CSC 470 – Section 3

Topics in Computer Science: Advanced Browser Technologies

Mark F. Russo, Ph.D.

Spring 2016

Lecture 6

Eloquent JavaScript: Chapter 5 and 10

Namespacing In JavaScript

Currently, there is no way in JavaScript to create formal namespaces.

This is required in order to prevent properties from one part of a program to hide/replace properties from another.

Multiple strategies have been developed for simulating namespacing in JavaScript

1. Object literal notation
2. Module Pattern

https://toddmotto.com/mastering-the-module-pattern

Object Literal Notation

- Object as namespace.
- All items in namespace defined as keys of a single object
- Avoids polluting the global namespace – limited to one variable def'n
- Patterns to create object if does not already exist

```
if(!NS) NS = {};
var NS = NS || {};
var NS = NS === undefined ? {} : NS;
```

- Statements may be placed at top of multiple JS files
  - Object is initialized only once in a program
- Items can be added at will to `NS` object
- Arbitrary nesting is possible – objects within objects

Objects Defined in Multiple Files

```html
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Parts</title>
    <script type="text/javascript" src="part1.js"></script>
    <script type="text/javascript" src="part2.js"></script>
    <script type="text/javascript" src="part3.js"></script>
  </head>
  <body>
    <textarea id="log"></textarea>
    <script type="text/javascript">
      $$$.func1();
      $$$.func2();
      $$$.func3();
    </script>
  </body>
</html>
```

```javascript
// part1.js
// Use existing object or create new one
var $$$ = $$$ || {};

// Add function to $$$
$$$.func1 = function() {
  var log = document.getElementById("log");
  log.innerHTML += "This is func1\n";
};
```

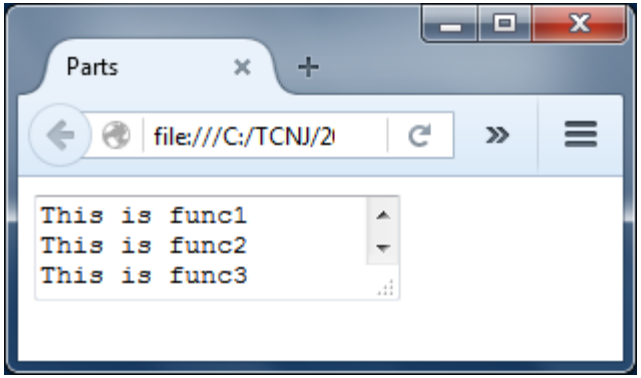
```javascript
// part2.js
// Use existing object or create new one
var $$$ = $$$ || {};

// Add function to $$$
$$$.func2 = function() {
  var log = document.getElementById("log");
  log.innerHTML += "This is func2\n";
};
```

```javascript
// part3.js
// Use existing object or create new one
var $$$ = $$$ || {};

// Add function to $$$
$$$.func3 = function() {
  var log = document.getElementById("log");
  log.innerHTML += "This is func3\n";
};
```

Objects Defined in Multiple Files



Module Pattern

- Built with an IIFE
- Returns an object that contains the public interface only

- Private functions/variables
  - Declared in IIFE but not returned in object
  - Returned object reserved for public interface
- Public functions
  - Declared as functions that ARE returned as part of object
  - Closure ensure public function scope has access to private functions and variables via scope chain
- Module importing
  - Pass any other object/module as argument to IIFE
  - Enters scope of IIFE

Module Example

```
// module.js

// Demonstration of the module pattern
// Combines IIFE's, closures, and objects as namespaces

// A module that manages an array of words
var manager = (function() {

  // Private module elements are not returned
  // as part of returned object

  // All words under management
  var _words = [];

  // Words must not contain spaces
  var _valid = function(word) {
    if (word.contains(" ")) {
      return false;
    } else {
      return true;
    }
  };

  // Check if word is on list already
  var _already_exists = function(word) {
    if (_words.includes(word)) { return true; }
  };

  // Clear the list
  var clearWords = function() {
    _words = [];
  }

  // continued…
```

IIFE avoids excess pollution of global namespace

Declarations are encapsulated within function scope, including variables and functions

Module Example

```
  // Get the current word count
  var getUniqueCount = function() {
    return _words.length;
  }

  // Add an array of words to word list
  var addWords = function( words ) {

    for (var i=0; i<words.length; i++) {
      var word = words[i];

      // Skip invalid words
      if (!_valid(word)) {
        continue;
      }

      // If word is already on list, skip
      if (_already_exists(word)) {
        continue;
      }

      // Add word to list
      _words.push( word );
    }
    return true;
  };

  // Return public interface
  return {
    addWords      : addWords,
    clearWords    : clearWords,
    getUniqueCount: getUniqueCount
  }

})();
```

Private functions within IIFE function scope may be invoked freely

Definitions intended to make up public interface returned as object literal

Module Example

```
// List of words to manage
words = ['cat', 'dog', 'fish', 'cat', 'brown bear', 'parrot'];

// Add words to manager
manager.addWords( words );

// Get count of unique/valid words
console.log( manager.getUniqueCount() );
```

```
4
```

- Only functions returned as part of object literal may be invoked
- Closure formed by internally defined functions ensures "private" variables and functions in scope chain remain accessible

Array Iteration Methods

Array methods that take functions as arguments that are applied to all elements of the Array

- `find`
- `findIndex`
- `filter`
- `every`
- `some`
- `forEach`
- `map`
- `reduce`
- `reduceRight`

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

find(...)

```
arr.find(callback[, thisArg])
```

Returns a value in the array, if an element in the array satisfies the provided test function. Otherwise undefined is returned.

```
// --- arr.find
// Find an element in the array that satisfies the function
var arr = ['a', 'b', 'd', 'B'];

var result = arr.find( function( val, idx ) {
  return ( val.toUpperCase() === 'B');
});

console.log( result );
```

```
b
```

findIndex(...)

```
arr.findIndex(callback[, thisArg])
```

Returns an index in the array, if an element in the array satisfies the provided test function. Otherwise -1 is returned.

```
// --- arr.findIndex
// Find an element in the array that satisfies the function
// Return the index where it was found
var arr = ['a', 'b', 'd', 'B'];

var result = arr.findIndex( function( val, idx ) {
  return ( val.toUpperCase() === 'B');
});

console.log( result );
```

```
1
```

filter(…)

```
arr.filter(callback[, thisArg])
```

Creates a new array with all elements that pass the test implemented by the provided test function.

```
// --- arr.filter()
// Tests whether some element in the array passes the test
// implemented by the provided function.
// Returns values that pass test

var arr = ['a', 'b', 'd', 'B'];

var result = arr.filter( function(val, idx) {
  return ( val.toUpperCase() === 'B');
});
console.log( result );
```

```
['b', 'B' ]
```

`find()` vs. `filter()`

- `find()` finds and returns the first array element that returns `true` when passed to the given test function

- `filter()` returns all array elements that return `true` when passed to the given test function

every(...)

```
arr.every(callback[, thisArg])
```

Tests whether all elements in the array pass the test implemented by the provided test function

```
// --- arr.every()
// Test if every element in an array satisfies a test function

var arr = ["aaaa", "bbb", "cc"];

var result = arr.every( function(val, idx) {
  return (val.length > 1);
} );
console.log( result );
```

```
true
```

some(...)

```
arr.some(callback[, thisArg])
```

Tests whether some element in the array passes the test implemented by the provided test function

```
// --- arr.some()
// Tests whether some element in the array passes the test
// implemented by the provided test function

var arr = [10, 20, 30];

var result = arr.some( function(val, idx) {
  return (val >= 30);
});
console.log( result );
```

```
true
```

forEach(…)

```
arr.forEach(callback[, thisArg])
```

Executes a provided function once per array element. Returns undefined.

```
// --- arr.forEach
// Repeat the function with every array element as its only argument.
// Returns nothing.
var arr = [10, 20, 30];

var result = arr.forEach( function(val, idx) {
  console.log(val, idx);
});
console.log( result );
```

```
10 0
20 1
30 2
undefined
```

Example: `forEach()`

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Rainbow</title>
  </head>
  <body>
    <div id="rainbow"></div>
    <script type="text/javascript" src="rainbow.js"></script>
  </body>
</html>
```

```
// Add color to the rainbow
var addColor = function(clr) {
  var el  = document.getElementById('rainbow');
  var div = document.createElement('div');
  div.style.paddingTop = '20px';
  div.style.backgroundColor = clr;
  el.appendChild(div);
};
```

http://jrsinclair.com/articles/2016/gentle-introduction-to-functional-javascript-arrays/

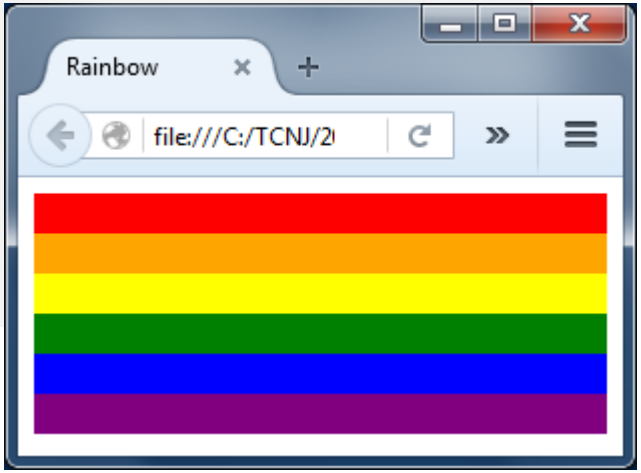
Example: `forEach()`

```
// rainbow.js

(function() {
  // Add color to the rainbow
  var addColor = function(clr) {
    var el = document.getElementById('rainbow');
    var div = document.createElement('div');
    div.style.paddingTop = '20px';
    div.style.backgroundColor = clr;
    el.appendChild(div);
  };

  // Add colors, one at a time
  // Could become VERY tedious depending upon size of array
  addColor('red');
  addColor('orange');
  addColor('yellow');
  addColor('green');
  addColor('blue');
  addColor('purple');

})();
```



06/rainbow.js

Example: `forEach()`

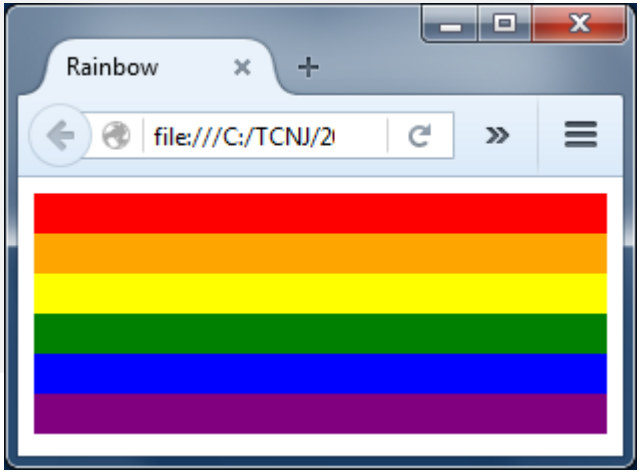
```
// rainbow.js

(function() {
  // Add color to the rainbow
  var addColor = function(clr) {
    var el = document.getElementById('rainbow');
    var div = document.createElement('div');
    div.style.paddingTop = '20px';
    div.style.backgroundColor = clr;
    el.appendChild(div);
  };

  // Add colors using forEach() Array method
  var colors = ['red','orange','yellow','green','blue','purple'];
  colors.forEach(addColor);

  //addColor('red');
  //addColor('orange');
  //addColor('yellow');
  //addColor('green');
  //addColor('blue');
  //addColor('purple');

})();
```



06/rainbow.js

map(...)

```
arr.map(callback[, thisArg])
```

Creates a new array with the results of calling a provided function on every element in this array.

```
// --- arr.map
// Repeat the function with every array element as its only argument.
// Collects and returns results.
var arr = [10, 20, 30];

var result = arr.map( function(val, idx) {
  return 2*val;
});
console.log( result );
```

```
[ 20, 40, 60 ]
```

`forEach()` vs. `map()`

- `forEach()` applies function to all elements of the Array, but returns `undefined`.

- `map()` applies function to all elements of the Array, collects results and returns as a new Array.

reduce(...)

```
arr.reduce(callback[, initialValue])
```

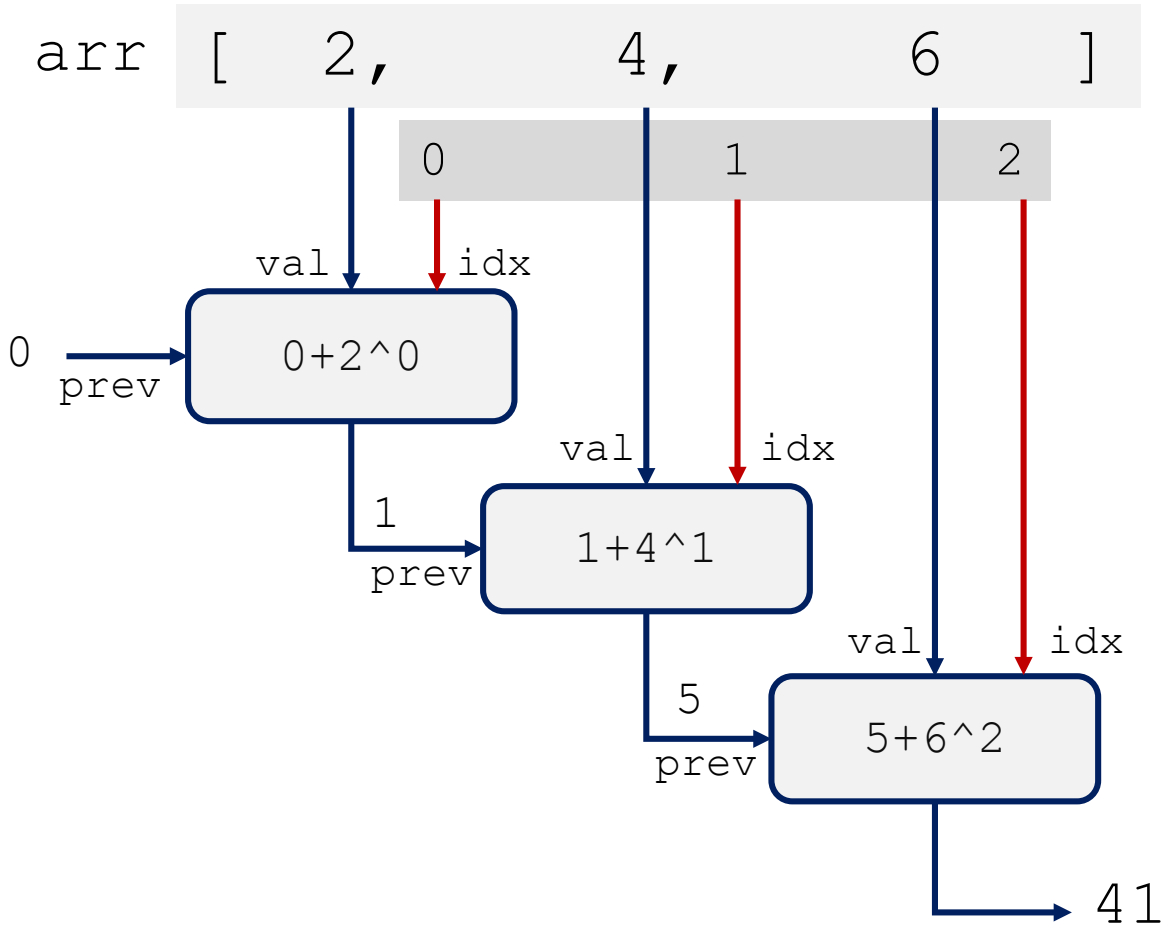
Applies a function against an accumulator and each value of the array (from left-to-right) to reduce it to a single value.

```
// --- arr.reduce
// Apply function to all values, accumulate and return result.
// Initial value passed as last argument.
var arr = [2, 4, 6];

var result = arr.reduce( function(prev, val, idx) {
  console.log( prev + "+" + val + "^" + idx);
  return prev + Math.pow(val, idx);
}, 0);
console.log( result );
```

```
0+2^0
1+4^1
5+6^2
41
```

Example: `reduce(...)`



reduceRight(…)

```
arr.reduceRight(callback[, initialValue])
```

Applies a function against an accumulator and each value of the array (from right-to-left) has to reduce it to a single value.

```
// --- arr.reduceRight
// Apply function to all values, accumulate and return result.
// Initial value passed as last argument, right to left.
var arr = [2, 4, 6];

var result = arr.reduceRight( function(prev, val, idx) {
  console.log( prev + "+" + val + "^" + idx);
  return prev + Math.pow(val, idx);
}, 0);
console.log( result );
```

```
0+6^2
36+4^1
40+2^0
41
```

Examples: `reduce()`

- My Little Pony Data Set
- An Array of Arrays of two-element Arrays
- Convert to an Array of Objects



```
// ponies.js

// A list of lists of two-element lists
var ponies = [
  [
    ['name', 'Fluttershy'],
    ['image', 'http://tinyurl.com/gpbnlf6'],
    ['description', 'Female Pegasus pony in My Little Pony Friendship is Magic.']
  ],
  [
    ['name', 'Applejack'],
    ['image', 'http://tinyurl.com/gkur8a6'],
    ['description', 'Female Earth pony in My Little Pony Friendship is Magic.']
  ],
  [
    ['name', 'Twilight Sparkle'],
    ['image', 'http://tinyurl.com/hj877vs'],
    ['description', 'Primary character of My Little Pony Friendship is Magic.']
  ]
];
```

http://mlp.wikia.com/wiki/My_Little_Pony_Friendship_is_Magic_Wiki

Examples: `reduce()`

```
// reduce1.js

var addToObject = function(obj, pair, idx) {
  //console.log(JSON.stringify(obj), JSON.stringify(pair));

  // Add key:value to obj passed forward
  obj[pair[0]] = pair[1];

  // Return object for subsequent call
  return obj;
};

// Convert first list in ponies data to an object.
var lol = ponies[0];
var ob = lol.reduce( addToObject, {} );

// Serialize object and print
console.log(JSON.stringify(ob));
```


```
{ "name":"Fluttershy",
  "image":"http://tinyurl.com/gpbnlf6",
  "description":"Female Pegasus pony in My Little Pony Friendship is Magic." }
```


http://jrsinclair.com/articles/2016/gentle-introduction-to-functional-javascript-arrays/

06/reduce1.js, 06/reduce1.html

Examples: `map()` + `reduce()`

```
// map_reduce.js

var addToObject = function(obj, pair, idx) {
  //console.log(JSON.stringify(obj), JSON.stringify(pair));

  // Add key:value to obj passed forward
  obj[pair[0]] = pair[1];

  // Return object for subsequent call
  return obj;
};

// Convert list of list pairs to an object.
var ponyToObject = function(lol) {
  return lol.reduce(addToObject, {});
}

// Map function over all
var ob = ponies.map( ponyToObject, ponies );

// Serialize object and print
console.log( JSON.stringify(ob) );
```


```
[ { "name":"Fluttershy", "image":"http://tinyurl.com/gpbnlf6",
    "description":"Female Pegasus pony in My Little Pony Friendship is Magic."},
  { "name":"Applejack", "image":"http://tinyurl.com/gkur8a6",
    "description":"Female Earth pony in My Little Pony Friendship is Magic."},
  { "name":"Twilight Sparkle","image":"http://tinyurl.com/hj877vs",
    "description":"Primary character of My Little Pony Friendship is Magic."}   ]
```


06/map_reduce.js, 06/map_reduce.html

Composing Functions

- Composability: Nested function invocation

```
var result = func1( funct2( func3() ) );
```

- Combining function composition with functions as objects provides a way to write programs in a way that does not require tangling logic into loops

http://eloquentjavascript.net/code/ancestry.js

Example: Ancestry Analysis

- Marijn Haverbeke provided family tree as JavaScript dataset
- Keys include: `name, sex, born, died, father, mother`

```
var ANCESTRY_FILE = [
  {name: "Carolus Haverbeke",        sex: "m", born: 1832, died: 1905, father: "Carel Haverbeke",              mother: "Maria van Brussel"},
  {name: "Emma de Milliano",         sex: "f", born: 1876, died: 1956, father: "Petrus de Milliano",           mother: "Sophia van Damme"},
  {name: "Maria de Rycke",           sex: "f", born: 1683, died: 1724, father: "Frederik de Rycke",            mother: "Laurentia van Vlaenderen"},
  {name: "Jan van Brussel",          sex: "m", born: 1714, died: 1748, father: "Jacobus van Brussel",          mother: "Joanna van Rooten"},
  {name: "Philibert Haverbeke",      sex: "m", born: 1907, died: 1997, father: "Emile Haverbeke",              mother: "Emma de Milliano"},
  {name: "Jan Frans van Brussel",    sex: "m", born: 1761, died: 1833, father: "Jacobus Bernardus van Brussel", mother: null},
  {name: "Pauwels van Haverbeke",    sex: "m", born: 1535, died: 1582, father: "N. van Haverbeke",             mother: null},
  {name: "Clara Aernoudts",          sex: "f", born: 1918, died: 2012, father: "Henry Aernoudts",              mother: "Sidonie Coene"},
  {name: "Emile Haverbeke",          sex: "m", born: 1877, died: 1968, father: "Carolus Haverbeke",            mother: "Maria Sturm"},
  {name: "Lieven de Causmaecker",    sex: "m", born: 1696, died: 1724, father: "Carel de Causmaecker",         mother: "Joanna Claes"},
  {name: "Pieter Haverbeke",         sex: "m", born: 1602, died: 1642, father: "Lieven van Haverbeke",         mother: null},
  {name: "Livina Haverbeke",         sex: "f", born: 1692, died: 1743, father: "Daniel Haverbeke",             mother: "Joanna de Pape"},
  {name: "Pieter Bernard Haverbeke", sex: "m", born: 1695, died: 1762, father: "Willem Haverbeke",             mother: "Petronella Wauters"},
  {name: "Lieven van Haverbeke",     sex: "m", born: 1570, died: 1636, father: "Pauwels van Haverbeke",        mother: "Lievijne Jans"},
  {name: "Joanna de Causmaecker",    sex: "f", born: 1762, died: 1807, father: "Bernardus de Causmaecker",     mother: null},
  {name: "Willem Haverbeke",         sex: "m", born: 1668, died: 1731, father: "Lieven Haverbeke",             mother: "Elisabeth Hercke"},
  {name: "Pieter Antone Haverbeke",  sex: "m", born: 1753, died: 1798, father: "Jan Francies Haverbeke",       mother: "Petronella de Decker"},
  {name: "Maria van Brussel",        sex: "f", born: 1801, died: 1834, father: "Jan Frans van Brussel",        mother: "Joanna de Causmaecker"},
  {name: "Angela Haverbeke",         sex: "f", born: 1728, died: 1734, father: "Pieter Bernard Haverbeke",     mother: "Livina de Vrieze"},
  {name: "Elisabeth Haverbeke",      sex: "f", born: 1711, died: 1754, father: "Jan Haverbeke",                mother: "Maria de Rycke"},
  {name: "Lievijne Jans",            sex: "f", born: 1542, died: 1582, father: null,                           mother: null},
  {name: "Bernardus de Causmaecker", sex: "m", born: 1721, died: 1789, father: "Lieven de Causmaecker",        mother: "Livina Haverbeke"},
  {name: "Jacoba Lammens",           sex: "f", born: 1699, died: 1740, father: "Lieven Lammens",               mother: "Livina de Vrieze"},
  {name: "Pieter de Decker",         sex: "m", born: 1705, died: 1780, father: "Joos de Decker",               mother: "Petronella van de Steene"},
  {name: "Joanna de Pape",           sex: "f", born: 1654, died: 1723, father: "Vincent de Pape",              mother: "Petronella Wauters"},
  {name: "Daniel Haverbeke",         sex: "m", born: 1652, died: 1723, father: "Lieven Haverbeke",             mother: "Elisabeth Hercke"},
  {name: "Lieven Haverbeke",         sex: "m", born: 1631, died: 1676, father: "Pieter Haverbeke",             mother: "Anna van Hecke"},
  {name: "Martina de Pape",          sex: "f", born: 1666, died: 1727, father: "Vincent de Pape",              mother: "Petronella Wauters"},
  {name: "Jan Francies Haverbeke",   sex: "m", born: 1725, died: 1779, father: "Pieter Bernard Haverbeke",     mother: "Livina de Vrieze"},
  {name: "Maria Haverbeke",          sex: "m", born: 1905, died: 1997, father: "Emile Haverbeke",              mother: "Emma de Milliano"},
  {name: "Petronella de Decker",     sex: "f", born: 1731, died: 1781, father: "Pieter de Decker",             mother: "Livina Haverbeke"},
  {name: "Livina Sierens",           sex: "f", born: 1761, died: 1826, father: "Jan Sierens",                  mother: "Maria van Waes"},
  {name: "Laurentia Haverbeke",      sex: "f", born: 1710, died: 1786, father: "Jan Haverbeke",                mother: "Maria de Rycke"},
  {name: "Carel Haverbeke",          sex: "m", born: 1796, died: 1837, father: "Pieter Antone Haverbeke",      mother: "Livina Sierens"},
  {name: "Elisabeth Hercke",         sex: "f", born: 1632, died: 1674, father: "Willem Hercke",                mother: "Margriet de Brabander"},
  {name: "Jan Haverbeke",            sex: "m", born: 1671, died: 1731, father: "Lieven Haverbeke",             mother: "Elisabeth Hercke"},
  {name: "Anna van Hecke",           sex: "f", born: 1607, died: 1670, father: "Paschasius van Hecke",         mother: "Martijntken Beelaert"},
  {name: "Maria Sturm",              sex: "f", born: 1835, died: 1917, father: "Charles Sturm",                mother: "Seraphina Spelier"},
  {name: "Jacobus Bernardus van Brussel", sex: "m", born: 1736, died: 1809, father: "Jan van Brussel",         mother: "Elisabeth Haverbeke"}
];
```

Example: Ancestry Analysis

Problem: Compute average age of all men

- Use function composition
- Start with functions that compute basic values and perform basic tests
- Declarative programming style

```
// Add two numbers
function plus(a, b) { return a + b; };

// Compute the average of an array
function ave(arr)    {
  return arr.reduce(plus) / arr.length;
}

// Compute age
function age(p)      { return p.died - p.born; }

// Test for a male
function male(p)     { return p.sex == "m"; }

// Test for a female
function female(p)   { return p.sex == "f"; }
```

Example: Ancestry Analysis

Computing average age with function composition

```
var ave_male_age = ave(  ancestry.filter( male ).map( age )  );
```

1) Filter file returning only males

2) Map males over age() returning all male ages

3) Reduce all ages to an average

Example: Ancestry Analysis

```
// ave_age.js
(function() {

  // Add two numbers
  function plus(a, b) { return a + b; };

  // Compute the average of an array
  function ave(arr)    {
    return arr.reduce(plus) / arr.length;
  }

  // Compute age
  function age(p)      { return p.died - p.born; }

  // Test for a male
  function male(p)     { return p.sex == "m"; }

  // Test for a female
  function female(p)   { return p.sex == "f"; }

  // Compute average age of male and female members
  var ave_male_age   = ave( ancestry.filter( male ).map( age ));
  var ave_female_age = ave( ancestry.filter( female ).map( age ));

  console.log('Average male age: ' + ave_male_age);
  console.log('Average female age: ' + ave_female_age);

})();
```

```
average male age: 61.666666666666664
average female age: 54.55555555555556
```

06/ave_age.js, 06/ave_age.html