- Range of variables
  - Global : the variable is visible from any point.
  - Local: the variable is visible only in the current function. To declare a variable which is local to a function the var keyword must be used.

```
<script language = "JavaScript">
Function test_visibilite() {
var test = 5;
}
test = 2;
test_visibilite();
alert (test);
</script >
```

Operators

- Assignment operators
  =, +=, -=, *=, etc.
- Arithmetic operators
  - +, -, /, *, %, --, ++
- Logical operators
  - &&, || and!
- Comparison operators
  - ==, != >, <,>=, <=,
- String operators
  - +, +=.

Special operators

- *new* creates a new object.
- *typeof* determines the type of an operand.
  - Syntax : typeof *expression* or typeof (*expression*).
- Void
- Conditional expressions « *? : »*
  - The form is condition? val1 : val2.
  - Example : m=(age>=18) ? "major" : "minor"

Language elements

- Comments : two possible forms:
  - // comment until end of line
  - /* comment until */
- Variable declaration
  - Syntax : var *name1 [= value1] [...*, *nameN [= valueN]]*.
    - examples : var i = 0, j = 2
- Function declaration
  - function *name*() { *instructions* }
  - function *name*(*param1*,...,*paramN*) { *instructions* }
- The return instruction (exit from a function)
  - return, return *expression*

Loops and branches

- The do ... while loop is :
  - do *instruction-or-block* while (*condition*).
- The while loop:
  - while (*condition*) *instruction-or-block*
- The for loop :
  - for (*initialisation* ;*condition* ;*incrementation*) *instruction-or-block*.
- The for ... in loop:
  - for (*variable* in *objet*) *instruction-or-block*
- The if ... Else test
- The multiple switch branching

```
switch(i) {
case 1:
// something
break
case 2:
// something else
break
default:
// yet again something else
break
}
```

Objects

- All objects have a set of properties
  - *<object>.<property>*
- To define a property :
  - Associate a value to it (student.name = " Lutton");
  - It can also be seen as an element of an array (student["name"]= " Lutton");
- A property can also be an object.

- A method is a function which is associated to an object
  - *<object>.<methode> = <function>*
  - *this* is used to reference the current object
  - *with* is used when we want to carry out several instructions with an object.

```
with(document)
        { open();
        write("The title of this page is " + title);
        close(); }
```

Object creation

- New objects can be created in JavaScript :
  - Define a type of object by creating a function
  - Create an instance using the ***new*** operator
    - An example :

```
function student(surname, firstname) {
this.surname = surname;
this.firstname = firstname;
}
P = new student ("Goldman", "Jean-Jacques");
```

It is possible to add properties dynamically.

JavaScript : procedural and object oriented

- Let r be an object defined with two properties : *r.width* and *r.height*.
  - To determine the surface of the rectangle :

    *function calculateAreaOfRectangle (r) {return r.width*r.length;}*

  Or otherwise:

  function Rectangle(w,h) {

  this.width= w;

  this.height = h;

  *this.area= function () {return r.width*r.height;}*

  }

  None of these solutions is optimal!!

Prototype and inheritancee

A special property *prototype*

```
function Rectangle(w,h) {
    this.width = w;
    this.height= h;
    }
```

The object prototype is designed to contained the methods and properties which can be shared by all the instances.

*Rectangle.prototype.area= function () {return r.width*r.height;}*

The prototype object is associated with the constructor and any object that it initialises inherits the same set of properties of the prototype.

Objects and predefined functions

- Some predefined objects : Array, Boolean, Date, Math, Number, Object, String…
  - For each object there are constructors, predefined properties and methods, eg. for *Array* :
    - ConstructorArray(n), Array(el_1,el_2,..,el_{n-1}),
    - Properties: index, input, length..
    - Methods : concat, join, pop, push, sort, toString…
  - The Math object is a non-instantiable object, which has properties and methods
  - …
  - DOM objects…
- Predefined functions
  - eval(), escape()/unescape(), etc.

JavaScript for the web

- At first, JavaScript was used for animation, graphic effects …,
- Since JavaScript is mainly used in interactive web pages, making life easier for the user but also providing added value to web pages providers.
- Different uses of JavaScript :
  - Html form control;
  - Audience measurement tools;
  - Advert managing;
  - Ajax: help with automatic form completion…

JavaScript for the web

- JavaScript and security
  - JavaScript cannot access files stored on the client site;
  - JavaScript cannot access html pages from other sites.
  - JavaScript does not allow installation of executables on the client site;

- Limits of JavaScript
  - Navigator and platform compatibility;
  - JavaScript is not visible with referencing motors;

Integration of a script in an HTML page

- Use of the <script> tag;

```
<script language="JavaScript">
alert("my first JavaScript text !!!!")
</script>
```

- Including an external file;

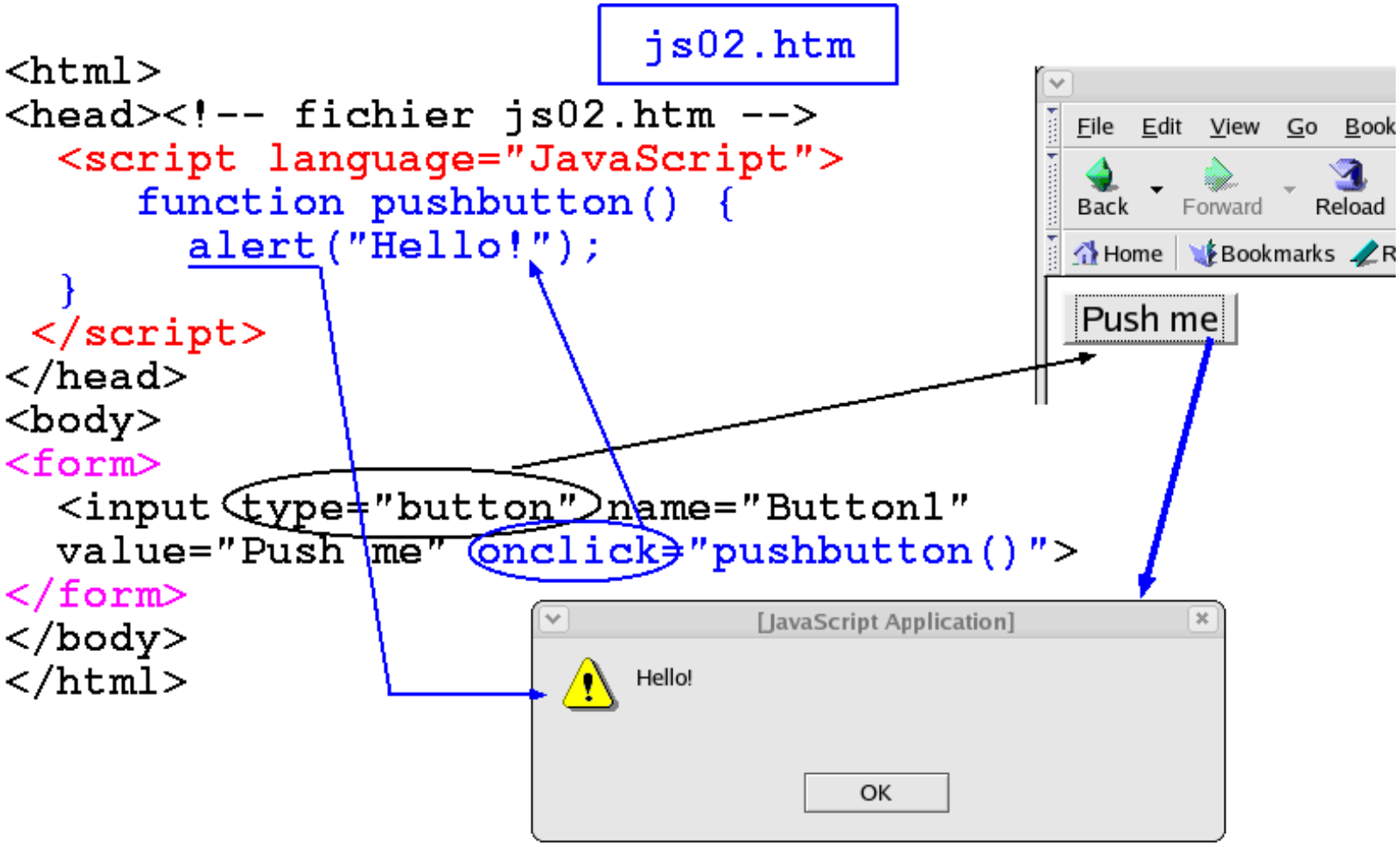
```
<script language="JavaScript«  src=« myfile.js">
```

- Directly between the tags by associating JavaScript to the events;

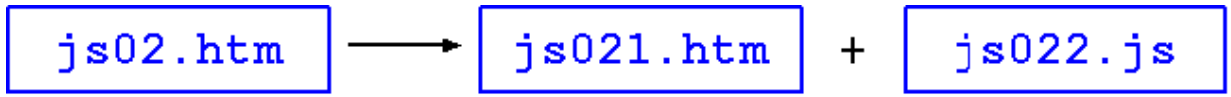
```
<INPUT TYPE="button" VALUE="this paragraph contains an important sentence"
 onClick='alert("We are all equal")' >
```

Avoid the latter!!

Use of the <script> tag



Including an external file;



Navigateurs modernes: possible stocker code javascript
(>=V4) dans des fichiers séparés

```
<html>
<head>
  <script language="JavaScript" src="./js022.js">
  </script>
</head>
<body>
<form>
  <input type="button" name="Button1"
  value="Push me" onclick="pushbutton()">
 </form>
</body>
</html>
```

```
function pushbutton() {
        alert("Hello!");
}
```

JavaScript et les cookies

- No methods which are native to JavaScript to manage cookies.
- The *cookie* property of *document* are used.
- SetCookie() and getCookie() functions are written.

```
function setCookie(name, value, expires, path, domain, secure) {
        document.cookie=name+"="+escape(value)+
                ((expires==undefined) ? "" : ("; expires="+expires.toGMTString()))+
                ((path==undefined) ? "" : ("; path="+path))+
                ((domain==undefined) ? "" : ("; domain="+domain))+
                ((secure==true) ? "; secure" : "");
}
function getCookie(name) {
        if (document.cookie.length==0) { return null; }
        var regCookies=new RegExp("(; )","g");
        var cookies=document.cookie.split(regCookies);
        for (var i=0; i<cookies.length ; i++) {
                var regInfo=new RegExp("=","g");
                var infos=cookies[i].split(regInfo);
                if (infos[0]==name) {
                        return unescape(infos[1]);
                }
        }
        return null;
}
```

Cookies managed by JavaScript

```
<script type="text/javascript" src="cookie.js"></script>
<script type="text/javascript">
function setPrefLangue(pref) {
	var dtExpiration1An=new Date();
	dtExpiration1An.setTime(dtExpiration1An.getTime()+365*24
*3600*1000);
	setCookie("langue", pref, dtExpiration1An);
	window.location.href=document.location;
}
function getPrefLangue() {
	var txtIntro="Choisissez votre langue :";
	var laLangue=getCookie("langue");
	if (laLangue=="fr") {
		txtIntro="Choisissez votre langue :";
	}
	if (laLangue=="us") {
		txtIntro="Choose your language :";
	}
	if (laLangue=="es") {
		txtIntro="Cual es su idioma :";
	}
	document.write(txtIntro+" <a
href=\"javascript:setPrefLangue('fr')\">Français</a> - <a
href=\"javascript:setPrefLangue('us')\">Anglais</a> -  <a
href=\"javascript:setPrefLangue('es')\">Espagnol</a>")
}
</script>
<script type="text/javascript">
	getPrefLangue();
</script>
```

The « cookie.js » file contains the two *setCokkie* et *getCookie* functions, talked about previously.

Example taken from « JavaScript le guide complet », by O. Hondermarck, MicroApplication 2009 editions.

DHTML

- **JavaScript is designed for making dynamic HTML**, (***Dynamic HTML***, or **DHTML) !**

- DHTML is a generic name given to all techniques used by the author of a web page so that it is able to modify itself while being viewed in the web browser :

DHTML presentation

- The technologies which que DHTML implements are :
  - HTML, necessary in order to present the document;
  - Style Sheets (CSS) to define a style for multiple objects and positioning them on the page ;
  - The Document Object Model (DOM), providing a hierarchy of objects, to facilitate their handling ;
  - The JavaScript scripting language, which is essential to define user events ;

DHTML – how it works

- Animation of elements:
  - Animation = property modification (position, height, width, visibility, color ...) or using their methods (functions associated with an element).
  - This can only be achieved with the help of :
    - a JavaScript code to modify the properties of elements in response to user events (mouse click, mouse movement, ...),
    - + structuring of elements in the page defined by the DOM (Document Object Model).

Document Object Model

- The Document Object Model (or DOM) corresponds to an interface independent of any programming language and any platform, allowing computer programs and scripts to access or update the content, structure or style of Documents.

- DOM is often identified by a tree structure of a document and its elements.
  - eg. each element generated from the tags as in the case of HTML, a paragraph, a title or a form button, forms a node.

- DOM is used to easily alter or access content from web pages.

- From a given DOM tree, it is also possible to generate documents in the desired tag language;

Dom versus JavaScript

- While JavaScript is the programming language that can operate and manipulate the DOM objects, the DOM provides methods and properties required to read, modify, update and remove items from the examined document.

- The Document interface is the first to be defined in the DOM Level 1 Core specification, and document is a host object implementing this interface. This document object contains everything that appears in a web page.

- An HTMLDocument interface is derived from the main interface, document. The HTMLDocument interface defines the operations and queries that may occur on an HTML document ;

HTML et DOM objects

- *Window* is an object which corresponds to the window in which a web page is displayed.

- Location is a subobject of Windows. It is a client-side object (ie created by the browser). Location is currently an interface which is integrated to DOM and also an attribute of HTMLDocument.

- The Document object defined in the DOM specification provides access to the content of an HTML page. The most widely used methods of the object are getElementByTagName and getElementById, open, write ...

- History is a subobject of Windows that provides an interface with the browser history stored by the browser.

- The Node interface is defined in the DOM 2 specification. It provides access to the structure of an HTML document viewed as a tree item, and allows modification of this structure..

- The NodeList interface is a DOM 2 object , which provides access to elements of a Web page or an XML file. It is a dynamic element, all structural changes to the page content modify the content of NodeList.

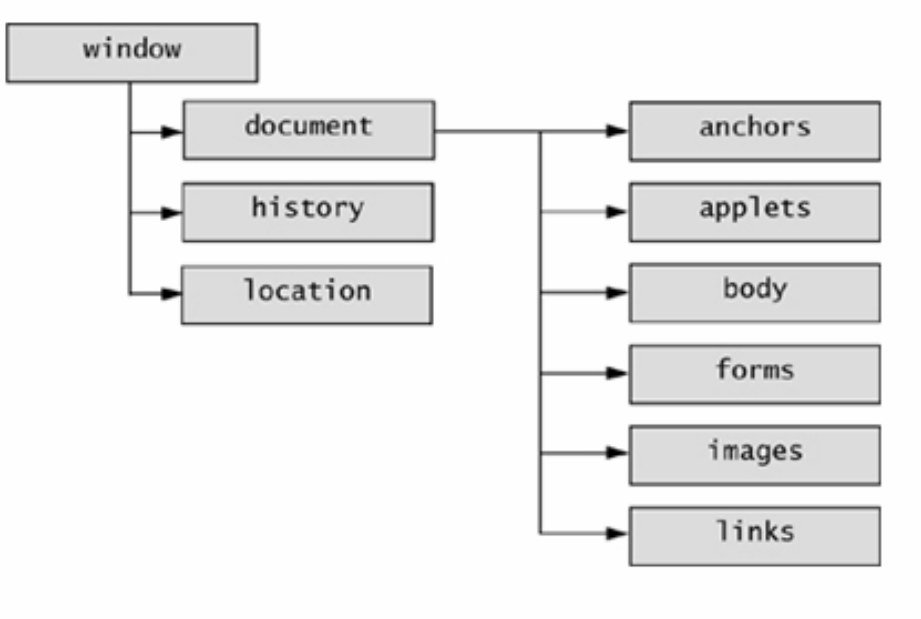
Window (I)

- *Window* est un objet qui correspond à la fenêtre dans laquelle s'affiche une page Web. Une telle fenêtre peut être créée dynamiquement.

- **Attributs de window**
  - **frames[]** The frames in a window. Read only.
  - **length** The number of frames. Read only.
  - **name** Name of the window.
  - **status** State bar text.
  - **parent** The parent window of a given window.
  - …

- Window creation properties
  - scrollbars, statusbar, toolbar, menubar, resizable, directories.

Window (II)



The window object and some of its components

Window (III)

- **Objects contained in window**
  - **Document :** Corresponds to a page, which contains a window.
  - **History :** List of the pages which were previously viewed in the same window.
  - **Location :** Corresponds to the URL of a page, which contains the window.
  - **Screen :** Corresponds to the screen and contains the following properties: width, height, availWidth, availHeight, colorDepth.
  - **Navigator**…
- **Window methods**
  - **open(url, nom [, property list])** Opens a new window. The URL and name are given as parameters as well as the list of options.
  - **close()** Closes the window. Example: x.close(); window.close();
  - **alert(message)** Displays a message.
  - …

Document

- **Attributes of *Document*** (these are read-only attributes, defined during the page creation)
  - *DocumentType* **doctype** (The doctype defined in the page header.)
  - *DOMImplementation* **implementation**
  - *Element* **documentElement** An *Element* is an object which represents a tag and possesses methods.
  - …

- **Attributs de HTMLDocument**
  - *DOMString* **title,** *DOMString* **URL,** *HTMLElement* **body,** *HTMLCollection* **anchors,** *HTMLCollection* **links,** *HTMLCollection* **forms,** *DOMString* **cookies** …

Document

- **Methods of *Document.*** DOM methods can take a *DOMString* paramater, which is usually a simple string in JavaScript.
  - *Element* **createElement(***DOMString***)**
    Creates an element, a tag, whose name is given, and returns an *Element* object*t*.
  - *Element* **getElementById(***DOMString***)**
    Returns an element of a given.…
  - ..

- **Methods of HTMLDocument**
  - void **open**() Creates a new document.
  - void **close**() Closes the current document.
  - void **write**(*DOMString*) Writes to the document.
  - void **writeln**(*DOMString*) Writes the parameter string and adds a line break.

Location

Take the following like:

http://www.xul.fr:80/ecmascript/tutoriel/window-location.php#content?name=value

[http:]//[www.xul.fr]:[80][/ecmacript/tutoriel/windows-location.php]#[content]?[name=value]

[protocol][host][port][pathname][hash][search]

protocol http:
host www.xul.fr:80
hostname www.xul.fr
port 80
pathname ecmascript/tutoriel/window-location.php
hash content
search name=value

href corresonds to the complete URL.

Use of location

The properties of *location* can be read or modified.

For example, display the path of the page as follows:

```
document.write(location.pathname)
```

Change the page as follows:

```
location.href = "url"
```

and other possibilities with replace(), reload()…

History

- **History Methods:** They allow movement from a page to the preceding or following one, or to a given page.

  - **back**() Load the preceding page.
  - **forward**() Load the next page, after going back once.
  - **go**(x) Load a page whose number in the list or URL is given. Example: history.go(-2) to reverse the last tpour annuler les deux derniers clics.

Node

- The Node interface is defined in the DOM 2 specification. It provides access to the structure of an HTML document, viewed as a tree of elements, and modify this structure.
- Properties
  - *DOMString* **nodeName** The name of the tag, like <table>, <pre> etc.
  - *DOMString* **nodeType** Node category: element, comments, text, etc.
  - DOMString **nodeValue** The value of node, when this is meaningful. All nodes which do not have values, are equal to *null*.
  - *NamedNodeMap* **attributes** List of the node attributes. Read only.
- Méthodes
  - *Node* **appendChild**(*Node* added) Adds a node to the list of child nodes.
  - *Node* **cloneNode**(*Boolean*)
  - Boolean **hasAttributes**() Returns true or false, depending on whether it has any attributes or not.
  - *Boolean* **hasChildNodes**() Returns true or false depending on whether it has any other nodes or not.
  - *Node* **removeChild**(*Node* deleted) Deletes a node or a branch. Returns the deleted element.
  - *Node* **replaceChild**(*Node* new, *Node* old) Replaces the second argument with the first.

NodeList

- The NodeList interface is a DOM 2 object, which provides access to elements of a Web page or an XML file.
  - It is a dynamic element, all structural changes to the page content change the contentNodeList.
  - NodeList has a unique attribute (*int* **length** Number of *Nodes* in the list), and one **Méthode** Node **item(int)** which returns the *Node* whose index is given as an argument.

Example:

```
var dnl = document.getElementsByTagName("a");
for(i = 0; i < dnl.length; i++)
    var lien = dnl.item(i);
```

The code above provides a list of links on a page, since they are introduced by the <a>. Each link is then obtained using the item() method of Nodelist

JavaScript : an event oriented language

- An event is an action generated by the user.
- Each tag can handle an event : on the occurrence of an event JavaScript code can be executed.
  - Syntax <balise EventHandler= "code JavaScript")

```
<input type= "button" value "Click here"
onclick = "alert (‘Thanks !’);">

<form name  "f1" >
<input name= "b1" type= "button" value "Click here" >
</form>

Document.f1.b1.onclick = function(){alert (‘Thanks !’  ); };
```

Events

- Page and window events
  - *onabort* – if there is an interruption while loading
  - *onerror* – if there is an error while loading a page
  - *onload* – after the end of page loading
  - *onbeforeunload* ; *onunload* ; *onresize* ;

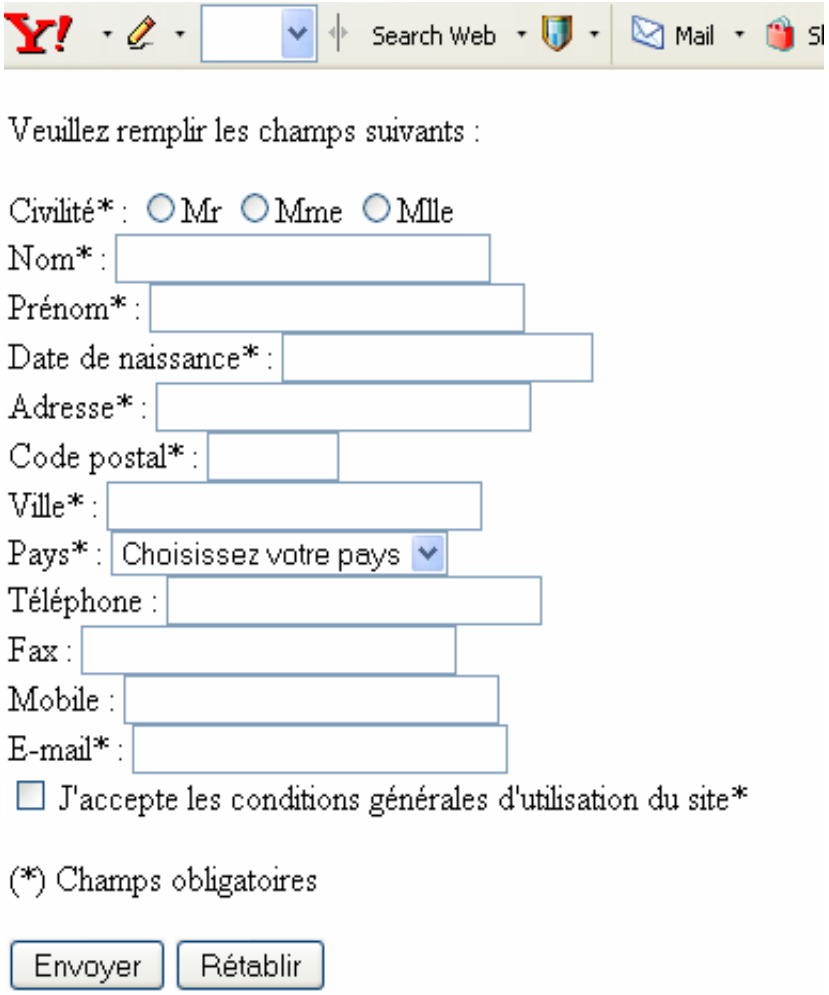
- Mouse event
  - *onclick* – a single click
  - *ondblclick* – a double click
  - *onmousedown* – when the mouse button is pressed, without necessarily being released
  - *onmousemove* - *onmouseout* - *onmouseover* – *onmouseup;*

- Keyboard events
  - *onkeydown* – when a key is pressed
  - *onkeypress* - when a key is pressed and released
  - *onkeyup* – when a pressed key is released
- Form events
  - *onsubmit* – when the form is validated (*via* a button of the "submit" type or a submit() function)
  - *onreset* – during the resetting of the form (*via* a "reset" button or a reset() function)
  - *Onselect; onblur; onchange; onfocus;*

Associating Javascript with events

```
<html>
<head>
<title>Exemple JavaScript</title>
</head>
<body>
<h1>Exemple JavaScript</h1>
<INPUT TYPE="button" VALUE="this paragraph contains an important
    sentence"
 onClick='alert("We are all equal")' >
<BR>
  </BODY>
</HTML>
```

Associating Javascript with events : form control(I)



- Associating javascript functions to control how the fields are filled :
  - Test if the Title is entered,
  - Test the E-mail field,
  - Test if the (code postal, téléphone, fax) fields have integer values,
  - Test the date…
  - Test the agreement to the general conditions

  - Create a testSaisie() function to associate to onSubmit().

Associating Javascript with events : form control(II)

```
<BODY>
<FORM name="form1" onSubmit="return testSaisie()">
  <P>Veuillez remplir les champs suivants :</P>
  <P>Civilité* :
    <INPUT type="radio" name="civ" value="Mr">Mr
    <INPUT type="radio" name="civ" value="Mme">Mme
    <INPUT type="radio" name="civ" value="Mlle">Mlle<BR>
…
<P>(*) Champs obligatoires</P>
  <P>
    <INPUT type="submit" name="Submit" value="Envoyer">
    <INPUT type="reset" name="Submit2" value="Rétablir">
    <BR>
  </P>
</FORM>
```

```
function testNumerique(valeur)
{
  if (valeur == parseFloat(valeur)) return true;
  else return false;
}
```

```
function testMail(email)
{
  var posArobase;
  posArobase = email.indexOf("@");
  if (posArobase == -1) return false;
  var posPoint;
  posPoint = email.lastIndexOf(".");
  if ((posPoint == -1) || (posPoint < posArobase)) return false;
  return true;
}
```

```
function testRadio(nomForm,nomGroupe)
{
  var compteur;
  compteur = 0;
  while (compteur < nomForm.elements[nomGroupe].length)
  {
    if (nomForm.elements[nomGroupe][compteur].checked)
    return true;
    compteur++;
  }
  return false;
}
```

JavaScript and CSS

- JavaScript allows modification of CSS styles:
  - on-line modifying of element styles :
    - ex. changing the colour of an element…
      - An item can be accessed via document.getElementById …
  - Modifying the class of an element :
    - The class of an element can be accessed via the className attribute of any html element…
    - Any CSS element is accessible via JavaScript and DOM depending on the rule:
      - The name does not change if there is no indent. Otherwise delete the hyphen and capitalize the first letter that follows.
  - Manipulating the page styles themselves:
    - This corresponds to activating and deactivating the style pages
      - Use the *disabled* property for the <link> and <style> elements

Review

- HTML allows text encapsulation between the tags, in order to continue to formating;
- CSS can define the format of the tags ... they can be conditional (eg depending on the media);
- JavaScript is a language which is integrated into HTML.
- DHTML is the union of HTML, CSS and Javascript. Using the DOM, it allows rewriting of the HTML page without reloading;

AJAX

Ajax: motivation and operation principle

- Ajax (Asynchronous JavaScript + XML) is the joint use of several technologies like XHTML, CSS, DOM, XML and / or JSON, all linked by JavaScript that uses the XMLHttpRequest mechanism.

- Ajax permet l'échange d'informations entre le navigateur et le serveur web sans recharger la totalité de la page.
  - Il permet de garder des pages web affichées et avoir des performances accrues;
  - analyser et travailler avec des documents XML.

- Principe: *l'exécution du code JavaScript continue en parallèle du traitement de la requête Ajax (asynchrone).*
  - Une requête est envoyée au serveur (grâce à XMLHttpRequest),
  - Le serveur envoie la réponse;
  - le JavaScript (et non le navigateur) réceptionnera et traitera la réponse.

Synchrone versus asynchrone

Take the example of the management of a shopping cart on e-commerce site.

In synchronous mode, the user adds an article to the basket.

- A request is sent to the 4D server
- The navigator is **waiting** a response from the server
- The server validates the addition
- The server sends the content of the new basket back
- The navigator refreshes the display zone of the basket
- The user **continues** the purchase

In asynchronous mode the user adds an article to the basket.

- A request is sent to the 4D server
- The navigator continues its execution
- The user **continues** his purchases
- The server validates the addition
- The server calls the navigator again and sends the content of the new basket
- The navigator refreshes the display zone of the basket

In this case, it is preferable to wait for the validation of the addition of the article (and therefore the synchronous mode).

Propose applications which use the asynchronous mode.

Ajax : how it works(I)

The object *XMLHttpRequest* is the cornerstone of Ajax.

Properties

- Onreadystatechange (a function is assigned to it with the response processing instructions);
- readyState: state of completion of the request from 0 (uninitialised) to 4 (complete).
- responseText / responseXML: the server response to the request in text (ie objects recognised by JavaScript) or formatted as an XML object.
- Status: status of the HTTP server response. (200 OK)

Methods

- open(*method*, *url*, *bool*) : (if asynchronous mode, bool is true, otherwise false)
- send(*data*) : Sends the HTTP request (when there is no more data, the value is null)
- setTimeouts(*timeout*)
- …

```
if (window.XMLHttpRequest) {
xhr_obj = new XMLHttpRequest(); }
else if (window.ActiveXObject) {
xhr_obj = new ActiveXObject("Microsoft.XMLHTTP");
xhr_obj.open("GET", location.href, false);
xhr_obj.send(null);
if(xhr_obj.readyState == 4)
alert("Request finished!");
```

Ajax : how it works(II)

Launching of an (asynchronous) HTTP

```
if (window.XMLHttpRequest) {
xhr_obj = new XMLHttpRequest(); }
else if (window.ActiveXObject) {
xhr_obj = new ActiveXObject("Microsoft.XMLHTTP");}
…
 xhr_obj.onreadystatechange = function() {
// response processing instructions};
…
xhr_obj.open('GET', 'http://url….', true);
xhr_obj.send(null);
```

Managing the server's response

```
if (xhr_obj.readyState == 4) { // answer received
}
else { // n
}
…

if (xhr_obj.status == 200) { // OK !
}
else { // there was a problem with the HTTP request
}
```

Data formats
XML versus JSON

- JSON (JavaScript Object Notation) is a recursive format compatible with JavaScript, which is structured as a JavaScript object saved in a JSON *(JavaScript Object Notation)* file.

```
<?xml version="1.0" ?>
<root>
<menu>File</menu>
<commands>
<item>
     <title>Nouveau</value>
     <action>CreateDoc</action></item>
<item>
    <title>Ouvrir</value>
    <action>OpenDoc</action> </item>
<item>
   <title>Close</value>
   <action>CloseDoc</action> </item>
</commands>
</root>
```

```
{
  "menu": "File",
  "is": [
     {
      "title": "New",
      "action":"CreateDoc"
      },
      { "title": "Open",
        "action": "OpenDoc"
       },
       {
        "title": "Close",
        "action": "CloseDoc"
       }
    ]
}
```

To use a JSON file:

Load the file as text (method XMLHttpRequest.responseText). use the JavaScript eval ():

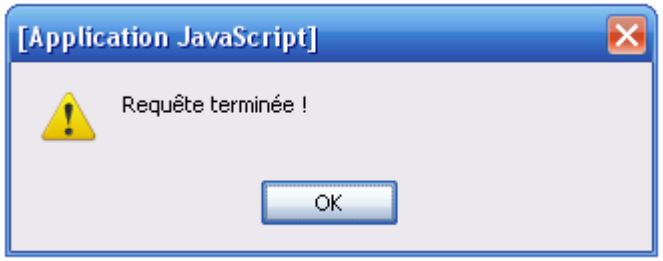
var jtxt = eval('(' + xhr_obj.responseText + ')');

Example of a simple AJAX request

```
<HTML>
  <BODY>
    <SCRIPT TYPE="text/javascript">
    </SCRIPT>
    <INPUT TYPE="button"
      VALUE="Lancer une requête asynchrone"
      ONCLICK="asynchrone()">
    <SCRIPT TYPE="text/javascript">
      function asynchrone() {
      if (window.XMLHttpRequest) {
        requete = new XMLHttpRequest(); }
        else if (window.ActiveXObject) {
        requete = new ActiveXObject("Microsoft.XMLHTTP");}
        //On ouvre une requêtre asynchrone : 'true'
        requete.open('GET', 'data.txt', true);
        //On indique la fonction de retour
        requete.onreadystatechange = exploiteRequete;
        //On envoie la requête
        requete.send(null);
      }
      //Fonction appelée quand la requête change d'état
      function exploiteRequete() {
        //Si la requête est terminée
        if ( requete.readyState == 4 ) {
          //Si la réponse ne contient pas d'erreur
          if ( requete.status!=500 && requete.status!=404 )
            eval(requete.responseText);
              }
        }
    </SCRIPT>
  </BODY>
</HTML>
```

The data.txt file contains :
alert('Request finished!');

Result of the request :



- Bibliography :

  - Michel Vayssade « poly SR03 »;
  - « HTML et JavaScript », S. Maccari et S. Martin, ed. MicroApplication, 2004.
  - « JavaScript: La référence » D. Flanagan, ED. O'Reilly, 5eme édition, 2007.
  - « AJAX : le guide complet »B. Catteau N. Faugout, édition Micro Applications, 2009.
  - « JAVASCRIPT : le guide complet », O. Hondermarck, édition Micro Applications, 2009.

Additionally, many useful websites:

- http://www.w3.org/
- *http://developer.mozilla.org/en/docs/Gecko_DOM_Reference* *Traduction française: Référence du DOM Gecko*
- http://www.commentcamarche.net/
- http://www.developpez.com/
- http://pagesperso-orange.fr/arsene.perez-mas/javascript/cours/document.htm