JavaScript Exercises (Day-1)

Node – Getting started

Hopefully you have already installed *Node.js*. If not, do it ;-)

Open a command prompt and type: `npm install -g learnyounode`

This will install a small Node module with a series of Node.js workshops and the ability to test and mark your solution as solved.

Create a folder in which to do the following exercises and type: `learnyounode`.

Complete 1+2 just to get a feeling about how to use node (we will complete all next time)

The magic of callbacks:

The JavaScript array has a number of cool iteration methods, that all take a callback as a parameter, like `forEach()`,`filter()`, `map()`, `reduce()` and many more.

In the following exercises we are first going to use these basic methods, to recap our knowledge about what they do, and then, we are going to implement the methods by our self, as if they did not already exist.

1) Using existing functions that takes a callback as an argument

Using the **filter** method:
Declare a JavaScript array and initialize it with some names (Lars, Jan, Peter, Bo, Frederik etc.). Use the filter method to create a new array with only names of length <=3.

Using the **map** method:
Use the names-array created above, and, using its `map` method, create a new array with all names uppercased.

2) Implement user defined functions that take callbacks as an argument

Now, assume the array did not offer these two methods. Then we would have to implement them by our self.

Implement a function: `myFilter(array, callback)` that takes an array as the first argument, and a callback as the second and returns a new (filtered) array according to the code provided in the callback (that is with the same behaviour as the original filter method).

Test the method with the same array and callback as in the example with the original `filter` method.

Implement a function: `myMap(array, callback)` that, provided an array and a callback, provides the same functionality as calling the existing `map` method on an array.

Test the method with the same array and callback as in the example with the original `map` method.

3) Using the Prototype property to add new functionality to existing objects

*Every JavaScript function has a prototype property (this property is empty by default), and you can attach properties and methods on this **prototype** property. You add methods and properties on an object's prototype property to make those methods and properties available to all instances of that Object. You can even implement (classless) inheritance hierarchies with this property.*

The problem with our two user defined functions above (myFilter and myMap) is that they are not really attached to the Array Object. They are just functions, where we have to pass in both the array and the callback[1].

Create a new version of the two functions (without the array argument) which you should add to the Array prototype property so they can be called on any array as sketched below:

```
var names = ["Lars", "Peter", "Jan", "Bo"];
var newArray = names.myFilter(function(name) {…});
```

4) Getting really comfortable with `filter` and `map`

a) Use `map()` to create to create the 's for an unordered list and eventually a string like below (use `join()` to get the string of 's):

```
<ul>
  <li>Lars</li>
  <li>Peter</li>
  <li>Jan</li>
  <li>Bo</li>
<ul>
```

b) Use `map()+(join + ..)` to create to create a string, representing a two column table, for the data given below:

```
var names = [{name:"Lars",phone:"1234567"}, {name: "Peter",phone:
"675843"}, {name: "Jan", phone: "98547"},{name: "Bo", phone: "79345"}];
```

c) Create a single html-file and test the two examples given above.

Hint: add a single div with an id=names, and use DOM-manipulation `(document.getElementById.innerHTML = theString)` to add the `ul` or `table`.

d) Add a button with a click-handler and use the `filter` method to find only names with a length >3.

Update the `ul` and/or the `table` to represent the filtered data.

---

[1] It's a generally accepted design rule that you should never add new behaviour to JavaScript's built in objects. We do it here, only to introduce the prototype property

Hosting

Team up with another member of the class. Read about hoisting (use literature suggested for period-1) and implement at least two examples (individually) to illustrate that:

- Function declarations are completely hoisted
- var declarations are also hoisted, but not assignments made with them

Explain to each other (as if it was the exam):

- What hoisting is
- A design rule we could follow, now we know about hoisting

`this` in JavaScript

Team up with another member of the class. Read about `this` in JavaScript (use the literature suggested for period-1) and implement at least three examples (individually) to illustrate how *this* in JavaScript differs from what we know from Java. One of the examples should include an example of explicit setting `this` using either `call(), apply()` or `bind()`.

Explain to each other, using the examples (as if it was the exam):

- How this in JavaScript differ from this in Java
- Why we (because we did not explain about this) followed a pattern in our third semester controller and stored a reference to this (var self = this)
- The purpose of the methods `call(), apply()` and `bind()`

Immediately Invoked Function Expressions

Team up with another member of the class. Read about `Immediately Invoked Function Expressions (IIFE)` in JavaScript (here you probably have to Google, and find your own references).

Implement 1-2 examples to illustrate its purpose.

Explain to each other, using the examples (as if it was the exam):

- The purpose of Self Invoking Functions (just another name for the same thing)

Objects

1)

Create an object with four different properties, with values, of your own choice (ex: name, birthday, hobby, email).

Use a foreach loop to demonstrate that we can iterate over the properties in an object.

Use the delete keyword to demonstrate we can delete existing properties from an object (delete a property, and iterate over the properties again)

Use the function (inherited from Object) `hasOwnProperty()` to test whether a property exist (directly) on your object (test with both and existing, and non-existing property).

2)

Create a Constructor function to create new Persons having:

- a firstName, lastName and an age.
- A method to get details about the Person

3)

Create an object of your own choice and list all properties in the object using one of the function given here "Enumerate the properties of an object":

Delete one of the properties in your object, and use the method above to verify that this is possible in JavaScript.

The need to enumerate all properties in an object is a very common problem, and something you should get familiar with.

Reusable Modules with Closures

1)

Implement and test the Closure Counter Example from the Slides

2)

Implement a reusable function using the Module pattern that should encapsulate information about a person (name, and age) and returns an object with the following methods:

- setAge
- setName
- getInfo (should return a string like Peter, 45)

All of it

Don't do this exercise in the class. Do it someday where you have some spare time and just want to summarize all you previous JavaScript knowledge.
Make yourself a nice cup of coffee, tea or perhaps even grab a beer ;-) and go to:
http://bonsaiden.github.io/JavaScript-Garden/

Read all the text and execute all examples.