
A UNIFIED THEORY OF JAVASCRIPT STYLE,
PART I
JAVASCRIPT
COMBINATORS

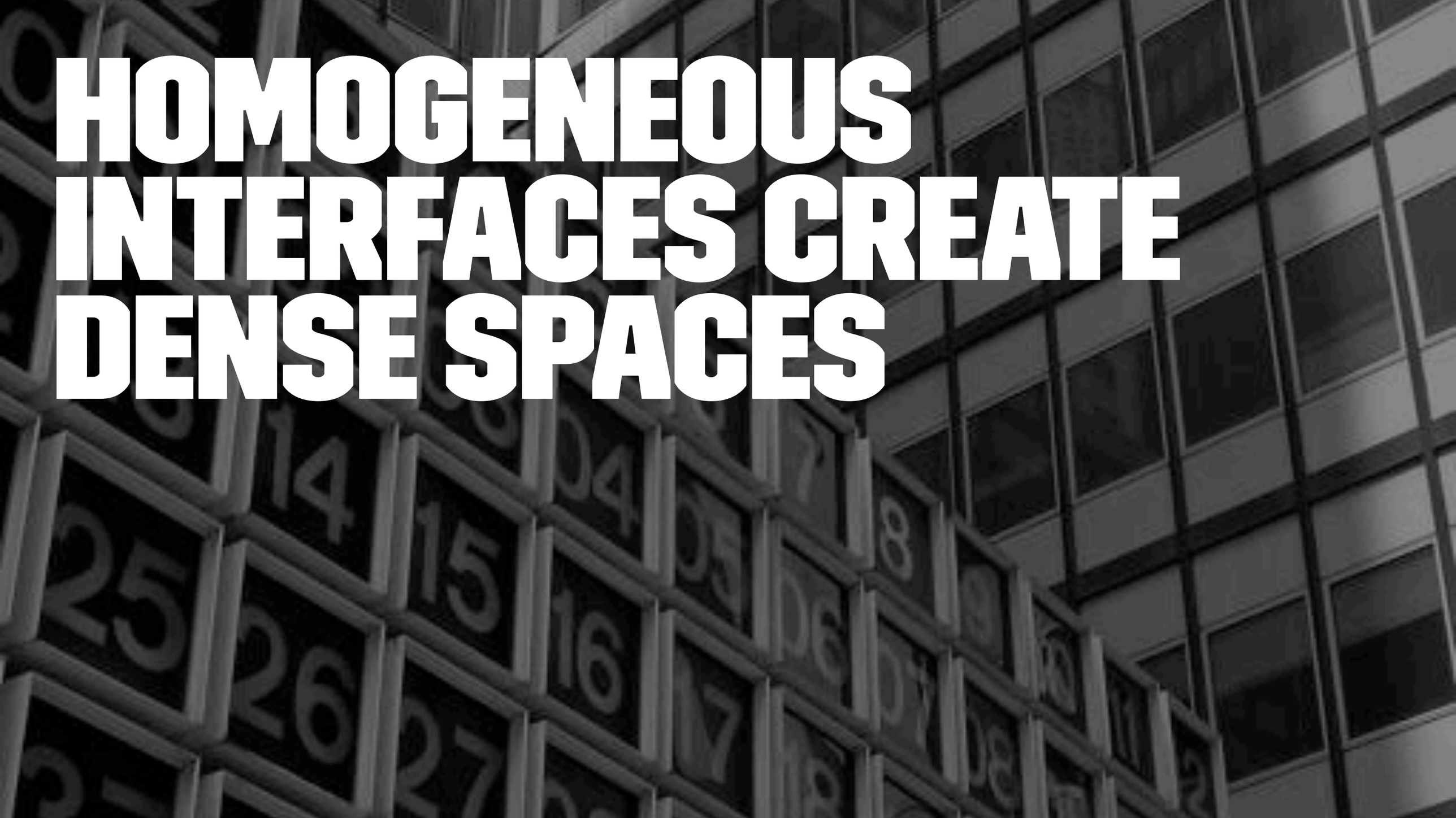

29

WE'LL TALK ABOUT COMBINATORS AND DECORATORS

BUT THINK ABOUT FLEXIBILITY AND DECLUTTERING


COMPOSITION
WE COMPOSE ENTITIES TO CREATE
NEW ENTITIES

INTERFACES
NOT ALL ENTITIES "FIT TOGETHER"

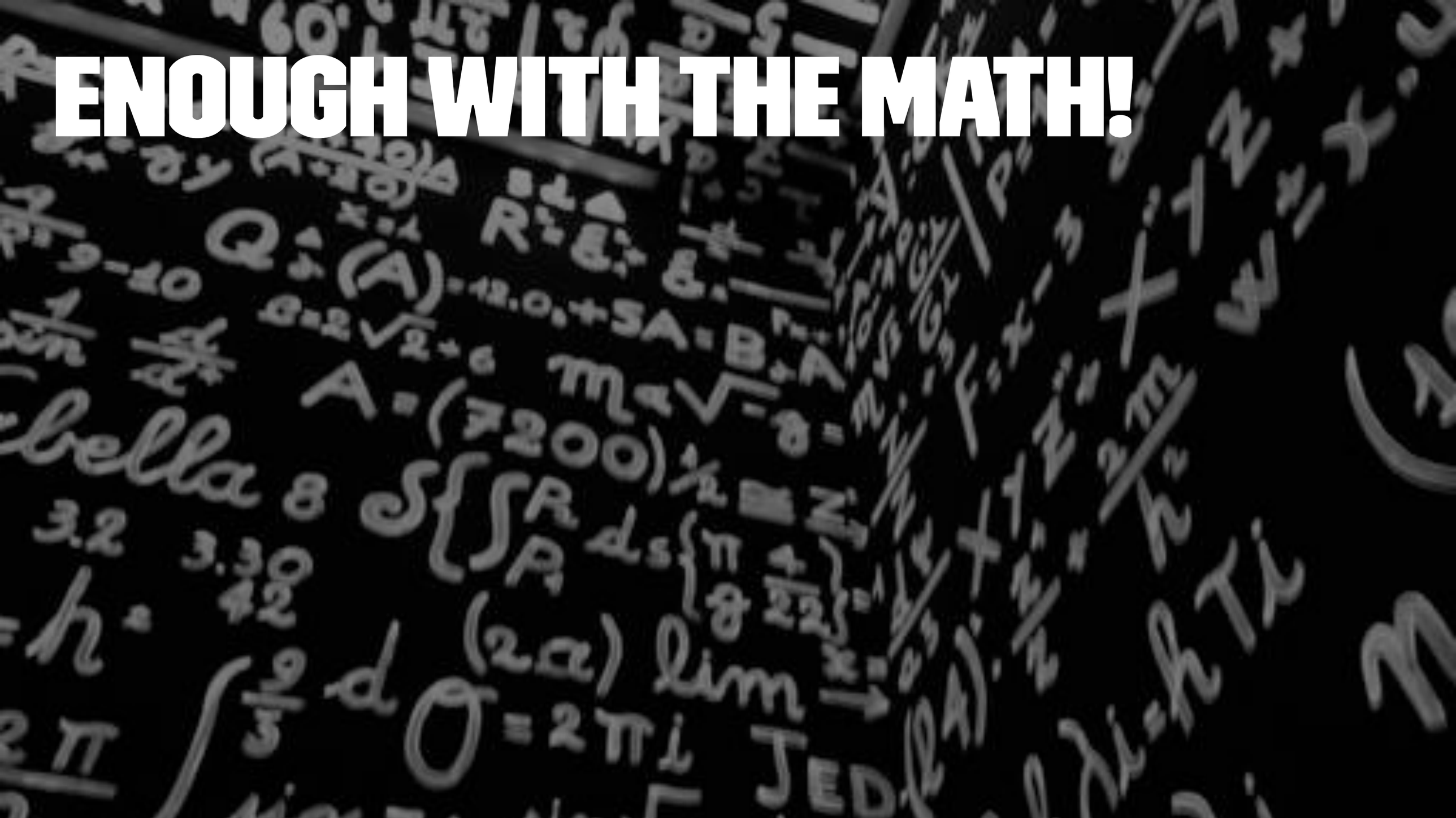

HOMOGENEOUS
INTERFACES CREATE
DENSE SPACES


HETEROGENEOUS
INTERFACES CREATE
SPARSE SPACES


DENSE IS MORE
FLEXIBLE THAN SPARSE



SPARSE CAN BE
QUICKER TO GRASP


ENOUGH WITH THE MATH!

PLUCK: "A CONVENIENT VERSION OF WHAT IS PERHAPS THE MOST COMMON USE-CASE FOR MAP: EXTRACTING A LIST OF PROPERTY VALUES."

"PLUCKWITH" IS THE FLIPPED FORM OF "PLUCK"

```
function pluck (mappable, key) {
  return mappable.map(function (obj) {
        return obj[key];
    });
};

function pluckWith (key, mappable) {
  return pluck(mappable, key);
};

var stooges = [
    {name: 'moe', age: 40},
    {name: 'larry', age: 50},
    {name: 'curly', age: 60}];

pluckWith('name', stooges);
    //=> ["moe", "larry", "curly"]
```

LET'S MAKE "PLUCKWITH" OUT OF COMBINATORS

A UNARY COMBINATOR

```
function flip (fn) {
    return function flipped (a, b) {
        return fn.call(this, b, a);
    }
}

function arrow (a, b) {
    return "" + a + " -> " + b;
}

flip(arrow)("x", "y")
    //=> 'y -> x'
```


CURRY
ANOTHER UNARY
COMBINATOR

```
function curry (fn) {
    return function curried (a, optionalB) {
        if (arguments.length > 1) {
            return fn.call(this, a, optionalB);
        }
        else return function partiallyApplied (b) {
            return fn.call(this, a, b);
        }
    }
}
```

CURRYING:

```
var curriedArrow = curry(arrow);
    //=> [function]

curriedArrow('finger')('moon')
    //=> 'finger -> moon'
```

PARTIAL APPLICATION:

```
var taoism = curry(arrow)('finger');
    //=> [function]

taoism('moon')
    //=> 'finger -> moon'
```


NOTA BENE
PARTIAL APPLICATION TRANSFORMS BINARY OPERATIONS INTO UNARY OPERATIONS

```
function get (object, property) {
    return object[property];
}

get({foo: 1}, 'foo')
    //=> 1
```

```
var getWith = curry(flip(get));

getWith('foo')({foo: 1})
    //=> 1
```

```
function map (mappable, fn) {
    return mappable.map(fn, this);
}

function double (n) {
    return n * 2;
}

map([1, 2, 3], double)
    //=> [2, 4, 6]
```

```
var mapWith = curry(flip(map));

mapWith(double, [1, 2, 3]);
    //=> [2, 4, 6]

var doubleAll = mapWith(double);

doubleAll([1, 2, 3])
    //=> [2, 4, 6]
```

“ALMOST THERE...”

```
function pluckWith (attr) {
  return mapWith(getWith(attr));
}
```

COMPOSE

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

QUOD ERAT DEMONSTRANDUM THE COMBINATOR IMPLEMENTATION OF "PLUCKWITH"

```
var pluckWith = compose(mapWith, getWith);
```

LET'S COMPARE BOTH IMPLEMENTATIONS OF "PLUCKWITH"

```
var pluckWith = compose(mapWith, getWith);

//// versus ////

function pluck (mappable, key) {
  return mappable.map(function (obj) {
        return obj[key];
    });
};

function pluckWith (key, mappable) {
  return pluck(mappable, key);
};
```

LESSON
COMPOSING FUNCTIONS WITH COMBINATORS INCREASES CODE FLEXIBILITY...

LESSON
COMPOSING FUNCTIONS WITH COMBINATORS DEMANDS INCREASED MENTAL FLEXIBILITY

USING COMBINATORS TO MAKE DECORATORS

```
function Cake () {}

extend(Cake.prototype, {
  mix: function () {
    // mix ingredients together
        return this;
  },
  rise: function (duration) {
    // let the ingredients rise
        return this;
  },
  bake: function () {
    // do some baking
        return this;
  }
});
```

FLUENT

```
function fluent (methodBody) {
  return function fluentized () {
    methodBody.apply(this, arguments);
    return this;
  }
}
```

```
function Cake () {}

extend(Cake.prototype, {
  mix: fluent( function () {
    // mix ingredients together
  }),
  rise: fluent( function (duration) {
    // let the ingredients rise
  }),
  bake: fluent(function () {
    // do some baking
  })
});
```

NEW REQUIREMENTS
MIX BEFORE RISING OR BAKING

```
extend(Cake.prototype, {
  mix: fluent( function () {
    // mix ingredients together
  }),
  rise: fluent( function (duration) {
        this.mix();
    // let the ingredients rise
  }),
  bake: fluent(function () {
        this.mix();
    // do some baking
  })
});
```

BEFORE
A COMBINATOR THAT TRANSFORMS DECORATIONS INTO DECORATORS

```
var before = curry(
    function decorate (decoration, method) {
        return function decoratedWithBefore () {
        decoration.apply(this, arguments);
        return method.apply(this, arguments);
        };
  }
);

var mixFirst = before(function () {
  this.mix()
});
```

THE FINAL VERSION

```
extend(Cake.prototype, {

  // Other methods...

  mix: fluent( function () {
    // mix ingredients together
  }),
  rise: fluent( mixFirst( function (duration) {
    // let the ingredients rise
  }})),
  bake: fluent( mixFirst( function () {
    // do some baking
  }))
});
```

LESSON
DECORATORS DECLUTTER
SECONDARY CONCERNS

AFTER

```
var after = curry(
    function decorate (decoration, method) {
        return function decoratedWithAfter () {
            var returnValue = method.apply(this, arguments);
        decoration.apply(this, arguments);
        return returnValue;
        };
  }
);
```

AROUND

```
var around = curry(
    function decorate (decoration, method) {
        return function decoratedWithAround () {
            var methodPrepended = [method].concat(
                [].slice.call(arguments, 0)
            );

        return decoration.apply(this, methodPrepended);
        };
  }
);
```

CALL ME MAYBE

```
var maybe = around(function (fn, value) {
    if (value != null) {
        return fn.call(this, value);;
    }
});

maybe(double)(2)
    //=> 4

maybe(double)(null)
    //=> undefined
```

GENERALIZED GUARDS

```
function provided (guard) {
    return around(function () {
        var fn = arguments[0],
            values = [].slice.call(arguments, 1);

        if (guard.apply(this, values)) {
            return fn.apply(this, values);
        }
    });
}

var maybe = provided( function (value) {
    return value != null;
});
```

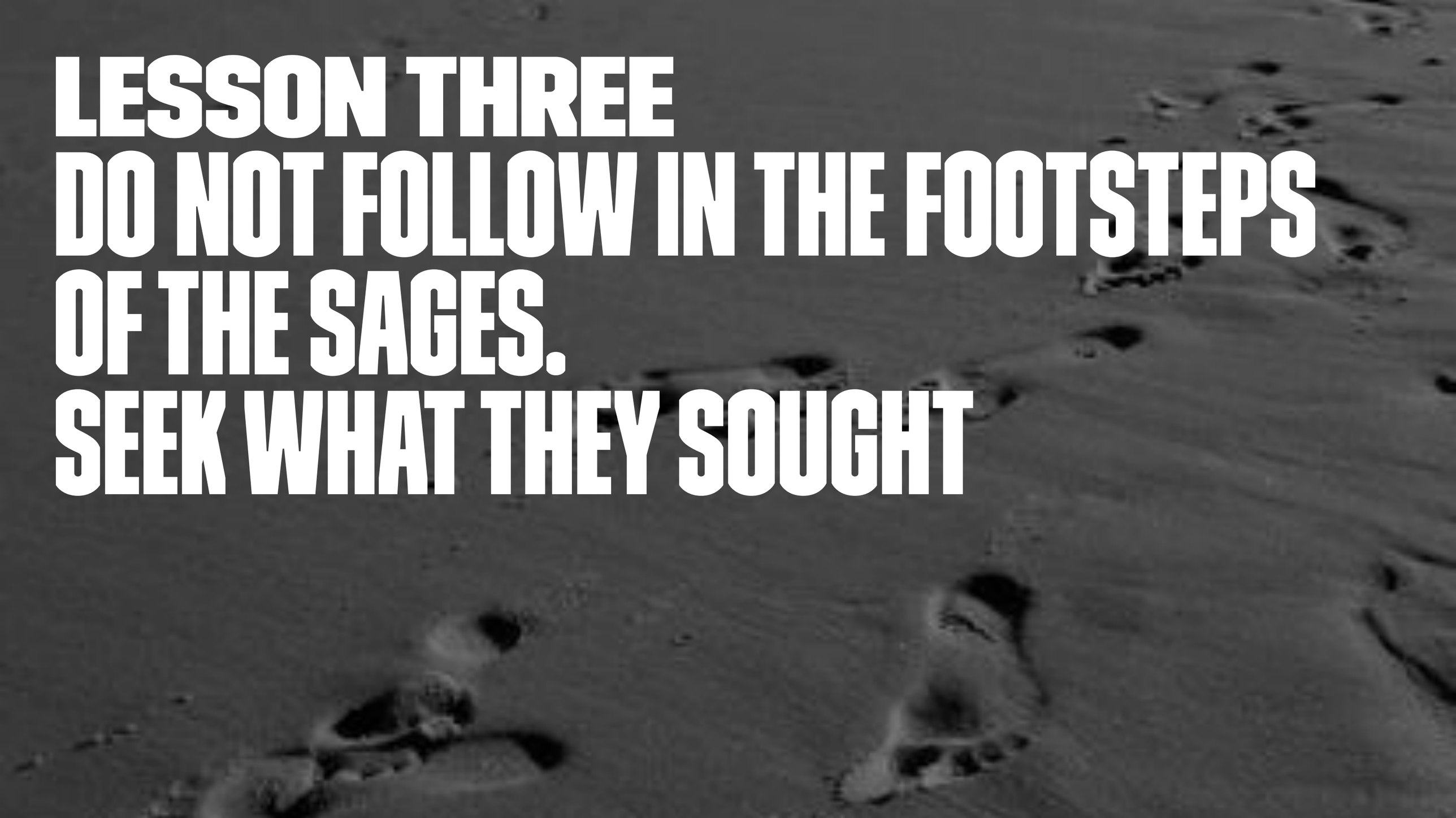
INVERSIONS

```
function not (fn) {
    return function notted () {
        return !fn.apply(this, arguments)
    }
}

var except = compose(provided, not);

var maybe = except( function (value) {
    return value == null;
});
```

LESSONS

LESSON ONE
COMBINATORS INCREASE CODE FLEXIBILITY AND REQUIRE INCREASED MENTAL FLEXIBILITY

LESSON TWO
DECORATORS DECLUTTER
SECONDARY CONCERNS


LESSON THREE
DO NOT FOLLOW IN THE FOOTSTEPS
OF THE SAGES.
SEEK WHAT THEY SOUGHT

REGINALD BRAITHWAITE

GITHUB, INC.

RAGANWALD.COM

@RAGANWALD

NDC Conference, Oslo, Norway,
June 5, 2014