JavaScript, ECMAScript 2015 en TypeScript

CM Telecom – 1 februari 2016

Peter Kassenaar

info@kassenaar.com

Peter Kassenaar

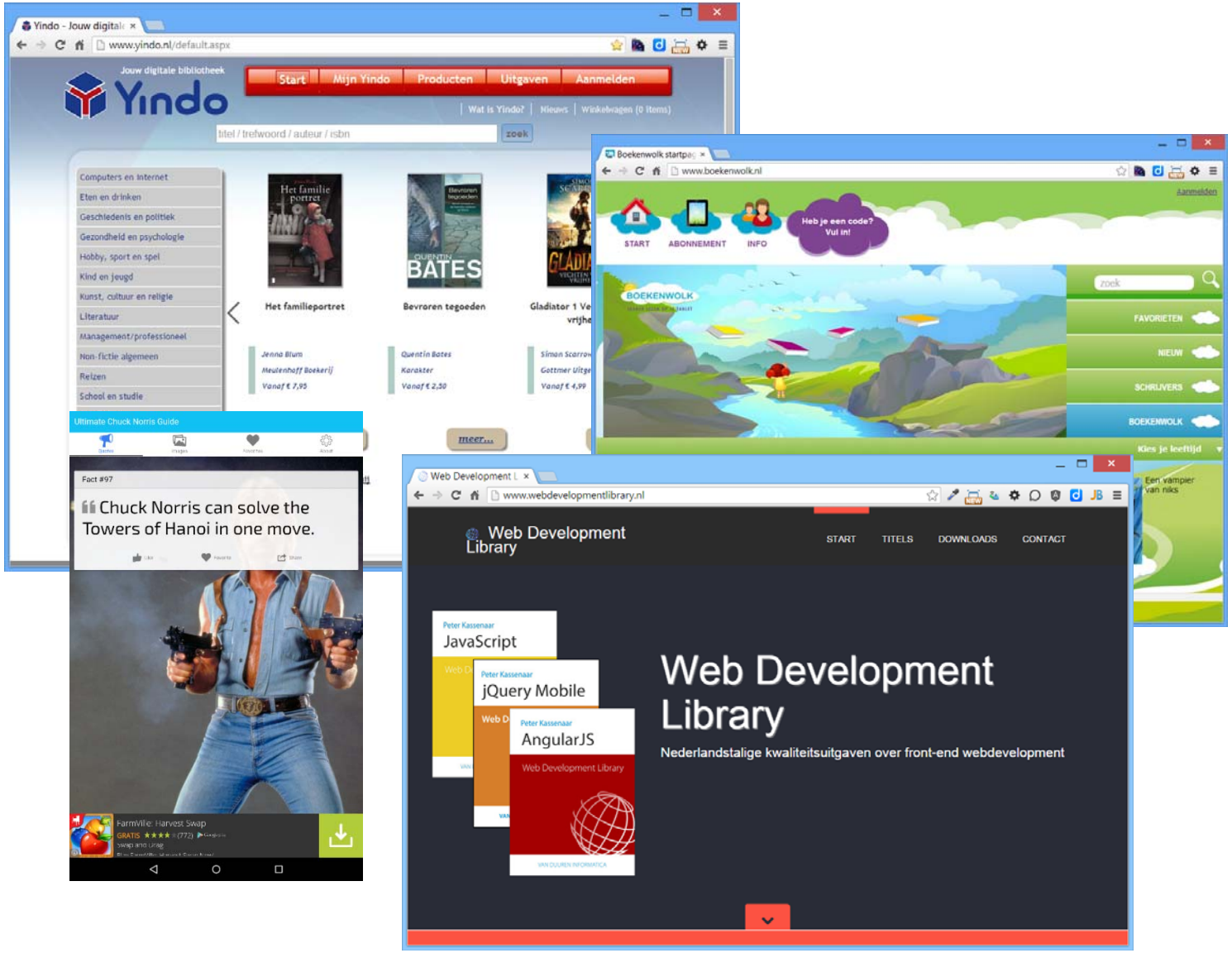
Over Peter Kassenaar:

- Trainer, auteur, developer – sinds 1996
- Specialisme: *"Everything JavaScript"*
- JavaScript, ES6, AngularJS, NodeJS, jQuery, PhoneGap, TypeScript

www.kassenaar.com/blog

info@kassenaar.com

Twitter: @PeterKassenaar


Yindo - Jouw digitale
www.yindo.nl/default.aspx
Jouw digitale bibliotheek
Yindo
Start
Mijn Yindo
Producten
Uitgaven
Aanmelden
Wat is Yindo?
Nieuws
Winkelwagen (0 items)
titel / trefwoord / auteur / isbn
zoek
Computers en internet
Eten en drinken
Geschiedenis en politiek
Gezondheid en psychologie
Hobby, sport en spel
Kind en jeugd
Kunst, cultuur en religie
Literatuur
Management/professioneel
Non-fictie algemeen
Reizen
School en studie
Het familieportret
Bevroren tegoeden
Gladiator 1
Jenna Blum
Meulenhoff Boekerij
Vanaf € 7,95
Quentin Bates
Karakter
Vanaf € 2,50
Vanaf € 4,99
meer...
Boekenwolk startpag
www.boekenwolk.nl
Aanmelden
START
ABONNEMENT
INFO
Heb je een code?
Vul in!
BOEKENWOLK
zoek
FAVORIETEN
NIEUW
SCHRIJVERS
BOEKENWOLK
Kies je leeftijd
Een vampier van niks
Ultimate Chuck Norris Guide
Fact #97
Chuck Norris can solve the Towers of Hanoi in one move.
FarmVille: Harvest Swap
GRATIS
Web Development L
www.webdevelopmentlibrary.nl
Web Development Library
START
TITELS
DOWNLOADS
CONTACT
Peter Kassenaar
JavaScript
jQuery Mobile
AngularJS
Web Development Library
Nederlandstalige kwaliteitsuitgaven over front-end webdevelopment

Over jullie…

Voorkennis webdevelopment, (mobile/web-) apps?

Kennis JavaScript, TypeScript?

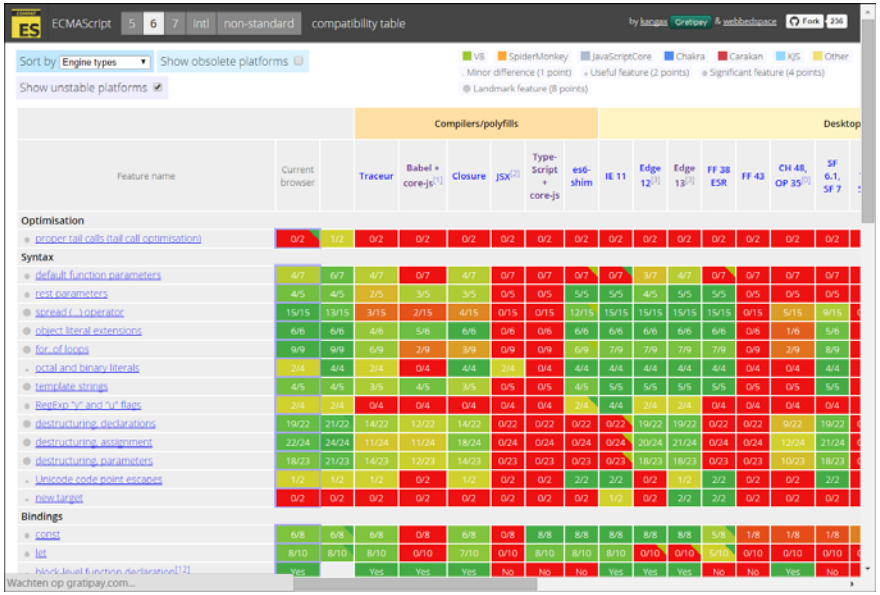
Voorkennis andere frameworks of (web-)talen?

Verwachtingen van de cursus?

Concrete projecten?

Materialen

Software	(downloads)
Handouts	(PPTX/PDF)
Oefeningen	(papier)
Websites	(online)

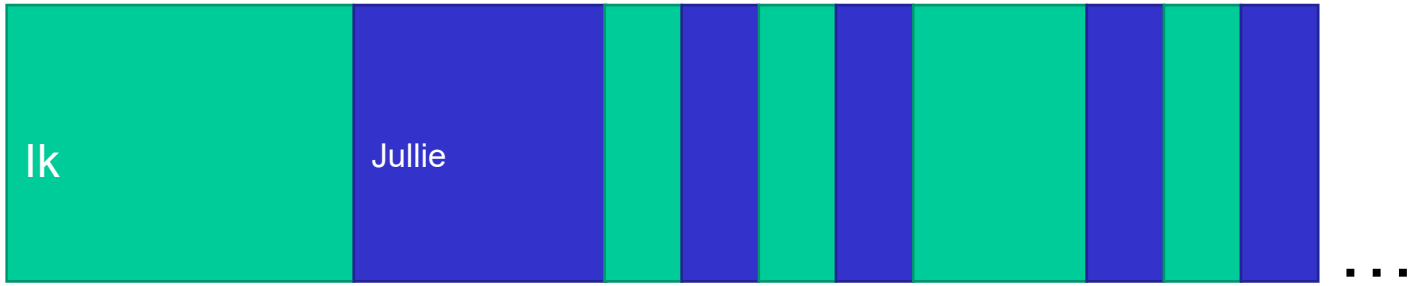


https://kangax.github.io/compat-table/es6/

Agenda

- JavaScript / ECMAScript 5 advanced concepts
  - Design patterns
  - Prototypal Inheritance
- ECMAScript 2015
  - New keywords
  - New parameters options
  - Transpiling
- TypeScript
  - Using TypeScript
  - Typing variables and functions
  - Classes
  - Interfaces / Generics

Globale werkwijze



Vragen?

ES5 Advanced concepts

Closures, design patterns, inheritance

Credits : Dan Wahlin

Blog

http://weblogs.asp.net/dwahlin

Twitter

@DanWahlin

pluralsight hardcore developer training

http://www.pluralsight.com

Contents

- ***Problem*:**
  - Function Spaghetti Code
- **Closures to the Rescue**

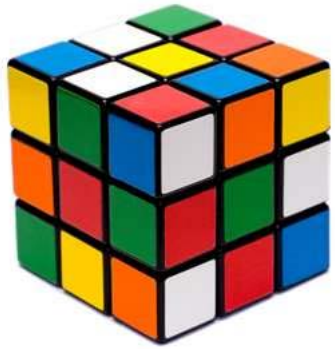
- ***Possible solutions:***
  - Object Literals and Namespaces
  - 1. Prototype Pattern
  - 2. Revealing Module Pattern
  - 3. Revealing Prototype Pattern

Function Spaghetti Code

```
<script type="text/javascript">
    window.onload = function () {
        currNumberCtl = document.getElementById('currNumber');
        eqCtl = document.getElementById('eq');
    };
    var eqCtl,
        currNumberCtl,
        operator,
        operatorSet = false,
        equalsPressed = false,
        lastNumber = null;
    function add(x,y) {
        return x + y;
    }
    function subtract(x, y) {
        return x - y;
    }
    function multiply(x, y) {
        return x * y;
    }
    function divide(x, y) {
        if (y == 0) {
            alert("Can't divide by 0");
            return 0;
        }
        return x / y;
    }
    …
```

Problems with Function Spaghetti Code

- Mixing of concerns
- No clear separation of functionality or purpose
- Variables/functions added into global scope
- Potential for duplicate function nam
- Not modular, not distributable
- Not easy to maintain
- No sense of a “container”



Better - Ravioli Code

Advantages of Ravioli Code

- Objects encapsulate and decouple code
- Loosely coupled
- Separation of Concerns
  - Variables/functions are scoped
  - Functionality in *closures*
- Easier to maintain

What is a Closure?

"…an inner function always has access to the vars and parameters of its outer function, even after the outer function has returned."

~ Douglas Crockford

Non-Closure Example

```
function myNonClosure() {
    var date = new Date();
    return date.getMilliseconds();
}
```

Variable lost after function returns

Closure Example

```
function myClosure() {
    var date = new Date();

    //nested function
    return function () {
        return date.getMilliseconds();
    };
}
```

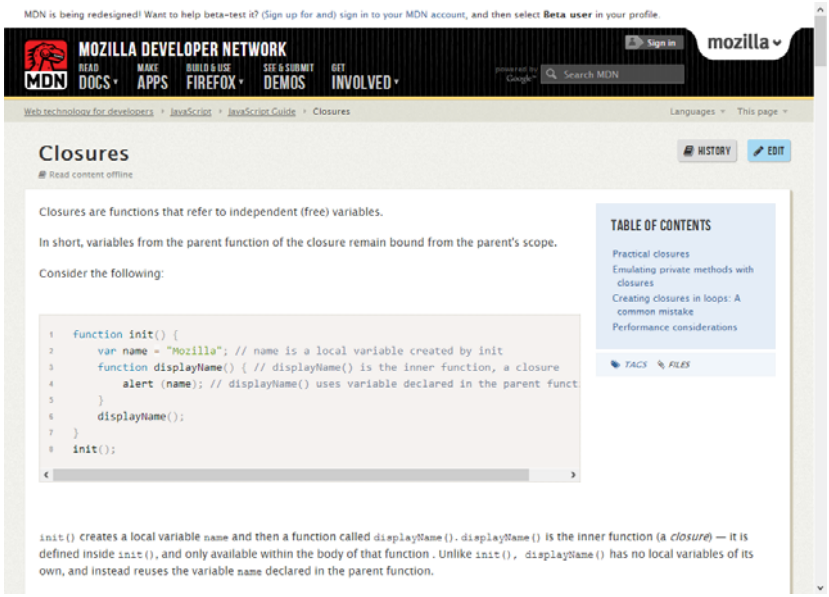
Variable stays around even after function returns

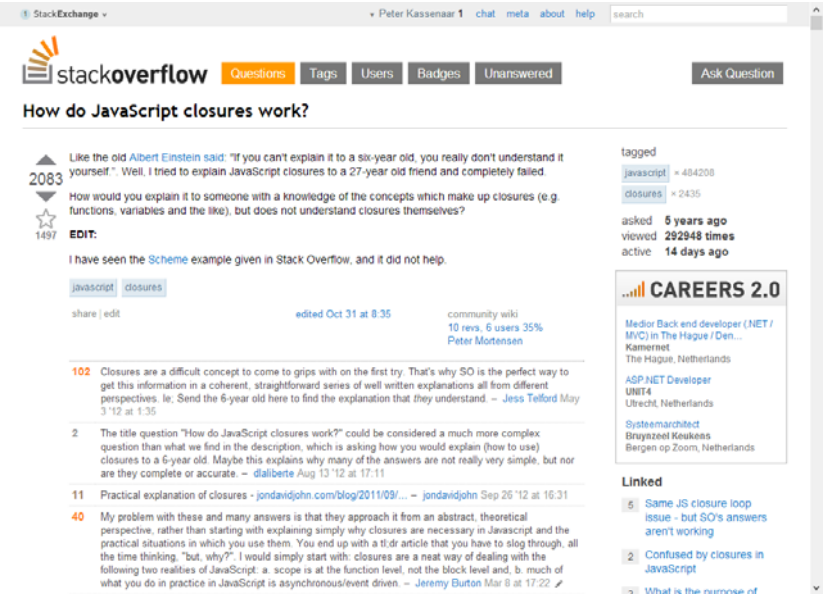
Other Example

- Force correct order of count/parameters between async (AJAX) calls

```
49          //***************************
50          // 2. Verbeterde poging, nu met closure
51          //***************************
52          // Load initial data: de getoonde pagina en de vorige- en volgende pagina. 3 in totaal.
53          var index = -1;
54          BW.pagesHaveLoaded = false;
55
56          // for-lus, ophalen 3 pagina's.
57          for (i = 0; i < 3; i++) {
58              // JavaScript-closure, om te voorkomen dat i een andere waarde heeft als de bitmapstring is
59              // opgehaald door de AJAX-call (en dus de pagina's in verkeerde volgorde in de SwipeView-array
60              // terechtkomen). Credits: http://stackoverflow.com/questions/2405064/ajax-call-in-for-loop-wont-return-values-to-correct-array-positions
61              (function (i) {
62                  page = i == 0 ? BW.slides.length - 1 : i - 1;
63                  $.ajax({
64                      type: 'GET',
65                      url: BW.slides[page].src,
66                      success: function (yindoPermission) {
67                          el = document.createElement('img');
68                          el.className = 'loading';
69                          el.src = '' + yindoPermission.pageBitmapString;
70                          el.onload = function () {
71                              this.className = '';
72                              this.className = BW.deviceOrientation;
73                              $('#loadingIndicator').hide();
74                              // <snip>
75                          };
76                          BW.gallery.masterPages[i].appendChild(el); // <== NU gaat het goed, want i is geisoleerd in de scope van de closure.
77                          index++;
78                      }
79                  });
80
81              })(i); // Nieuwe scope creeren door de functie aan te roepen met huidige i als parameter.
82          }
83
```

Further reading on closures





https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Closures

http://stackoverflow.com/questions/111102/how-do-javascript-closures-work

Achieving Ravioli Code

Patterns to the rescue

Simplest - Encapsulating Data with Object Literals

- Quick and easy
- All members are available
- Best suited for data

```
var calc = {
    tempValue: 1,
    add: function(x, y){
        return x + y;
    }
};
```

Namespaces

- **Encapsulate your code in a namespace:**

```
var myNS = myNS || {};

myNS.calc = {
    tempValue: 1,
    add: function(x, y) {
        return x + y;
    }
};
```

Verdict: quick and simple. Suitable for smaller projects, where no modularity or (re)distribution is required

JavaScript patterns

Prototype pattern, Revealing module pattern, Revealing prototype pattern

1. The Prototype Pattern

- **Benefits:**
  - Leverage JavaScript's built-in features
  - "Modularize" code into re-useable objects
  - Variables/functions taken out of global namespace
  - Functions loaded into memory once
  - Possible to "override" functions through prototyping

- **Challenges:**
  - "this" can be tricky
  - Constructor separate from prototype definition

Prototype Pattern Structure

```
//Constructor defines properties and inits object
var Calculator = function (eq) {
   this.eqCtl = document.getElementById(eq);
   …;
};
//Prototype defines functions using JSON syntax
Calculator.prototype = {
   add: function (x, y) {
      return x + y;
   },
   subtract: function (x, y) {
      return x - y;
   },
   …
}
```

```
var calc = new Calculator('eqCtl');
calc.add(2,2);
```

2. The Revealing Module Pattern

- **Benefits:**
  - "Modularize" code into re-useable objects
  - Variables/functions taken out of global namespace
  - Expose only public members

- **Challenges:**
  - Functions may be duplicated across objects in memory when not using singleton
  - Not easy to extend
  - Some complain about debugging

Revealing Module Pattern Structure

```
var calculator = function() {
    var eqCtl,
        …,

        init = function(equals, currNumber) {
            eqCtl = equals;
            currNumberCtl = currNumber;
        },

        add = function(x, y) {
            return x + y;
        },

        subtract = function(x, y) {
            return x - y;
        },

    return {
        init: init,
        add: add,
        …
    };
}(); // <== IIFE patroon
```

```
calculator.add(2,2);
```

3. The Revealing Prototype Pattern

- **Benefits:**
  - Combines Prototype and Revealing Module patterns
  - “Modularize” code into re-useable objects
  - Variables/functions taken out of global namespace
  - Expose only public members
  - Functions loaded into memory once
  - Extensible
- **Challenges:**
  - "this" keyword can be tricky
  - Constructor separate from prototype definition

Revealing Prototype Pattern Structure

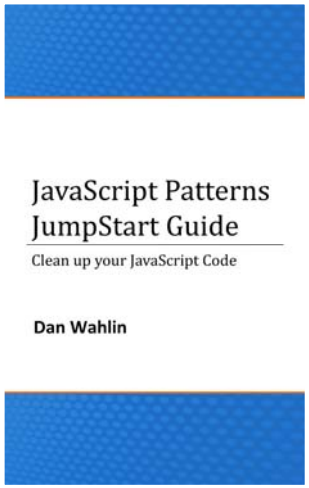
```
var Calculator = function (eq) {
    this.eqCtl = document.getElementById(eq);
};

Calculator.prototype = function() {
    var doAdd = function (x, y) {
        var val = x + y;
        this.eqCtl.innerHTML = val;
    };
    return {
       add : doAdd
     };
}();
```

```
var calc = new Calculator('eqCtl');
calc.add(2,2);
```

Summary

- **Function spaghetti code is bad, Ravioli code is good**
- **Closures provide encapsulation**
- **Key patterns:**
  - 1. Prototype Pattern
  - 2. Revealing Module Pattern
  - 3. Revealing Prototype Pattern
- **Which pattern should you use? You decide….**

More info
Sample Code, Book, and Videos



http://tinyurl.com/StructuringJSCode

Structuring JavaScript Code

This course walks through several key patterns that can be used to encapsulate and modularize JavaScript code. Throughout the course you'll learn how closures and other techniques can be used to better organize your JavaScript code and make it easier to re-use and maintain in HTML5 applications.

Authored by: Dan Wahlin
Duration: 2h 10m
diate
Released: 12/12/2011

pluralsight
hardcore developer training

Meer over JavaScript design patterns

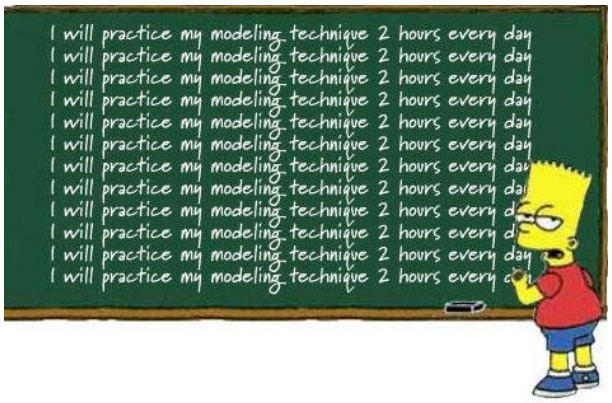
Addy Osmani: Essential JavaScript design patterns

http://addyosmani.com/resources/essentialjsdesignpatterns/book/

Table Of Contents

Oefening

- Bekijk de voorbeelden, of maak zelf een object (`Person`, `Product`, `Car`, …)
- Breidt dit uit op elk van de drie manieren
  - `Prototype pattern`
  - `Revealing module pattern`
  - `Revealing prototype patterns`
- Maak een herdistribueerbare module van je object en bespreek/ leg uit aan collega hoe hij werkt.



JavaScript inheritance

OO met klassiek JavaScript

JavaScript inheritance

Algemeen: maak een ParentClass, of BaseClass

```
// parent class. Convention: classes start with Uppercase
function ParentClass(){
    this.parentProp1 = 'Hello';
    this.parentMethod1 = function (arg1){
        return arg1 + 'parent method 1 return data';
    }
}
```

Maak een afgeleide klasse, of ChildClass

```
//child class
function ChildClass(){
    this.childProp1 = 'Goodbye';
    this.childMethod1 = function (arg1){
        return arg1 + 'Child method 1 return data';
    }
}
```

Zorg er voor dat het prototype van ChildClass verwijst naar ParentClass en maak de instanties

```
// make the child class inherit all parent class characteristics by
// using prototype property.
ChildClass.prototype = new ParentClass();

// instantiate class
var instance1 = new ChildClass();
```

Optioneel : override methods uit de ParentClass

```
// if desirable, you can override the parent methods.
// Again, use prototype
ChildClass.prototype.parentMethod1 = function parentMethod1(arg1){
    return arg1 + 'I have overridden parent method 1';
}
```

http://youtu.be/pu08qQCmw8I

Oefening

- Maak zelf een Parent – Child relatie tussen classes
  - Bijvoorbeeld : voertuigen, personen, producten, etc.
  - Instantieer verschillende instanties en check of je in de child-classes members of methodes uit de parent-class kunt gebruiken.
  - Override een methode uit de parent-class in de child-class en test of dit werkt.