JavaScript

Tomasz Bold
tomasz.bold@fis.agh.edu.pl
D11 – pok. 107

Today

- History
- JS – language and environment
- Libraries
  - jQuery
    - DOM interactions
    - events
    - talking to outside world
- Projects

History of JS

- The affair started by Brendan Eich working for Netscape corp. and was first available with Netscape 2.0 in 95 (a kind of firefoxosaurus)
  - it was called JavaScript, soon after followed by JScript (MS), available in IE3.0
  - It was meant as Java (internet era programming language) for non-professionals
- Then the ECMA standard show up
  - Most recent standard is ECMA-262 from 2011
- Now supported by all browsers and also server side of the web communication (i.e. Node.js)

Who would write programs of which the code is publicly available- no it won't work?

JS language

- JS scripts are run in virtual machine (VM)
  - ECMA standard is quite portable (except the details which happen to be important
  - JS VM is considered to be best performant, best debugged, best ... etc.
    - How much this is the truth statement see Gmail – which is app. entirely based on the JS

- The JS is ”very” simple:
  - Simple types: float, int, string + lists + associative container (a map)
  - Objects which in fact are an associative containers
    - DOM is interacted through the OO abstraction

Environment

- The environment of the JS script is a document, it can add to, remove from it or alternate it in any way you can imagine.
- This is scripting language so the pros. and cons. of scripting languages apply here.
  - no compilation process to verify all statements are correct (syntax checking is performed)
  - The development cycle is code → test → code
  - Most web browsers (say: Firefox, chrome, safari) have scripting console, it can be turned on and you can play with the document
  - there is usually multiple views: the most interesting is the ”console” try to open it and run:
    alert(“Hello World”)
    - your first script has been interpreted and invoked
  - Normally you should work with the console and then embed what have worked in the document.

Embedding scripts in the document

- In the document:

  <script>alert(”Great!");</script>

- In if the script is external:

  <script src=”url/to/theScript.js"></script>

Simple & complex types

- Variable obtains the type by assigning the value to it (scripting rule):
  var myLuckyNumber = 7;
  var pi=3.14;
  var myName="Kowalski", myNick="Kawasaki", pi=3;
  var answer=true;
  var allOfThem=new Array();
  allOfThem[0]="Ewa";
  allOfThem[1]="Irena";
  var student={year: 3, name:"Jerzy", fname:"Jurkowski"};
  alert(student["name"]);
  alert(student.name)

- There is special value: **undefined** when no assignment is made.

- The objects are <u>associative</u> arrays

- Use **var** for local names **var a=1**;, skipping the **var** has quite different meaning → **x =1;** assigns to the variable **x** or if the **x** does not exist makes global variable **x** and assigns to it!

Always use the **var** unless you mean global variables.

Functions

- It can be defined like this:

```
function bla(arg1){
  hello(arg1);
  return arg+2;
}
```

- Functions can become methods of objects once they are assigned to the object property:

```
position = function( px, py) {
  this.x = px;
  this.y = py;
  this.distanceFromCenter = function() {
    return Math.sqrt( this.x*this.x + this.y*this.y);
  }


}


var p1 = new position(3,4);
alert(”x: ”p1.x+ “ y: “ + p1.y+ “  to center: “ +p1.distanceFromCenter());
```

Control statements & misc

- **if** – like in c
- **for** - variant like in c
  for (x in arr) { arr[x]; }
  - python like variant - useful when iterating over the objects elements
    - There is also **while** loop
    - The **switch** statement
- This is not different than in other languages
- There is number of useful objects in JS:
- String, Math, Number, Date, Array, RegExp, Boolean

Normally what comes here is the DOM interaction - only brief info here

- The document contains elements which can be accessed by the ID (unique ID) – this is most efficient

- But the operations on the DOM are so common that the community seek for long time set of handy tools to manipulate the DOM elements in portable and efficient manner
  - this is to find elements
  - alternate them
  - remove
  - add
  - apply styles
  - handle events related to the document elements

- The jQuery seems to be the library of choice at the moment

- There are other very useful things (i.e. Prototype, MooTools)

Before the tour over the jQuery

- The jQuery and other JS libs make use of the **closures**
  - This is not a straightforward concept unfortunately - originates from functional languages (not limited to JS, it is present or can be mimicked elsewhere)
  - Think of following case:
    You've got a cake so you eat it, then get grab coffee and drink it → but there is other option → get a plate put the cake on it and eat later once you also have some coffee. Doesn't it taste better?

    The latter is typical event based execution model which is very useful in web programming (and anywhere else where the **unpredictable time enters the control flow**.

- Let's try to have some glimpse of it: if one calls the function **fn(Arg1, Arg2)**:
  - the **Arg1, Arg2** are available inside the body of the function, usual thing
  - variables defined inside the function body, usual thing
  - so are the variables in the surrounding/outer scope – often called **context** (i.e. the global variables)
  - the context is defined by physical location of the function in the code i.e. inside of other function, in global scope etc.
  - at each function invocation in JS the context is made available (**by reference not by value**)
  - **the function + the context** are called **closure**

Closure by example

- think of simple accumulator (sum of numbers):

```
function stat(){
  var totSum=0;
  return function adder(x) {
    totSum+=x;
    return totSum;
  }
}
s1 = stat(); // s1 is a function (closure)
s1(3); // 3
s1(4); // 7
s2 =stat();
s2(1); // ?? quiz
```

Even with this simple code you need to twist your mind a bit. i.e. the **totSum** after the invocation of stat() **is not destroyed** (as it is out of scope – but remains in lexical scope of the **adder**). Each invocation of the stat() makes new totSum so we can have several accumulator instances.

Think of closures as of way of saving the function and it's surrounding environment for the latter use.

And do not worry to much if it is not clear yet ☺.

Now to the jQuery

- The functionality of the jQuery can be subdivided into:
  - events
  - selectors
  - document traversal & manipulation
  - ajax
  - +data storage, effects,forms etc...
  - jQueryUI – user interface widgets (i.e. calendar)
  - jQuery mobile – for your phone
  - QUnit – for automated testing

- All is very well documented in the jquery.com

- There is number of plugins based on the jQuery (let me list the ones I played with myself):
  - HighCharts for plotting → amazing charts quality, check it!
  - DataTables → extremaly rich and yet simple to use
  - Validation → something you all use
  - autocompleter → kicks in if you enter on the forms “Kra”

Events

- The event is something that happens to the document
- The first document one deals with is the document loaded
  At this moment you can start playing.

```
<script>
$(document).ready(function(){
    // your code

});

</script>
```

The $ == jQuery is main library object. It has very rich functionality. Here it turns the argument passed to the jQuery object.

“ready” means invoke when the document is ready i.e. all text is retrieved to the browser.

Run this function (unnamed one) once this happens. This is closure.

Let’s handle more usual event

- Imagine you want to have stat’s calculator on the page.
  something that calculates the the total, average, dispersion of all entered numbers.
  First we need the field where one enters the number, button to accept it and place to write the result.

<**input id="number**"/>

<**input** type="button" **id="entered"** value="append"/>

<**div id="results"**> this is not visible yet

The script

```
<script>
$(document).ready(function() {
 count = stat();
 sum = stat();
 sumOfSq = stat();
 $("input#entered").click( function() {
   var x = parseInt($("input#number").attr("value"));
   tot = sum(x);
   totsq = sumOfSq(Math.pow(x, 2))
   c = count(1);
   stdev = Math.sqrt(totsq/c - Math.pow(tot/c, 2));
   $("div#results").html("Entered so far: " + c +" numbers, average: " + tot/c + " standard deviation: " +
stdev);
  });
</script>
```

- We have: listen to the button clicked → read the number → increase some counters: sum of all numbers, count of all numbers & sum of squares → fill the page with the content.

The list of events is quite long but the documentation clear.

The event always comes with an event object (which can tell for instance the position of the element, originating element → (useful when one handler serves number of elements).

Right here we enter second subject the selectors

- The elements of the document need to be grabbed somehow before any operation can be don on them

- Here the jQuery offers very rich functionality
  - by tag → ”a” = all anchors in the document
  - by ID using the # → ”div#result”
  - by class using the . → ”p.content”

- By pseudoclasses
  - :first, :last
  - :odd, :even
  - :first-child, :last-child, :nth-child(arg)
  - :empty, :visible, ... you have to see the documentation

- By relation to other elements
  - parent > child → ”body > p” = all main paragraph
  - parent descendant ....

- By attributes
  - [attr] → ”a[href]” = all the links on the page
  - [attr ~=’name’] → ”a[href~=’http’]” = external links

Document manipulation

- As you have seen we can add the text easily using the **html(”bla”)**

- All the content of the document can be alternated in fact by one of this methods:
  - **after, before, append, prepend, clone, remove, replaceAll, replaceWith, wrap**

  add a link icon ↵ in each link, make space after each link

  $(”a”).append(”↵”).after(” ”);

This shows another nice feature of jQuery, selection can be performed once and number of operations can be done on the set of elements.

The special modifiers

- The visual aspects of the elements can be changed via the **css(”attr”, ”value”)** → **$(”a”).css(”color”, ”red”);** = all the links are red now

- Predefined sets of the visual aspects: the CSS classes can be added by **addClass** or removed **removeClass**

Effects

- Listen now, most of the projects will be about them because it is fun

- The visual aspects can be applied to elements in various way i.e. gradually change the color

- The simplest effects are hide, show
  i.e. **$("a").hide()** hides all links on the pageh

- There are other nice effects **fadeIn(), fadeOut(), slideDown/Up()**

- **animate()** → this in fact can do anything you imagine

jQuery and ajax (adżaks)

- The ajax = **a**synchronous **ja**va script e**x**ecution
- The RII require constant connection with the outside world (databases in numerous forms) in order to fulfill the task
  - Think of auto completion in the form
- The ajax facilitates that communication demand by outsourcing the communication to java script
- The task is usually quite simple: send request to the server (with arguments) I get the response and fill the content of the document (i.e. get the picture and place it in the frame)

The jQuery way

- Simplest of all, data in some unusual form:
  $**.get**(**url**, function(data, status) {
  /* format and add somewhere*/
  });
  - there is **$.getJSON()** version which uses commonly used format: JavaScript Objects Notation (almost like python dict dumped with pprint)
  - there is also **$.getScript()** which downloads JS code and executes it

- $(”body”)**.load**(url) → content added to the body (any elements can be used here)

- **$.post**(url) → when you need to send to the server some data

Testing the JS

- The JS scripts are rarely for you only. They are meant to be used by other people by construction.
- So you have to assume they other person is ignorant and asked for the age will write: ”twenty four”
  - Validate your input
  - Test the code
- There is couple of systems which can do testing of the JS but we stick to the one used by jQuery: QUnit
- The biggest problem usually in JS testing is to find ”the units”
  - this is easy in our stats calculator though

Guide to JS testing in steps

```
function stat() {
  var totSum=0;
  return
function(x) {
    totSum += x;
    return
totSum;
  }
}
```

```
function statsCalculator() {
    var count = stat();
    var sum = stat();
    var sumOfSq = stat();
    return function(x) {
        var tot = sum(x);
        var totsq = sumOfSq(Math.pow(x, 2))
        var c = count(1);
        var stdev = Math.sqrt(totsq/c - Math.pow(tot/c,
2));
        return {"count": c, "average": tot/c, "stdev":
stdev};
    }
};
```

- Separate code to another file (JS file) → we will need to use it in two places.
  - The page where it is supposed to work
  - and the one where the **test run**.

Not so much different but refactored a bit: no DOM access involved, returns explicit, self explanatory object.

Create test document

```
<link rel="stylesheet" href="qunit.css">
<script src="qunit.js"></script>
<script src="stat.js"></script>
 <script>
 test("stats calculator unit test", function() {
  var calc = statsCalculator();
  var o = calc(1);
  equal(o.count, 1, "count is ok");
  equal(o.average, 1, "average starts ok");
  equal(o.stdev, 0, "std of single entry is 0");

  var o = calc(1);
  equal(o.count, 2);
  equal(o.average, 1);
  equal(o.stdev, 0);

  var o = calc(4);
  equal(o.count, 3, "count is growing as expected");
  equal(o.average, 2, "average ok");
  equal(o.stdev, 1.4142135623730951, "stdev ok");
});
</script>
```