Learning to embrace events Elm

We want speed!
When do we want it!
Yesterday!
What's coming up?

1. Who am I?

2. What was my problem?

3. How did I learn to solve that kind of problem?
   - Intro to FP.
   - Intro to Elm.

4. How can you solve that problem yourselves?
   - Some working code.

5. What did I do to solve my particular problem?
Who am I?

1. Engineer for nearly 10 years.
   - Electronics -> Web

2. Work for MarketInvoice.
   - We help SMEs with cash flow using their existing invoices.

3. Full disclosure.
   - I'm not a master and I'm not a contributor, I'm just a fan.
My list of front-end problems

1. Two way binding?
2. Where is that state again?
3. Wait what kind of an object are you?
4. `{}` + `[]` === `0`; `[]` + `{}` === `{}`; <----- wut?
5. Are views even HTML anymore? Should they be?
6. `Something.prototype.botheringMe = someFuncThatIsnt`
7. Why hasn't this MVC model moved on!
8. Angular and similar can be................ slow.
Learn how someone else is tackling their problems!

Where the magic happens ...

Your Comfort Zone

more-leadership.com/ More-Leadership Bernd Geropp
Functional Ideals
but it’s haaaaaard.
Type System

There tends to be 4 mains 'kinds' of type.

1. Alias.
   - Naming a kind of primitive.
2. Union.
   - Think of it as a label or tag.
3. Tuple.
   - It's kind of a bucket for things.
4. Record.
   - A set of named properties.

Plus the common useful primitives;
Int, Float, String, Array, Dict and (linked) List
The languages usually enforce immutability. Comparisons tend to be structural by default and data tends to be copied. Which is good for concurrency.

Also, therefore in Elm;

If something is **referentially** the same.

It must be **structurally** the same.

*This is how virtual-dom does its magic.*
The function always returns the same result when given the same set of arguments, there are no side effects (with some I/O caveats).

1. No side effects:
   - Means this is easily tested.
   - Also easy to 'reason' about.
2. There is no 'state', just functions.
3. Elm has a runtime that deals with all the pesky 'real world' side effects.
Partial Application

A function called with one argument will return the same function with one less arity.

This is hard to explain but easy to demonstrate... in javascript.

```javascript
// Multiplication.
function a(x) {
    return function b(y){
        return x * y;
    }
}

var willEqual4 = a(2)(2);
var multiplyBy2 = a(2);
var willEqual8 = multiplyBy2(4); // 8
```

All functions with arity more than one are usually curried this way by default, to nested sets of arity one functions.
What is ELM

1. Installed with NPM.
2. Transpiles to Javascript.
3. Runtime packaged with build files.
4. Functional.
5. Reactive.
6. Strongly typed, with generics.

7. ????

8. Profit.
Elm Architecture

Single direction for flow of information.

1. Event driven (signals send actions to an address) - ng.emit
2. Single source of data (foldp) - ng.service
3. Modular abstractions (filters for your 'address space') - ng.on
SHOW ME THE CODE!
Event -> Update -> Model -> View
Composition

Composing the DOM.
Subscriptions

For external input
Commands

For side effects
Tasks

For async work
All working together!
Elm Take Homes

2. No 'floating' state.
3. Change only when you need to.
4. Rich type system.
Doing that but, like, with angular?

1. Event driven - ng.emit
2. Single source of data - ng.service
3. Reactive - ng.on / $watch
4. Use Typescript.

... Or of course React with Redux!
TRY IT
YOU MIGHT LIKE IT.
Thanks

Tech@MI: tech.marketinvoice.com
Tweet: @jasond_s
Email: jason@jasonds.co.uk

Get started with elm.

Elm Lang for all your Elm needs.

Doing in with javascript reactjs/redux was based on the Elm runtime, but works in JS.