Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

5) Javascript

Emmanuel Benoist

Spring Term 2017

Computer Science Division

Javascript I

- Introduction
  - Include Javascript in an HTML Page
  - A First Example
- Basics
  - Event Handling
- Document Object Model - Manipulations
  - HTML Modifications
  - AJAX Principles
- Javascript Object Notation
  - Arrays
  - Objects
  - JavaScript Object Notation (JSON)
  - Using JSON instead of AJAX
  - Constructors
  - Prototypes

Javascript II

Reflexion

- Methods and Functions
  - Function as First Class Objects
  - Events Handling and Function Context in AJAX

Introduction

Include Javascript in an HTML

Presentation

- **Javascript $\neq$ Java**
  - Javascript is not a Java Dialect
  - Is a functional language,
  - Far away from OO design
  - Values have types (not variables like in Java)
- **A language for scripting**
  - Validate field value localy (format validation)
  - Change the content of a page
  - React to event handling
- **Is now much more**
  - Is used for communication between client and server (asynchrone)
  - Serves to modify the content of a page.

How to insert Javascript inside an HTML page

- **In the file itself**

```
<head><title>Test</title>
<script type="text/javascript">
<!--
 alert("Hello World!");
//-->
</script>
</head>
```

- **In an external File**

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

A First Example

Our First Script

```
<script type="text/javascript">
function square() {
    var result = document.form.number.value *
                    document.form.number.value;
    alert("The Square of " + document.form.number.value +
            " = " + result);
}
</script>
...
<form name="form" action="">
<input type="text" name="number" size="3">
<input type="button" value="Compute the Square"
        onClick="square()">
```

Basics

Event Handling

Event Handling

- **Javascript Reacts to Events generated by Html**
- **Modification of the Document**
  - `onError`, `onLoad`, `onAbord` (if you interrupt the load of an image), `onUnload` (when we quit the document)
- **Mouse Events**
  - `onClick`, `onDblClick`, `onMousedown`, `onMousemove`, `onMouseout`, `onMouseover`,`onMouseup`
- **Events generated by elements**
  - `onBlur` (when you leave an element), `onChange` (when a modification succeeded), `onFocus` (when the user enters an element), `onKeydown`, `onKeypress` (as long as it remains pressed, `onKeyup` (when is is up), `onReset` (at the reinitialisation of a form), `onSelec` (when you select a text), `onSubmit` (when the form is submited)

Document Event Handling

- **Suppose we have this HTML file**
  - We react to the events: `keydown`, `keyup`, `mouseover`, `mouseout`, and `click`

```
<div id="res"></div>
<form>
<input type="text" id="text1" onkeydown="keyDown1();">
<div id="copyText1"></div>
<input type="text" id="text2" onkeyup = "keyUp2(this.value);">
<div id="copyText2"></div>
<h2 id="text3" onmouseover="mouseover3();" onmouseout="mouseout3()↘
→;">On mouse over</h2>
<div id="copyText3"></div>
<div id="copyText4" ondblclick="test();">Double click on this text</div>
    <input type="button" value="send" id="btnTest" onclick="btnClick(↘
    →this);">
</form>
```

Document Event Handling (Cont.)

script3.js

```
function keyDown1(){
  var text1 = document.getElementById("text1").value;
  document.getElementById("copyText1").innerHTML = text1;
}
function keyUp2(text2){
  document.getElementById("copyText2").innerHTML = text2;
  }
function mouseover3(){
  document.getElementById("text3").innerHTML = "MOUSE IS OVER";
}
function mouseout3(){
  document.getElementById("text3").innerHTML = "On mouse over";
}
function test(){ alert("Double click"); }
function btnClick(item){ alert("ID of the item is "+item.id); }
```

Document Object Model - Manipulations

HTML Modifications

Document Object Model

- **Javascript DOM is a Tree**

```
<html><head><title>Test</title>
</head><body>
<p id="thetext">Text with <b>some bold content</b> and
<u>some underlined</u></p>
<script type="text/javascript">
<!--
 var number = document.getElementById("thetext").childNodes.↘
 →length;
 var first = document.getElementById("thetext").childNodes[0].↘
 →nodeValue;
 document.write("Nb␣of␣children␣nodes:␣<b>" + number +"<\/b↘
 →><br>");
 document.write("Value␣of␣the␣first␣node:␣<b>" + first +"<\/b><↘
 →br>");
//-->
</script>
</body></html>
```

ChildNodes array

```
<ul id="first_list"><li>first item</li><li>second item</li></ul>
<script type="text/javascript">
// 2 Children
 var number =
        document.getElementById("first_list").childNodes.length;
 document.write("First list: number of children nodes: <b>" +
                    number +"<\/b><br>");
</script>
<ul id="second_list">
<li>first item</li> <li>second item</li> <li>third item</li>
</ul>
<script type="text/javascript">
// 7 Children (spaces count as text)
 var number = document.getElementById("second_list").childNodes.length;
 document.write("second list: number of children: <b>" +
                    number +"<\/b><br>");

</script>
```

Insert a new Child to a node

```
<ol id="Liste">
<li>Element</li>
</ol>

for(var i = 0; i < 10; i++) {
   var nouveauLI = document.createElement("li");
   var numeroli = i + 1;
   var nouveautexteli = document.createTextNode(
            "It is the item number" + numeroli);
   document.getElementById("Liste").appendChild(nouveauLI);
   document.getElementsByTagName("li")[numeroli].
                        appendChild(nouveautexteli);
}
```

AJAX Principles

Ajax Principles

AJAX application life cycle.

- Use Javascript for collecting information
- Create a HTTP Request (containing a random number in order to avoid caching)
- Send this request and organize a handler for being executed after the reception of the response.
- Display the results inside the DOM.
- ...

AJAX Example

- **We have a Form containing a selection box**
- **On Change of the selection, the function `showCustomer()` is executed**
- **The function creates an Object (XMLHttpRequest or its MS-cousins)**
- **A request is sent to a PHP file,**
- **The PHP program generates a Table**
- **The table is included in the html DOM.**

The form containing a selection

```
<form>
Select a Customer:
<select name="customer" onchange="showCustomer(this.value↘
→)">
<option value="ALFKI">Alfreds Futterkiste
<option value="NORTS">North/South
<option value="WOLZA">Wolski Zajazd
</select>
</form>
<p>
<div id="txtHint"><b>Customer info will be listed here.</b↘
→></div>
</p>
```

Show Customer

We create a Request and send it using the XML/HTTP object

```
function showCustomer(str) {
  xmlHttp=GetXmlHttpObject();
  if (xmlHttp==null) {
    alert ("Your browser does not support AJAX!");
    return;
  }
  var url="getcustomer.php";
  url=url+"?q="+str;
  url=url+"&sid="+Math.random();
  xmlHttp.onreadystatechange=stateChanged;
  xmlHttp.open("GET",url,true);
  xmlHttp.send(null);
}
```

Function for creating the XML/HTTP object

```
function GetXmlHttpObject()
{
 var xmlHttp=null;
 try { // Firefox, Opera 8.0+, Safari
   xmlHttp=new XMLHttpRequest();
 }
 catch (e) { // Internet Explorer
     try {
         xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
     }
     catch (e) {
         xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
     }
 }
 return xmlHttp;
}
```

Handler for the Response

- **Each Time the state change, the handler function is executed:**
- It should only react for state 4 since it means: response received.

```
function stateChanged() {
  if (xmlHttp.readyState==4){
    document.getElementById("txtHint").innerHTML=
              xmlHttp.responseText;
  }
}
```

The PHP Program used to generate the response

- **The program reads the informtion and generate a response:**

```
$q=$_GET["q"];
$db_table = array(...);
echo "<table border='1'>
<tr> <th>Firstname</th> <th>Lastname</th></tr>";
$row = $db_table[$q];
 echo "<tr> echo " <td>" .. $row['FirstName'] .. "</td>";
 echo "<td>" .. $row['LastName'] .. "</td></tr>";
 echo "</table>";
```

Javascript Object Notation

Javascript Object Oriented

Javascript Object Notation JSON

Arrays

Arrays in Javascript

- **Arrays in JavaScript have only numerical indexes**
  - Similar to Java Vector
  - Elements can (should) be contiguous (like a normal index)
  - Elements can be initialized at the creation of the array
  - Elements can be inserted later.

```
var arr1=['one','two','three'];
var arr2 = [];
arr2[2]='Hello';
arr2[1]='World';
arr2[0]='Test';
for(var i=0; i< arr1.length;i++){
   document.write(arr1[i]);
   document.write(" ");
  }
for(var i=0; i< arr2.length;i++){
   document.write(arr2[i]);
   document.write(" ");
  }
```

Objects

Object in Javascript

- **An Object in javascript is more like a "numerical array" in PHP**
  - It contains fields that can be added dynamically and initialized or removed programatically.
- **Creation of an Object:** Creates an empty container

```
var myObject = new Object();
```

- **Can contain fields:**

```
myObject.shoeSize="42";
myObject['shoeSize']="39";
```

Add a function to the Object (not a class)

- **We can define a new function**

```
myObject.speakYourshoesSize=function(){
  alert("shoe size : "+this.shoeSize);
}
```

- **or use a predifined one**

```
function sayHello(){
  alert ('hello, my shoeSize is '+this.shoeSize);
}
...
myObject.sayHello=sayHello; // WITHOUT ↘
→PARENTHESIS !!!!
```

Complex Objects

- **We can attach objects inside other objects**

```
var myLibrary=new Object();
myLibrary.books=new Array();
myLibrary.books[0]=new Object();
myLibrary.books[0].title="Ajax in Action";
myLibrary.books[0].authors=new Array();
var dave=new Array();
dave.age=45;
dave.name="Dave Crane";
myLibrary.books[0].authors[0]=dave;
...
```

JavaScript Object Notation (JS

Use JSON

- **JSON=JavaScript Object Notation**
  - Standard notation is not easy to create large objects
- **Data in an Array indexed with numbers**
  list of objects enclosed in []

  myLibrary.books=[predefinedBook1, predefinedBook2,
  predefinedBook3];

- **Construct a new JavaScript object**
  List of `key:value` pairs enclosed in {}

  myLibrary.books={
  bestSeller : predefinedBook1,
  cookbook : predefinedBook2,
  spaceFiller : predefinedBook3

JSON (Cont.)

- **We can define more complicated objects by merging the syntaxes** (or use functions to fill the content),

```
var myLibrary={
    location : "my office",
    keywords : ["AJAX", "PHP", "JSP", "Servlets"],
    books : [
        { title : "Ajax in Action",
            authors : [
                { name : "Dave Crane", age=45 },
                { name : "Eric Pascarello", age="41" }
            ],
            publicationDate : new Date(2006,04,01)
        },
        { ... }
}
```

JSON (Cont.)

- **We can define and use member functions**

```
function giveDate(){
    return new Date(2005,10,5);
}
var harryPotter={
    title : "Harry Potter and the Half Blood Prince",
    authors : [ {name: "J. K. Rowling", age : 42}],
    publicationDate : giveDate();
    summerize : function(){
        var summary = this.title+" by "
            +this.authors[0].name
            +" was published in "+ this.publicationDate ;
        alert summary;
        }
    }

...
harryPotter.summerize();
```

JSON (Cont.)

- **We can mix JSON and Standard Javascript Notation**

```
var numbers={ one : 1 , two:2, three:3};
numbers.five=5;
```

Browser implement the "Sandbox" Principle

- **The javascript originating from one server can only connect this server**
- **AJAX program can not be used to merge many source of information**
- **It makes sens: harder to write a Distributed Deny of Service**

Using JSON instead of AJAX

JSON for contacting many servers

- **We can use JSON to contact another server.**
  - We add some code in the page requesting a javascript file
  - But this file is generated by a program
  - One parameter is the function to be executed on the return value
- **Example**
  - We want to produce the same result as with Ajax example (see the detail of a patient)
  - We have an html page, refering a JavaScript file
  - when a value is selected inside the page, a function is executed
  - The function adds a `<script>` element in the head of the document
  - This element is calling a PHP program that selects the right patient
  - A `callback` function is called by this script when returned.

JSON Example

json4.php

```
<form>
Select a Customer:
<select name="patients" onchange="showPatient(this.value)">
<option value="DOJ2">John Doe
<option value="BIE1">Emmanuel Benoist
<option value="KNR1">Reto Koenig
</select>
</form>
<p>
<div id="txtHint"><b>Patient info will be listed here.</b></div>
</p>
```

JSON Example

json4.js

```
function showPatient(str) {
  var url='json4GetPatient.php?q=';
  url+=str;
  url+="&loopbackdata=objectData&loopback=responseOK";
  url=url+"&sid="+Math.random();
  script = document.createElement('script');
  head=document.getElementsByTagName('head')[0]
                  || document.documentElement;
  script.src=url;
  head.appendChild(script);
}
function responseOK() {
  var text="<table><tr><th>First_Name</th><th>Last_Name</th↘
  →></tr>";
  text += "<tr><td>"+objectData.first_name+"</td><td>"+↘
  →objectData.last_name+"</td></tr></table>";
    document.getElementById("txtHint").innerHTML=text;
}
```

JSON Example I

json4GetPatient.php

```
<?php
$q=$_GET["q"];
$loopback = $_GET["loopback"];
$loopbackdata =$_GET["loopbackdata"];
$db_table = array( 'DOJ2' => array(
                    'FirstName' => 'John',
                    'LastName' => 'Doe',
                    'Age' => 45,
                    'Hometown' => 'Biel',
                    'Job' => 'Writer'
                    ),
                'BIE1' => array(
                    'FirstName' => 'Emmanuel',
                    'LastName' => 'Benoist',
                    'Age' => 40,
                    'Hometown' => 'Paris',
                    'Job' => 'Professor'
```

JSON Example II

```
            ),
        'KNR1' => array(
            'FirstName' => 'Reto',
            'LastName' => 'Koenig',
            'Age' => 34,
            'Hometown' => 'Bern',
            'Job' => 'Professor'
            ));
$row = $db_table[$q];
// we generate a JavaScript array in PHP
echo "var " .$loopbackdata."={'first_name':'";
echo $row['FirstName']."','last_name':'" .$row['LastName'];
echo "','age':" .$row['Age']."," .'hometown':'" .$row['Hometown'];
echo "','job':'" .$row['Job']."'};\n";
echo $loopback."();\n";
?>
```

Constructors

Objects, Classes and Prototypes

- **Java is fully object-oriented**
  - Everything is an Object (`java.lang.Object`)
  - An object has a Type and a Class

    MyType myObject = new MyClass();

  - Classes can extend an existing Class and implement interfaces
- **Javascript has another type of Objects**
  - Each Object belong to one unique class
  - It has capabilities to link new member functions and member variables dynamically.
  - We can use Prototypes to instantiate objects with a default set of functions and variables (like a new in Java).

Constructor

- **We can create a new object**

  ```
  var myObj=new MyObject();
  ```

- **using a constructor**- which is not a class but rather a *"function"*

  ```
  function MyObject(name,size){
    this.name=name;
    this.size=size;
  }
  var myObj=new MyObject("laptop","35cm");
  alert("size␣of␣"+myObj.name+" is "+myObj.size);
  ␣␣
  ```

Constructor (Cont.)

- **We can also define a member function**

```
function MyObject(name,size){
  this.name=name;
  this.size=size;
  this.tellSize=function(){
    alert("size of "+this.name+" is "+this.size);
  }
}
var myObj=new MyObject("laptop","35cm");
myObj.tellSize();
```

Constructor (Cont.)

- **It works, but is problematic**
  - We create the same function for each instance of `MyObject`. Risks of Memory Leaks if the number of instances becomes large.
  - We created a *closing* which is harmless but can be dangerous if included in the DOM.
- **We use a better solution: The Prototype**

Prototypes

Prototype

- **Functions and properties can be tied to the prototype of a constructor**
  - Each time the constructor function is executed with a `new`,
  - the properties and functions of the prototype are attached to the new object.

```
function MyObject(name,size){
  this.name=name;
  this.size=size;
}
MyObject.prototype.tellSize=function(){
  alert("size of "+this.name+" is "+this.size);
}
var myObj=new MyObject("laptop","32cm");
myObj.tellSize();
```

Extending existing classes

- **In JavaScript, you can incorporate native objects in your programms**
  - They are written in C++ or Java
- **Let us consider the `Array` class**

```
Array.prototype.indexOf=function(obj){
  var result=-1;
  for (var i=0;i<this.length;i++){
    if(this[i]==obj){
      result=i;
      break;
    }
  }
  return result;
}
```

Extending existing classes (Cont.)

- **We can also add other methods**

```
Array.prototype.contains=function(obj){
  return (this.indexOf(obj)>=0);
}
Array.prototype.append=function(obj,nodup){
  if(!(nodup && this.contains(obj))){
    this[this.length]=obj;
  }
}
var numbers=[1,2,3,4,5];
var got8=numbers.contains(8);
numbers.append("cheese",true);
```

Reflexion

Reflexion

- **It is possible to discover the type and functionalities of unknown objects**
- **We can test if an object supports a given method or has a given property**

```
if(MyObject.someProperty){
...
}
```

  - It does not work it the value of `someProperty` is `false` (or simply `0` or `null`).

- **we can do this more properly (much like the php `isset` function)**

```
if(typeof(MyObject.someProperty) != "undefined"){
```

Reflexion (Cont.)

- **If we want to test the type of an object**

```
if(myObj instanceof Array){
   ...
} else if (myObj instanceof Object){
   ...
}
```

- **Or test our self-defined classes**

```
if(myObj instanceof MyObject){
   ...
}
```

- **Restrictions**
  - JSON can only create `Arrays` and `Objects`
  - Any `Array` is also an `Object` (take care to the order of the test)

Reflection (Cont.)

- **Iterator over the properties and functions of an object**

```
function MyObject(){
  this.color='red';
  this.flavor='strawberry';
  this.azimuth='45 degrees';
  this.favoriteDog='collie';
}
var myObj=new MyObject();
var debug="discovering ....\n";
for(var i in myObj){
  debug+=i+'' -> ''+myObj[i]+"\n";
}
alert(debug);
```

Encapsulation

- **It is not possible to extend classes or have interfaces. We have to use Prototypes instead.**
- **Encapsulation is also part of any OO-framework, that does not exist in JavaScript**
  - It can be "emulated" using Duck Typing:
    "If it walks like a duck and sing like a duck, then it is a duck"
  - It is heavy testing the input and you have to rely on the quality of your team
  - But it is the only thing we have

Methods and Functions

Function as First Class Objects

Function as First Class Objects

- **In Java:**
  - Functions belong to a class and/or an object and can not live without it
- **In JavaScript**
  - Functions are floating entities
  - They have an existance outside the objects (can be transfered for instance)

Attach functions to an Object

- **We can simply define a function**

  function doSomething(x,y,z){...}

- **Or using the syntax of the definition of a variable**

  var doSomething=function(x,y,z){ ...}

- **And we can attache this to an object**

  myObj.doSomethingNew=doSomething;
  myObj.doSomethingNew(x,y,z);

Calling a function from another object

- **The `call` function**
  - Start a method with another object

```
function Tree(name){ this.name = name;}
Tree.prototype.describe=function(){
  alert this.name;
}
function Dog(name){ this.name = name; }

myTree=newTree("Oak")
myDog=new Dog("Blacky");
var tmpFunc=myTree.describe;
tmpFunc.call(myDog);
```

The Function Objects

- **Each function is a `Function`**
  - It extends the `Object` class.
  - It can handle properties and contain functions itself.
- **A function can be executed using its `call()` method** (or its cousin `apply()`

```
function multiply(){ return this.y * this.x;}
myObject=new Object();
myObject.x=3;
myObject.y=4;
myOtherObject=new Object();
myOtherObject.x=5;
myOtherObject.y=4;
MyObject.operation=multiply;
var res=myObject.operation(); // returns 12
res=MyObject.operation.call(myOtherObject); // returns 20
```

Events Handling and Function Context in AJAX

Events Handling and Function Context in AJAX

- **We can define event handling inside HTML taging**

- **Or in the JavaScript programm**

  ```
  function clickHandler(){ alert(this.id);}
  myDiv.onclick=clickHandler;
  ```

- **Or with an anonymous function**

  ```
  myDiv.onclick=function(){ alert(this.id); }
  ```

Assign an Handler to a tag in JavaScript

- **In a javascript controler**

```
function MyObj(id,div){
  this.id=id;
  this.div=div;
  this.div.onclick=this.clickHandler;
}
MyObj.prototype.clickHandler=function(event){
  alert(this.id);
}
```

- **Problems**
  - The method does not return the `id` of the `MyObj`
  - It returns the `id` of the `div`

Conclusion

- **JavaScript is not Java ;-)**
- **You can easyly make a mess**
- **Work very properly to prevent massive errors.**

References

- `http://fr.selfhtml.org`
- `http://www.w3schools.com`
- Crane et al., Ajax in Action