WorkSheet 1 - JavaScript Fundamentals

February 11, 2013

1 Higher Order Functions

Filters, Cumulators and Transformers

Exercise 1.1 *Write a function $filter$ that receives as input an array arr and a function f and outputs that same array without the elements for which f evaluates to $false$. Remark: the algorithm should be linear in the size of the array and you should not allocate a new array.*
Example:

```
function pair(x) {
  return !(x%2);
}
var arr = filter([1,3,4,6,8], pair) // [4,6,8]
```

Exercise 1.2 *Write a function map that receives as input an array arr and a function f and outputs that same array after applying function f to each one of its elements. Example:*

```
function f(x) {
  return x*x;
}
var arr = map([1,3,4,6,8], f) // [1,9,16,36,64]
```

Exercise 1.3 *Consider the following function:*

```
var accumulator = function(arr, f, v) {
  for(var i=0, len=arr.length; i<len; i++) {
     v = f.call(null, v, arr[i]);
  }
  return v;
}
```

Example:

```
function soma(x,y){
  return x+y;
}
var v = accumulator([1,3,4], soma, 0) // 8
```

Without using the two previous functions, write a new function, processor, that is simultaneously a filter, an accumulator and a transformer. First, it filters, then it transforms and finally applies the accumulator. processor must have the following signature:

```
processor(arr, filter, transformer, accumulator, v0)
```

Examples:

```
var f1 = function() {
   return !(x%0);
}
var f2 = function(x) {
   return x*x;
}
var f3 = function(x, y){
   return x+y;
}
processor([1,3,3,2,4,9], f1,f2,f3,0) // 20
```

Rewrite the previous three functions using the function processor.

Exercise 1.4 *Like in exercise 1.1, write a filter that instead of receiving as input an array, receives as input an object.*

```
var obj1 = {p: 3, q: 2};
var pair = function(x) {
return !(x%2);
};
var obj2 = filter(obj1, pair) //{q: 2}
```

Exercise 1.5 *Using the functions developed in the previous exercises:*

1. *Write a function that receives as input an array of numbers and removes the odd values.*
2. *Write a function that receives as input an array of numbers and computes the product of all the numbers in the array.*
3. *Write a function that receives an array of numbers and outputs the maximum.*

2 Objects and Prototypical Inheritance

Exercise 2.1 *Implement the following class hierarchy:*

```
Child, Adult << Person << Animal
Dog << Animal
```

```
Child.saySomething() { super.saySomehting(); alert('I\'m a child!'); }
Adult.saySomething() { super.saySomething(); alert('I\'m an adult!'); }

Person.saySomething() { alert('I\'m'+Person.name); }
Person.makeASound() { alert('I can speak'); }
Dog.makeASound() { alert('ao ao '); }

Animal.breathe() { alert('this animal is breathing'); }
```

Test your class hierarchy using QUnit.

Exercise 2.2 *Write a function f that receives an object o as input and outputs a new object o' having o as a prototype.*

Exercise 2.3 *A publish/subscribe object is an object on which you can register and fire "events" (in the context of this exercise, they shall be considered as strings).*

```
o.subscribe('eventName', 'function', 'subscriber');
o.publish('eventName');
```

When you publish an event, all functions that subscribed it must be invoked on the corresponding subscribers (receiving object o as an argument).

Exercise 2.4 *Write a function that receives a string as an argument, and outputs an object that maps each word of the given string to the number of anagrams of that word that occur in the original string. Example:*

```
s = "o gato vestiu uma toga"
countAnagrams(s) // {o : 0, gato: 1, vestiu: 0, uma: 0, toga: 1}
```

3 Closures

Exercise 3.1 *Detect the problem in the following program:*

```
function openLinksInNewWindows() {
   for (var i = 0; i < document.links.length; i++) {
      document.links[i].onclick = function() {
          window.open(document.links[i].href);
          return false;
      }
   }
}
```

Solve it without changing $document.links[i].href$.

Exercise 3.2 *Taking advantage of JavaScript scoping rules, write a function inspect that once invoked allows you to evaluate an arbitrary expression in the environment on which it was invoked. Use the prompt() window method to interact with the user. Example:*

```
function f() {
  var n = 0;
  inspect(...);
}
f();
```

When inspects evaluates, the user should receive a prompt to enter an expression that will be evaluated in the current scope. So, for example, if the user enters n, there should be an alert with 0.
Hint: Perhaps inspect() must receive an argument.

Exercise 3.3 *Write a function proxy that receives as input a function f and an object o and outputs a new function f' such that every time f' is invoked, f will be called on object o ($o.f()$). Write an example of how this function can help you avoiding the problems that can arise from scope confusions.*

Exercise 3.4 *Implement a function that takes a function as an argument and returns a memoized version of that function, that is, a version of that function that chaches results. The memoized version of the original function should have an internal table and, when invoked on an argument, it should first check if the argument is in the table. If it is, it returns the value associated with the argument immediately. If it is not, the function computes the result normally, stores it in the table, and finally returns.*