Functions, combinators, and decorators in JavaScript

Seth House <seth@eseth.com>

Utah JS

2013-02-19

Outline


JavaScript
Allongé

Reginald Braithwaite

- https://github.com/raganwald
- http://allong.es/

Outline

The creation of small functions that compose with each other.

JavaScript Allongé in a nutshell

Learn when it's appropriate to write:

```
return splat(maybe(get('name')))(customerList);
```

Instead of:

```
return customerList.map(function (customer) {
    if (customer) {
        return customer.name
    }
});
```

As Little As Possible About Functions, But No Less

- Identities
- Applying
- Returns
- Call by value, call by reference
- Closures, scope, & environments

Let

What if you didn't have variable assignment?:

```
function (diameter) {
    return diameter * Pi
}
```

Let

Wrap within a function that takes an argument with the name you want:

```
(function (Pi) {
    return function (diameter) {
        return diameter * Pi
    }
})(3.14159265)
```

Let

"Let" is using an IIFE binds values to names:

```
(function (Pi) {
    return function (diameter) {
        return diameter * Pi
    }
})(3.14159265)(2)
//=> 6.2831853
```

Pure functions

- No side-effects
- It operates on its input and returns output
- No effect on other objects or states
- Contains no free variables

Outline

Combinators Higher-order pure functions that take only functions as arguments and return a function.

Why composition?

Chaining two or more functions together:

```
function cookAndEat (food) {
    return eat(cook(food))
}
```

Generalized:

```
function compose (a, b) {
    return function (c) {
        return a(b(c))
    }
}

var cookAndEat = compose(eat, cook);
```

The for-loop

```
var fruit = [' orange ', ' apple ', ' pear '];

var result = [];

for (var i = 0; i < fruit.length; i++) {
    result.push(fruit[i].trim());
}
```

The map

```
var fruit = [' orange ', ' apple ', ' pear '];

var result = fruit.map(function(val) {
    return val.trim();
});
```

The composition

```
var fruit = [' orange ', ' apple ', ' pear '];

var result = fruit.map(globalize('trim'));
```

The “second argument” to array-extras

```
var fruit = [' orange ', ' apple ', ' pear '];

var result = fruit.map(
    Function.prototype.call, String.prototype.trim);
```

Composing for map

```
['1', '2', '3'].map(parseFloat);
//=> [1, 2, 3]

// HOWEVER:

['1', '2', '3'].map(parseInt);
//=> [ 1, NaN, NaN ]

['1', '2', '3'].map(applyLast(parseInt, 10));
```

Outline

One function per task; splitting a function in two; extracting sub-functions

Partial application

```
function xhr(method, path, data, headers) {
    ...
}

var post = applyLeft(xhr, 'POST');

var get = applyLeft(xhr, 'GET');
```

Partial application

```
var status = applyLeft(xhr, 'GET', '/status');

var getJSON = applyRight(xhr, {
              'Accept': 'application/json',
              'Content-Type': 'application/json'})
```

Maybe

```
function Model () {};

Model.prototype.setSomething = maybe(function(value) {
    this.something = value;
});
```

Cleaner callbacks with partial application

```
post('entry/create',
     postFormAndUpdate('formname', 'mydiv'));
```

Currying

Extract single-argument functions out of a multi-argument function.

```
add('sum', 5, 6)
```

Becomes:

```
addCurried('sum')(6)(5)
```

Currying

```
function converter(toUnit, factor, offset, input) {
    ...
}

var milesToKm = converter.curry(
    'km', 1.60936, undefined);
var poundsToKg = converter.curry(
    'kg', 0.45460, undefined);
var farenheitToCelsius = converter.curry(
    'degrees C', 0.5556, -32);

milesToKm(10);              //=> 16.09 km
poundsToKg(2.5);            //=> 1.14 kg
farenheitToCelsius(98);     //=> 36.67 degrees C
```

Currying

```
function logError(message, inDevmode) {
    if (inDevmode) console.error(message);
}

function makeLogger(inDevmode) {
    return function (err) {
        return logError(err.message || err.toString(),
            inDevmode);
    };
}

window.onerror = makeLogger(true);
```

Outline

Decorators

Decorators

Takes one function as an argument, returns another function, and the returned function is a variation of the argument function.

Example

```
function Todo (name) {
    ...
};

Todo.prototype.do = fluent(function () {
    this.done = true;
});

Todo.prototype.undo = fluent(function () {
    this.done = false;
});
```

Common decorators

AKA “advice”, AKA aspect oriented programming, AKA Lisp Flavors

- before
- after
- around
- provided

Common decorators

```
function SomeModel(name) {
    ...
};

SomeModel.prototype.delete = isAdmin(user, function ()
    this.delete();
});
```

Outline

How We Learned To Stop Worrying And Love JavaScript

@danwrong @angustweets

Angus Croll

"Some developers like rulebooks and boilerplate—which is why we have Java. The joy of JavaScript is rooted in its lack of rigity and the infinite possibilities that this allows for."

—Angus Croll

Functional mixins

- https://github.com/raganwald/method-combinators
- https://github.com/PuerkitoBio/advice
- https://github.com/twitter/flight/

Functional mixins

```
function() {
    function withDrama() {
        this.before('announce', function() {
            clearThroat();
        });

        this.after('leaving', function() {
            slamDoor();
        });
    }

    return withDrama;
}
```

Functional reactive programming (FRP)

- Bacon.js
- Demo:
  `http://raimohanska.github.com/bacon.js-slides/`