Reducing Incidental Complexity with Functional Programming

Michael Erwin @metamje

"There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity [in software engineering]."

Agenda

1. What is Functional Programming?
2. Actions vs Values
3. The Incredibly Powerful For
4. The Silver Bullet: Map-Filter-Reduce
5. Map
6. Filter
7. Reduce
8. Functional Programming in JavaScript
9. Functional Programming for the Web
10. Questions, Feedback, Insults, Roundtable on who you voted for yesterday

What is Functional Programming

- Pure functions
- Immutable data
- Higher-order functions
- Strong types (maybe)
- Only math geniuses allowed

Actions vs Values (1)

```
class Customer {
    name = "Donald";
    getName() { return name; }
    /* action          value */
}
```

Actions vs Values (2)

```
class Customer {
    name = "Donald";
    getName() { return this.name; }
    // action         value! value
}
```

Actions vs Values (3)

```
const donny = {name: "Donald"};
const getName = customer => customer.name;
      // action             value    value

const getProp = prop => obj => obj.prop;
const getName2 = getProp('name');

getName(donny); // 'Donald'
getName2(donny); // 'Donald'
```

Stop wondering what this `this` is

The Incredibly Powerful For (1)

```
const list = [..];

for (let i=0; i < list.length; i++) {
    // whatever the hell you feel like
}
```

The Incredibly Powerful For (2)

```
const list = ['1','2',..];

for (let i=0; i < list.length; i++) {
    list[i] = Number(list[i]);
}
```

Turn list of strings into numbers

The Incredibly Powerful For (3)

```
const list = [1,2,..];
const evens = [];

for (let i=0; i < list.length; i++) {
    if ( list[i] % 2 === 0 ) { evens.push(list[i]); }
}
```

Filter out odds

The Incredibly Powerful For (4)

```
const list = [1,2,..];
const sum = 0;

for (let i=0;, i < list.length; i++) {
    sum += list[i];
}
```

Add it up

The Silver Bullet: Map-Filter-Reduce

```
Array.prototype.map
Array.prototype.filter
Array.prototype.reduce
```

"The passions of man will not conform to the dictates of reason and justice without constraint." -- Alexander Hamilton

Map: Wrapped Value in, Wrapped Value out (1)

```
const list = ['1','2',..];
const nums = list.map(s => Number(s));
console.log(nums); // [1,2,..]
```

"90% of iterator operations are probably maps" -- Me

Map: Wrapped Value in, Wrapped Value out (2)

```
const list = ['1','2',..];
const nums = list.map(Number);
console.log(nums); // [1,2,..]
```

We don't need that lambda

Map: Wrapped Value in, Wrapped Value out (3)

```
const nums = ['1','2',..].map(Number);
console.log(nums); // [1,2,..]
```

Please stop filling your head with names you don't need

Filter: Keep Only What You Want (1)

```
const nums = [1,2,..];
const evens = nums.filter(n => n % 2 === 0);
console.log(evens); // [2,4,..];
```

Filter: Keep Only What You Want (2)

```
const nums = [1,2,..];
const isEven = n => n % 2 === 0;
const evens = nums.filter(isEven);
console.log(evens); // [2,4,..];
```

Point-free is a little harder but maybe more expressive

Filter: Keep Only What You Want (3)

```
const evens = [1,2,..].filter(n => n % 2 === 0);
console.log(evens); // [2,4,..];
```

Reminder: Please stop filling your head with names you don't need

Reduce: The Silverest Bullet (1)

```
const nums = [1,2,..,10];
const sum = nums.reduce((a, b) => a + b);
console.log(sum); // 55
```

Reduce: The Silverest Bullet (2)

```
const nums = [1,2,..,10];
const add = (a, b) => a + b;
const sum = nums.reduce(add);
console.log(sum); // 55
```

Reduce: The Silverest Bullet (3)

```
const sum = [1,2,..,10].reduce((a, b) => a + b);
console.log(sum); // 55
```

Reduce: The Silverest Bullet (4)

```
const sum = [].reduce((a, b) => a + b); // TypeError
```

```
const sum = [].reduce((a, b) => a + b, 0);
console.log(sum); // 0
```

Reduce: The Silverest Bullet (5)

```
const customers = [c1,c2,..];
console.log(c1); // {id: 1, name: "Donald"};
```

```
const cmap = {};
for (let i=0; i < customers.length; i++) {
    let customer = customers[i];
    cmap[customer.id] = customer;
}
```

```
const cmap = customers.reduce((m, c) => Object.assign(m, {[c.id]: c}), {});
```

Reduce: The Silverest Bullet (6)

```
const map = f => l => l.reduce((r, e) => r.concat(f(e)), []);

const filter = p => l => l.reduce(r, e) => p(e) ? r.concat(e) : r, []);
```

Map-Filter-Reduce: Bringing it all together

```
const nums = [1,2,..];

const count = customers.map(Number)
                       .filter(n => n % 2 == 0)
                       .reduce((a,b) => a + b, 0);
```

Functional Programming in JavaScript

- Easy to get started (especially if on > ES6)
  - First class functions
- Many built-in constructs
  - `Array.prototype.*`
  - `Object.entries`
- Many Libraries
  - Ramda
  - Sanctuary
  - Lodash
  - Immutable.js
- Gradual Typing
  - Typescript
  - Flow

“To make biological survival possible, Mind at Large has to be funnelled through the reducing valve of the brain and nervous system. What comes out at the other end is a measly trickle of the kind of consciousness which will help us to stay alive on the surface of this particular planet. To formulate and express the contents of this reduced awareness, man has invented and endlessly elaborated those symbol-systems and implicit philosophies which we call languages. Every individual is at once the beneficiary and the victim of the linguistic tradition into which he or she has been born -- the beneficiary inasmuch as language gives access to the accumulated records of other people's experience, the victim in so far as it confirms him in the belief that reduced awareness is the only awareness and as it be-devils his sense of reality, so that he is all too apt to take his concepts for data, his words for actual things.” -- Aldous Huxley

Functional Programming for the Web (1)

Typescript

- A "superset" of JavaScript
- Gradual typing
- Many typings available
- Microsoft

Clojurescript

- Dynamic language
- Lisp
- Much community support

Functional Programming for the Web (2)

ReasonML

- Ocaml with brackets
- Static types
- Ocaml ecosystem
- Facebook

Elm

- Haskell inspired
- Strongly-typed
- Language and Framework in one
- Front-end only
- Limited features

Functional Programming for the Web (3)

Purescript

- Dialect of Haskell
- Strongly-typed, pure, functional language
- The most FP of what's currently available (if you want types)
- Small community
- Used in production by several for-profit companies

Many Returns of the Day to You

Michael Erwin

Twitter: @metamje

Github: metame