Exercises – JavaScript

# Lab 1 – Introduction

## Install Browser

Install the Google Chrome Browser (recommended) on your machine (if it is not already there) and make it your default browser.

## Write Hello World

Create two files:

Hello World.html

And

Hello World.js

1. In Hello World.js, write a JavaScript function that prints “Hello World” in the console.
2. In Hello World.html, make a reference to Hello World.js and make sure the function is called.
3. Open Hello World.html in Chrome, open the console (F12) and observe the result.
4. Declare a variable: me
5. Assign an object literal to me that holds your name and city
6. Print it in the console

# Lab 2 – Types, Conversions, & Operators

## Baseline Questions

1. With new knowledge gained, answer again the first eleven questions (shown below).
2. Let your favourite JavaScript engine evaluate the first eleven expressions and verify your answers.

1. null == null

2. undefined == undefined

3. null == undefined

4. NaN == NaN

5. 2 == "2"

6. 2 + "2"

7. "2" \* "2"

8. true && false

9. 2 && 4

A. 4 && 2

B. null || 7

## Reasoning about Expressions

Look at the code below.

1. What would be the value of the four expressions below?
2. What steps (conversions) have been taken and what triggered those conversions?

var x = "1";

var y = 1;

x + y /\* 1 \*/

x \* y /\* 2 \*/

x == y /\* 3 \*/

x ? 2 : 3 /\* 4 \*/

## Puzzle (if time permits)

1. Find a value x for which:  
    +!x === 1
2. Check your answer by writing a little JavaScript program.
3. Find *all* different values for which this holds.

# Lab 3 – Arrays

## Baseline Questions

1. With new knowledge gained, answer again the questions (shown below).
2. Let your favourite JavaScript engine evaluate the expressions and verify your answers.

C. { x:null } || 7

D. [1,2,3] || [3,4,5]

E. ~[4]

F. { x:null } == { x:null }

10. { x:null } === { x:null }

## Arrays

Write the following functions on arrays:

function contains(array, item)

// returns true if the array contains the item

// returns false otherwise

function add(array, item)

// adds the item to the array, if it is not yet included

// does nothing, otherwise

function remove(array, item)

// removes the item from the array, if it is included

// does nothing, otherwise

function sum(array)

// returns the sum of all elements

## Puzzle (if time permits)

1. What is the value of x in:  
    var x = [ , , ].length;
2. What is the value of x in:  
    var x = ++[[]][+[]]+[+[]];
3. If you have found the answers, then evaluate the expressions in a browser.

# Lab 4 – functions

## Baseline Questions

1. With new knowledge gained, answer again the questions (shown below).
2. Let your favourite JavaScript engine evaluate the expressions and verify your answers.

11. test(3)

12. test("2")

13. test(4, 2)

14. test.a + 5

15. test.b

16. test.c

17. test["b"] + 7

## Functions

Write a function with an arbitrary number of parameters:

function sum(/\*arbitrary number of parameters\*/)

// returns the sum of all parameters

Write a function with an optional parameter:

function sayHello(/\*optional: name\*/)

// writes ‘Hello’ to the console,

// or ‘Hello <name>’ if a name is provided

// (hint: try using the || operator)

## Revealing Module Pattern

Build a *reflection API* that exposes the following methods:

functionName(funct)

// returns the name of the function as a string, if the function has a name,

// returns the empty string, if it is an anonymous function,

// returns undefined, otherwise

(hint: try using regular expressions)

hasProperty(object, propertyName)

// returns true, if the object has the propery with that name

// returns false, otherwise

(hint: try using the in operator)

hasProperties(object, name\_1, name\_2, … , name\_n)

// returns true, if the object has all listed properties

// returns false, otherwise

(hint: this function has one or more arguments)

This reflection API must be usable as follows:

function add(a,b) { return a + b; };

var obj = { x: 4, y: 7, sum: add };

reflection.functionName(vector.sum) // => "add"

reflection.hasProperty(obj, "x") // => true

reflection.hasProperties(obj, "x", "sum", "y") // => true

## Namespace Pattern

Make the reflection API a member of the namespace com.yourCompany.common.

## Puzzle (if time permits)

Make a function

overload(function\_with\_n\_parameters, function\_with\_m\_parameters)

that throws a TypeError, if both function have the same number of parameters,

and otherwise, returns an function that

when called with n parameters, forwards the call to function\_with\_n\_parameters,

when called with m parameters, forwards the call to function\_with\_m\_parameters,

when called otherwise, throws a TypeError

(hint: every function has a length property that returns the number of formal paramters in the definition of that function)

var createVector = overload(

function (a, b) {

return { x: a, y: b };

},

function (length) {

return { x: length / 1.414, y: length / 1.414 };

}

);

createVector(3, 4) // => { x: 3, y: 4 }

createVector(7.07) // => { x: 5, y: 5 }

# Lab 5 – objects

## Baseline Questions

1. With new knowledge gained, answer again the questions (shown below).
2. Let your favourite JavaScript engine evaluate the expressions and verify your answers.

18. typeof test

19. typeof test.prototype

1A. test.toString()

1B. "b" in new test()

## Prototypes

Add a getName() method to the Function prototype. Implement it using the functionName(funct) in the reflection API from exercise 9.

## Classes

Write a Customer class in the namespace com.yourCompany.CRM.

1. Write a class Customer that has the following properties:

id, name, city, nrOfUnpaidBills

and the methods:

buyStuff(), payBill()

and the constructor

Customer(id, name, city)

when a Customer is created, the nrOfUnpaidBills equals 0. Each time buyStuff() is called, the nrOfUnpaidBills is increased by 1, and each time payBill() is called, the nrOfUnpaidBills is decreased by 1.

1. Make the property nrOfUnpaidBills private (not accessible outside the Customer object), and add a method

badPayer(int n)  
 that returns true, if the nrOfUnpaidBills is n or more

and returns false, otherwise.

1. Override the toString() method for Customer objects. It should return id, name, and city, formatted like:  
    "(id) name - city"

## Event Class

Write a helper class (in the namespace com.yourCompany.common) that implements an event mechanism.

1. Write a class EventHandler, that has

a private array

listeners

and public methods

addListener(listener)

adds the listener to the array of listeners, if the listener is a function

fails silently, otherwise

removeListener(listener)

removes the first occurrence of the listener in the array of listeners, if the listeners array contains this listener (same object reference)

fails silently, otherwise

invoke(sender, args)

iterates through the array of listener functions and invoke all functions by passing them the sender and args objects.

1. Change the Customer class.  
   Add a property  
    unpaidBillsChanged  
   and assign it a new EventHandler object in the Customer constructor.
2. Invoke the unpaidBillsChanged event handler each time the nrOfUnpaidBills changes value, passing the customer object and an object literal with a bills property that contains the new number of unpaid bills.
3. Write the following code and see that information is printed in the console:

function accountant(sender, args) {

console.log(sender.toString());

console.log(args.bills);

}

var bob = new Customer(102, "Bob", "Utrecht");

bob.unpaidBillsChanged.addListener(accountant);

bob.buyStuff();

// prints "(102) Bob - Utrecht" and 1 to console

bob.payBill();

// prints "(102) Bob - Utrecht" and 0 to console

## Puzzle (if time permits)

For long namespaces, the namespace pattern can get a bit out of control.

var com;

(function(ns, undefined) {

//...

}((com = com || {},

com.infoSupport = com.infoSupport || {},

com.infoSupport.someProject = com.infoSupport.someProject || {},

com.infoSupport.someProject.v1 = com.infoSupport.someProject.v1 || {},

com.infoSupport.someProject.v1.model = com.infoSupport.someProject.v1. model || {} )));

Let’s try to make this somewhat less cumbersome.

It would be more convenient to write:

!function(ns, undefined) {

//...

}(comNS("com.infoSupport.someProject.v1.model"));

Try to implement this solution.

1. Add a file comNS.js .
2. Free programmers from the need to declare var com; each time.
3. Define this comNS function .