Javascript Notes

E-Applications Spring 2015

M. Lampis

Acknowledgment

- The material on these slides follows the excellent book “Speaking JavaScript” by Axel Rauschmayer
- http://speakingjs.com/es5/

Introduction

- Javascript is a scripting language (doh!)
- Dominant for client-side web programming
- We will be using it inside a modern browser (e.g. Firefox)
- All modern browsers come with a javascript engine
- Javascript is (generally) an interpreted language: the user is given the **source code**

Relationship with other languages

- Javascript is **not** really related to Java.
  - There are some common points between the two
  - Syntax is similar to C-family languages (C++/Java)
- Javascript is more **functional**
  - Similar to Lisp/Scheme in some respects
- Javascript is relaxed with types
- Javascript is relaxed with objects

Up and running

- Basic way to run a javascript program
  - Include in an HTML file, between <script> </script> tags.

```
<script>
alert(“Hello world”);
</script>
```

- Can also use the js console of a browser
  - In Firefox Ctrl+Shift+K

Basic Structure

- A js program is a series of **statements**
  - **Statements** should be separated by ;
  - This is (tried to be) done automatically! (more later)
- Statements resemble Java:
  - if (condition) { statement } else {statement}
  - for (var i=0 ; i<5; i++) { statement }
  - while() { }, do { } while();
  - switch() { }

Basic Structure

- Variables must (should) be declared with the **var** keyword

  ```
  var x=5;
  ```

- Observe that no type is specified, this is found in run-time and can change.

  ```
  x = 'abc';        // no problem
  ```

Basic Structure

- Arrays use C-like notation

```
var arr = [ 'a', 2 , 'c' ];  //different types OK
arr.length == 3
arr[2] == 'c'
```

- Elements are indexed from 0 up to arr.length-1
- OK to add elements!

```
arr[3] = 'd'; // arr == [ 'a', 2, 'c', 'd']
```

Basic Structure

- Objects in js are more like maps/dictionaries than Java objects
- Again the . (dot) operator is used to access methods/properties

```
var myobj = { }; // empty object

var obj2 = { key1: 'val1', key2: 15 };

obj2.key2 == 15 //true

obj2.key3 = 'hello'; //OK to add fields!
```

- Object/Array variables are references (Java-like?)
- Arrays are objects! (verify with **typeof**)

Basic Structure

- Functions are declared using the **function** keyword

```
function fact(n){
    if(n<1) return 1;
    return n * fact(n-1);
}
```

- Argument types are not specified

Basic Web Page Interaction

- JS programs have a “global” object
  - For programs running in a browser → **window**
- Inside window object one finds the **document** object
  - This gives methods to access HTML elements
  - More details to be discussed later (DOM)
  - Important to know:

    document.getElementById(“..”)

    method that returns a reference to an HTML element

Basic Web Page Interaction

- Annoying input output
  - alert(“msg”); (no return value)
  - confirm(“msg”); (boolean return)
  - prompt(“msg”,”default”); (string return)
- Less annoying
  - Give an id to an input textbox
  - Access via document.getElementById().value

Basic Web Interaction

- Event-driven programming
  - Basic idea: the web page **waits** for the user to do something (generate an event) and respond
- Events:
  - Mouse click, mouse movement, window resizing, …
- Events can be caught by adding appropriate attributes to the relevant HTML tags

```
<input type=”button” onclick=”some js...” />
```

More Javascript

- In the remainder we give some more details on various useful features of javascript
- Emphasis on points of difference with other, more familiar languages (Java)

Strings

- One of the **primitive** data types in js

  var x = “abcd”;

- Can use either kind of quotes (“ or ')
  - Be consistent!
- \ is an escape character
  - ex. x='He\'s using quotes!';
- No “char” type for single characters (everything is a string)

String functionality

- Strings are not references
- String literals are immutable

  ```
  a = "abc";  // ->"abc"
  a[2] = 'd';  // “ignored”, a == “abc”;
  ```

- Expressions that produce a new string are OK

  ```
  a += 'd'; // a == “abcd”;
  ```

- Note the array-like access
  - This can be done with .charAt(i) method also

String Conversions

- Other values can be converted to strings, usually easily
- Manual way: `String( expr );`
  - `' ' + expr; // Dirty!!`
- Note that conversions are inconsistent sometimes

```
Boolean(String(false)) != false // ???
```

String Operators

- Comparisons (<, >, <=, >=, ===)
  - Work OK (alphabetically)
  - Not reliable for international characters (accents etc.), use localeCompare

    ```
    'é'.localeCompare('f') // gives -1
    ```
- + performs concatenation
- .length gives the length of a string

String Operations

- The .split method splits a string into an array of strings, using the given separator

  ```
  'a b c'.split(' '); // → Array [ "a", "b", "c" ]
  ```

- The separator can also be a regular expression (very useful, see later)
- .toUpperCase, toLowerCase (self-explanatory)
- .indexOf(sth) finds the index where sth appears in a string (could be -1, sth could be reg ex)

Booleans

- true or false values
- Operators &&, ||, !
- Careful with conversions:

```
Boolean(0) == false, Boolean(123) == true
Boolean('') == false, Boolean('a') == true
Boolean([]) == true (!) //all arrays
Boolean({ }) == true (!) //all objects
```

Booleans

- Logical operators are short-circuited
  - `false || x === x;  true && x === x;`
- Application: setting default value to a parameter

  `function f(x,y,z){ y = y || some_value; ... }`
- Uses the fact that undefined is converted to false
- NOTE: This may not be what you want!

  (ex. If y = 0)
- Recall also ternary operator x ? y : z

Numbers

- No distinction between ints and floats
- Standard operators +, -, *, /, %
- Standard function Math.abs(), Math.floor(), Math.round()
- Number(expr) → convert expr to number
- parseInt (expr) → convert expr to STRING   then integer
- Specials: NaN (never equal to anything!), Infinity

Non-primitive types

- We have seen the primitive types
  - String
  - Boolean
  - Number
- Everything else is non-primitive
  - Object (also arrays and reg exps)
  - Function
  - Undefined (this is a special type!)

Arrays

- Arrays are objects! (see with typeof)
- ...with many useful properties pre-defined
  - arr.length gives the length of an array
  - Can be used to shorten/lengthen array!
  - We can also use .push() to add an element to the end of an array and .pop() to remove it.

Arrays with holes and more

- It's allowed to have some “missing” (undefined) positions in an array.
- These are called holes.
- → Arrays are **maps**
- Usually, arrays without holes are optimized → faster
- Arrays are also allowed to have arbitrary properties (they are objects)

2-d Arrays

- 2dimensional arrays can be defined indirectly:
- Construct an array **rows**
  - Each element of this array should be an array
  - Now possible to say rows[2][3] = 5;
- Exercise: construct and print a 2-d array of size 3x3 with the numbers 0,1,...,8

Array operators

- The **in** operator checks if a given index exists/is not a hole
  - This will also return true for non-index properties (can be used for objects)
- Can be used to iterate through an array
  - `for(var key in arr) { do sth with arr[key]; }`
- Bad idea!
  - Skips holes (maybe not bad?)
  - Iterates through other keys (?)

Array Iterations

- Standard (C/C++/Java) way
  - for (var i = 0; i<arr.length; i++) { … };
- Use the forEach method (only available for arrays, not array-like objects such as strings)
  - arr.forEach(alert); //NOT arr.forEach(alert());
  - Argument is a function that is to be applied to each element of the array
  - Skips holes

Array methods

- .sort() will sort the array (doh!)
  - Caution! Sorting will first convert elements to strings → lexicographic sorting
  - Can give an optional function argument that decides the order of two elements
  - [1,2,3,20].sort((function(x,y){ x<y? -1: (x>y?1:0)})); //gives [1,2,3,20]
  - [1,2,3,20].sort(); //gives [1,2,20,3]

Searching

- .indexOf(elem) returns the first index where elem occurs, or -1
- .lastIndexOf(elem) returns the last index
- Interesting: can never find NaN (since it is not equal to anything)
- Uses strict equality === (more later)

Arrays exercise

- Write a function that counts the elements of an array
- .length will also count the holes...
  - Hint: easier with a “temporary” function

Functions

- Three roles of functions in javascript
  - Normal functions
    - function f(args) {..};   … f(expr);
  - Constructors
    - new Object(…);
  - Methods
    - myObj.doSomething(…);

Function definitions

- The usual

```
function add(x,y){
    return x+y;
    //other things ignored... (?)
}
```

Function variables

- We can use a **function expression**

```
var add = function (x,y){
    return x+y;
    //other things ignored... (?)
}
```

- Now the typeof add is function
- These two are almost(!) equivalent

Hoisting

- Functions are hoisted
  - This means that no matter where in scope a function is defined it is implicitly moved to the beginning of the scope
- Variables are hoisted
  - Their scope is the whole function (blocks are ignored)
- But variable assignments are not hoisted!

Function expressions

- Function expressions can be named
  - This can make them recursive
  - `var superf = function f(x) { return x<1? 1: x*f(x-1);};`
  - Here, f is only accessible within f.
  - But superf is a variable that can be called from outside
  - The name “f” can be accessed with the property superf.name

Checking passed parameters

- Functions can be called with **more** or **less** parameters than defined
  - JS will not complain (!)
- Useful to check the special **arguments** object
  - Array-like (but not array)
  - .length tells us the number of actual parameters

```
function alertArgs() {
    for(var i=0;i< arguments.length; i++)
        alert("arg "+i+" = "+arguments[i]);
}
```

Does a parameter exist

- Easy answer: check if it is undefined
  - `if (x===undefined) {…}`
- Similar
  - `if (!x) {…}`
- Recall how to set default values
  - `x = x || default;`

Pass By Value

- All function calls are normally pass-by-value

```
function inc(x) { x++;};
var y=0;
inc(y); //no effect
```

- One workaround: Arrays (which are refs)

```
function inc(x) { x[0]++;};
var y=[0];
inc(y); //y[0]==1
```

Careful with function signatures

- Meet the .map() method of Arrays

  ```
  [1,2,3].map( function(x){ return x+2;} );
  ```

- How about the following?

  ```
  [“1”,”2”,”3”].map( parseInt );
  ```

- This fails because map feeds the given function 3 parameters (element, index, array)
- The function in the first example ignores the other 2
- parseInt does not

One caveat for return

- Recall that ; are automatically inserted where missing (!)

```
var x=5
var y=3 //no problem
```

- How does JS know when they are missing?
  - New line starts unexpectedly
  - Block ends unexpectedly
  - ...

One caveat for return

- Consider the following:

```
return { foo: “bar” };
```

- Or

```
return
{
    foo: “bar”
};
```

- Not equivalent!

The eval function

- The eval function takes as input a string
- The string is evaluated as js code
  - Similar to writing something on the console
  - Use case: evaluating arithmetic expressions given by the user
  - Careful: allowing the user to evaluate arbitrary things may not be a good idea
  - On the other hand, this code is running on the client...

Other problems: dangling else

- Dangling else problem (also in C/Java)
  - `if (test1) if(test2) {...} else {...}`
- When is else executed?
  - When test1 is false?
  - When test1 is true and test2 is false?
- Answer: please use { } to make clear
- Answer: else is matched to closest if

Reminder: the switch statement

- Also present in C/C++

```
function useFruit(fruit) {
  switch (fruit) {
    case 'apple':
      makeCider();
      break;
    case 'grape':
      makeWine();
      break;
  }
}
```

Reminder: Exceptions

- Work similarly to Java/C++

```
try{
    throw("OOPS!");
} catch(exception) {
    alert(exception)
}
```

Regular expressions

- Can be given between / and /
  - Special characters:
  - ? match 0 or 1 time
  - * match 0 or more times
  - + match 1 or more times
  - . any character
  - [ ] range/group of characters
- Examples
  - / *, */ → any amount of whitespace that includes a comma
  - / *,? */ → any amount of whitespace that may include a comma
  - /[1-9][0-9]*/ → a non-empty integernumber

An application: split

- The String split(sep) method splits a string into an array of strings, using the separator sep
- sep can be a string or a reg exp
- Examples:

```
"1,2,3,4".split(",");
→ Array [ "1", "2", "3", "4" ]
"1 ,2 , 3 , 4".split(",").map(Number);
→ Array [ 1, 2, 3, 4 ]
"1 ,2 , 3 , 4".split(/ *, */);
→ Array [ "1", "2", "3", "4" ]
```