Functional Programming with **JavaScript**

Mark Shelton | 10 November 2017

Prerequisites

Install node.js

nodejs.org/en/download

Install git client

git-scm.com/downloads

Setup

```
git clone https://github.com/fp-uwa/getting-started-javascript.git
cd getting-started-javascript
npm install
npm install -g npx
```

This Seminar

Intro to JavaScript

Developments in JavaScript

Array Functions

Currying & Closures

Useful Resources & Next Steps

Intro to JavaScript

High-level & interpreted

Dynamic & weakly-typed

Object-oriented

Functional

- Functions are objects

History of JavaScript

First appeared 1995

Big developments recently:

- Node.js in 2009
- ES5.1 in 2011
- ES6 in 2015

What is ES5.1?

Introduced Array Functions:

- map
- filter
- reduce
- etc.

What is ES6 / ES2015?

Introduced heaps more:

- Iterators
- Generators
- Arrow Functions
- Promises etc.

Fat Arrow Functions

```
var multiply = function(x, y) {
    return x * y;
}

const multiply = (x, y) => x * y;
// Concise & helps with closures
```

But...

Browser support for ES6 is still incomplete

So we use a tool called Babel to transpile our **ES6+** code (now at **ES8**) to **ES5** code

Examples

pluck – map

unique – reduce

match – filter

batsmen – map, filter, sort

make_tree – filter, forEach

map

```
const new_array = array.map(
    function(el, index, array) {
        return new_element;
    }
);
```

Example: pluck

Using **map** write a function that accepts an array and a property and returns an array containing that property from each object.

```
open ./problems/pluck.js
npx babel-node problems/pluck.js
```

reduce

```
const result = array.reduce(
    function(acc, el, index, array) {
        return new_acc;
    }, initial_value
);
```

Example: unique

Using **reduce** write a function that accepts an array and returns an array with all duplicates removed.

```
open ./problems/unique.js
npx babel-node problems/unique.js
```

filter

```
const sub_array = array.filter(
    function(el, index, array) {
        return test_bool;
    }
);
```

Example: match

Using **filter** write a function that accepts An array of objects and a property (key-value pair) and returns an array of all objects that match the given property.

open ./problems/match.js

npx babel-node problems/match.js

Currying

```
const hasValue =
    key => value => object =>
        object[key] === value;

ladders.filter(hasValue("height")(25));
```

Composition

```
const hasValue =
    key => value => object =>
        object[key] === value;

const hasHeight = hasValue("height");
result = ladders.filter(hasHeight(25));
```

Closures

```
const hasValue = function(key) {
    return function(value) {
        return function(object) {
            return object[key] === value;
}
```

Example: batsmen

Read in batsmen from a file and convert them into a list of objects with initials, surnames, runs, and averages.

Round averages to the nearest integer, sort batsmen in desc order by total runs and filter for surnames that start with C.

```
open ./problems/batsmen.js
npx babel-node problems/batsmen.js
```

Example: make_tree

Using array functions & recursion write a function that accepts a list of objects representing a hierarchy and builds it into a tree. **filter** & **forEach** may be helpful.

```
open ./problems/batsmen.js
npx babel-node problems/batsmen.js
```

Useful resources

Fun Fun Function:

www.youtube.com/watch?v=BMUiFMZr7vk&t

ES6 Javascript Guide:

www.rallycoding.com/courses/es6-javascript-the-complete-developers-guide/

Next steps

Functions: Ramda, Underscore, Lodash

Typing: TypeScript, Flow, Reason

Views: React Higher Order Components (HOC)

Store: Redux, ImmutableJS

Actions: Redux Observable, Redux Saga