Functional programming and Curry Cooking in JS

Stefanie Schirmer
@linse

Into the frying pan

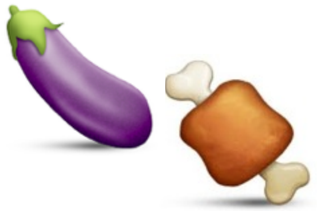
http://commons.wikimedia.org/wiki/File:Spices_for_Indonesian_food.jpg

what a mess

https://mpkdesign.files.wordpress.com/2010/02/img_4833.jpg

```
var spices = [ 'cinnamon',
  'cumin',
  'cilantro',
  'clove',
  'cardamom',
  'black pepper',
  'chili',
  'ginger or ginger paste',
  'Garam Masala',
  'turmeric' ];
```

```
var base = [ 'onions',
  'yoghurt',
  'coconut milk' ];
```



```
var ingredients = [ 'chicken',
  'lamb',
  'cauliflower und potato',
  'paneer',
  'aubergine' ];
```

```
var spices = [ 'cinnamon',
  'cumin',
  'cilan
  'clove
  'cardamom',
  'black pepper'
  'chili',
  'ginger or ginger paste',
  'Gara
  'turm

var base = [ 'onions',
  'yogh
  'coco

var ingredients = [ 'chicken',
  'lamb',
  'cauliflower
  'aubergine' ];
```

cinnamon

chili

cumin

cilantro

cardamom

clove

black pepper

garam masala

ginger

turmeric

coconut milk

yoghurt

onions

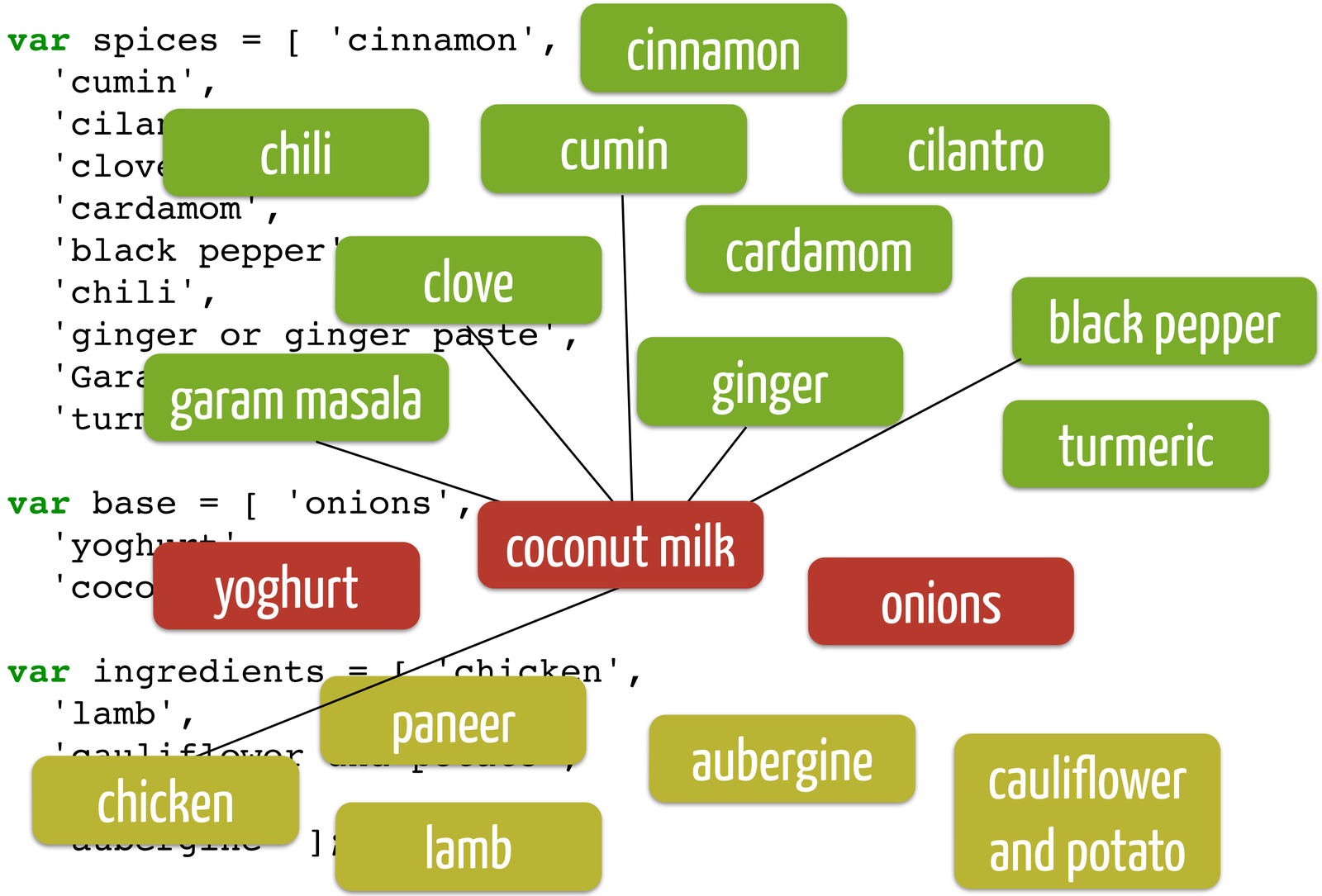
paneer

aubergine

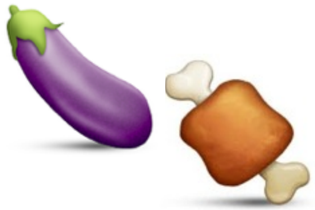
cauliflower and potato

chicken

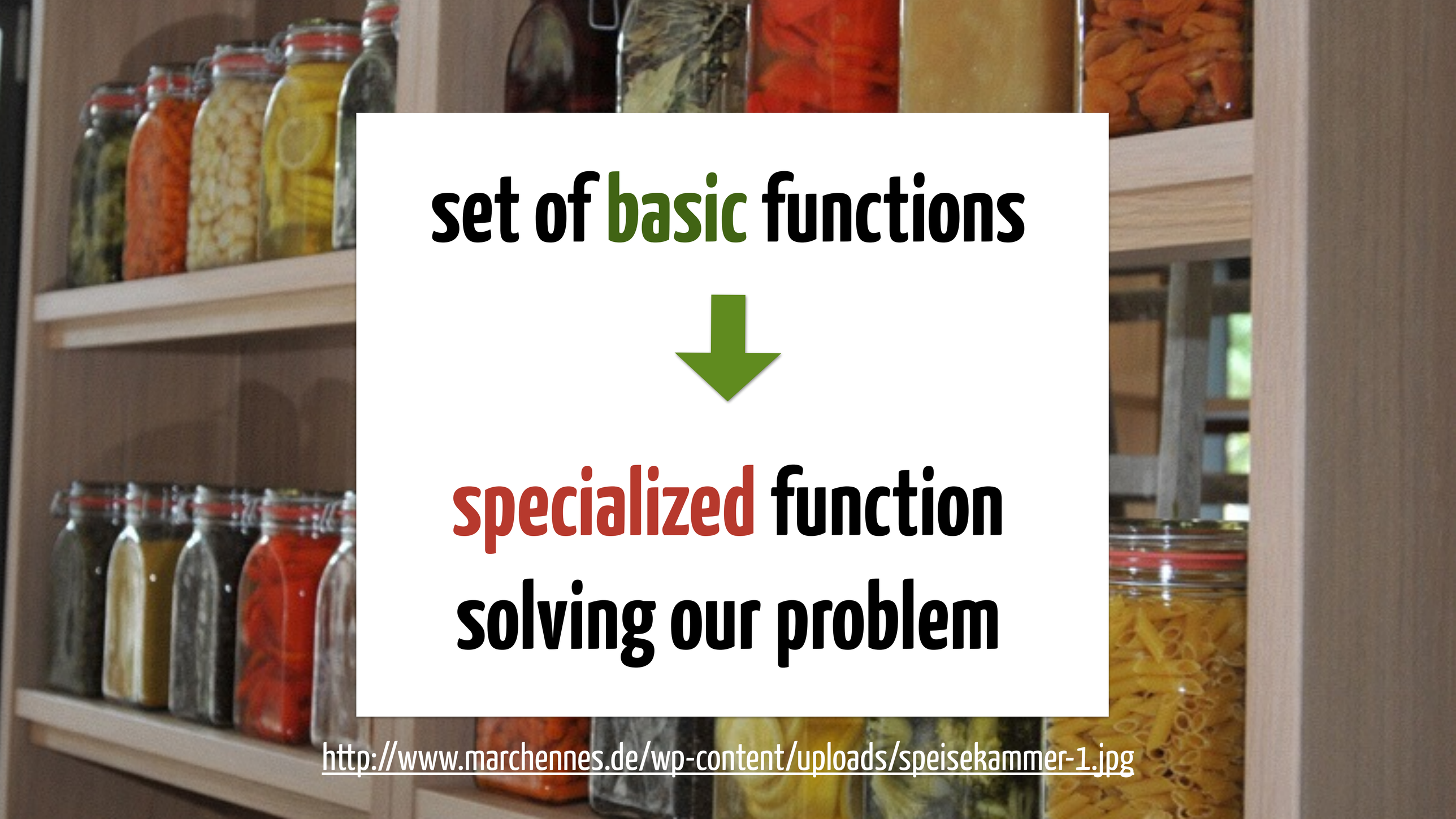
lamb


var spices = ['cinnamon',
'cumin',
'cilan
'clove
'cardamom',
'black pepper'
'chili',
'ginger or ginger paste',
'Gara
'turm
var base = ['onions',
'yogh
'coco
var ingredients = ['chicken',
'lamb',
'cauliflower
aubergine];
cinnamon
chili
cumin
cilantro
cardamom
clove
black pepper
garam masala
ginger
turmeric
coconut milk
yoghurt
onions
paneer
aubergine
cauliflower and potato
chicken
lamb

```
function shuffle(array) {
  // do the Fisher-Yates shuffle on the array
  // …
  return array;
}

function pick(n, array) {
  return shuffle(array).slice(0, n);
}

function printItem(i) {
  console.log('*', i);
}

// we can now:
// pick(5, spices).map(printItem);
```



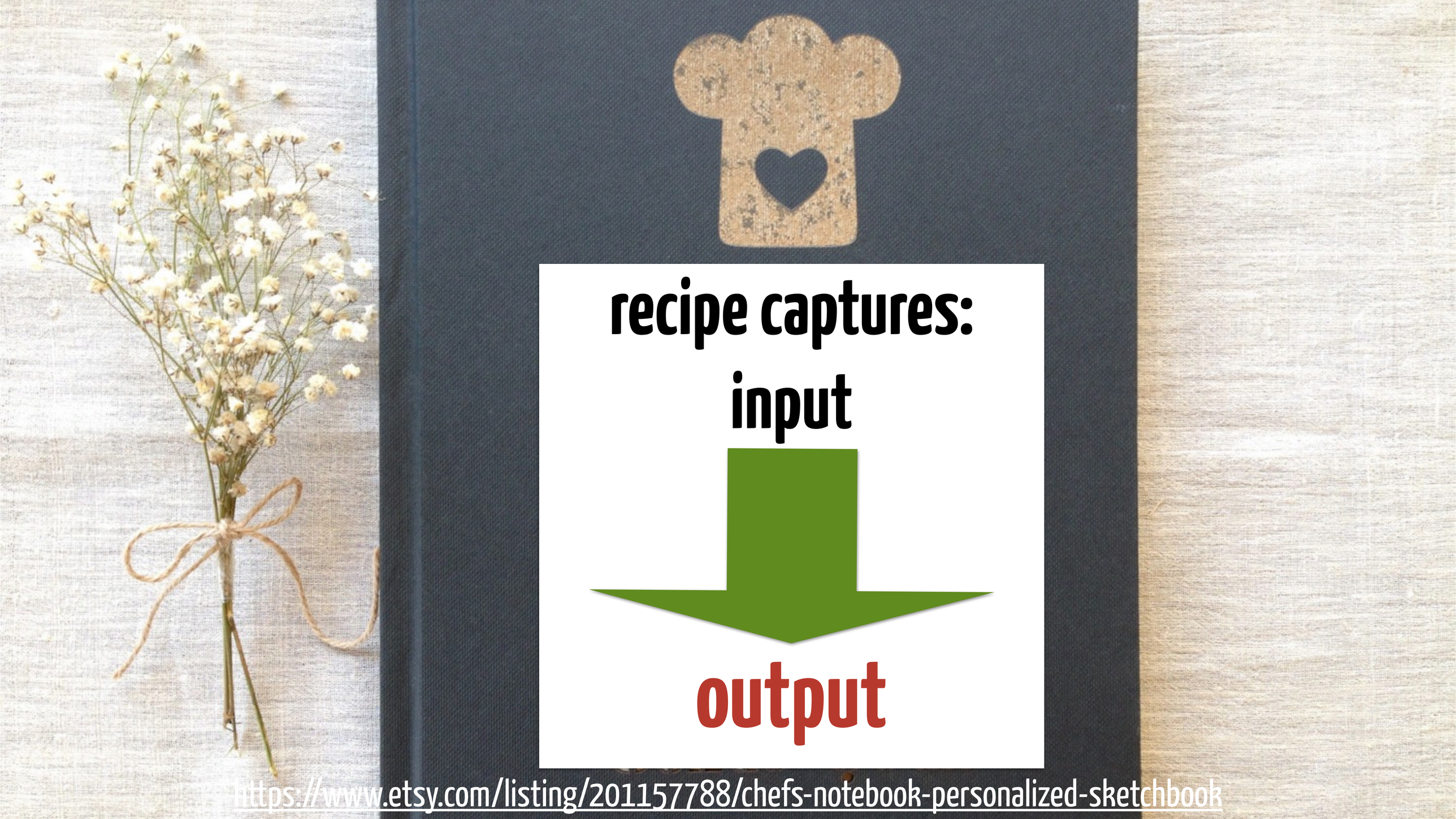
```
function printRecipe() {

  console.log('Curry will be based on:\n’);
  pick(1, base).map(printItem);

  console.log('Spices to roast:\n');
  pick(5, spices).map(printItem);

  console.log('Main ingredient:\n');
  pick(1, ingredients).map(printItem);

  console.log('Fry main ingredient on high heat,',
   'add spices and base and let it stew\n');
  console.log(‘Enjoy!');

}
```



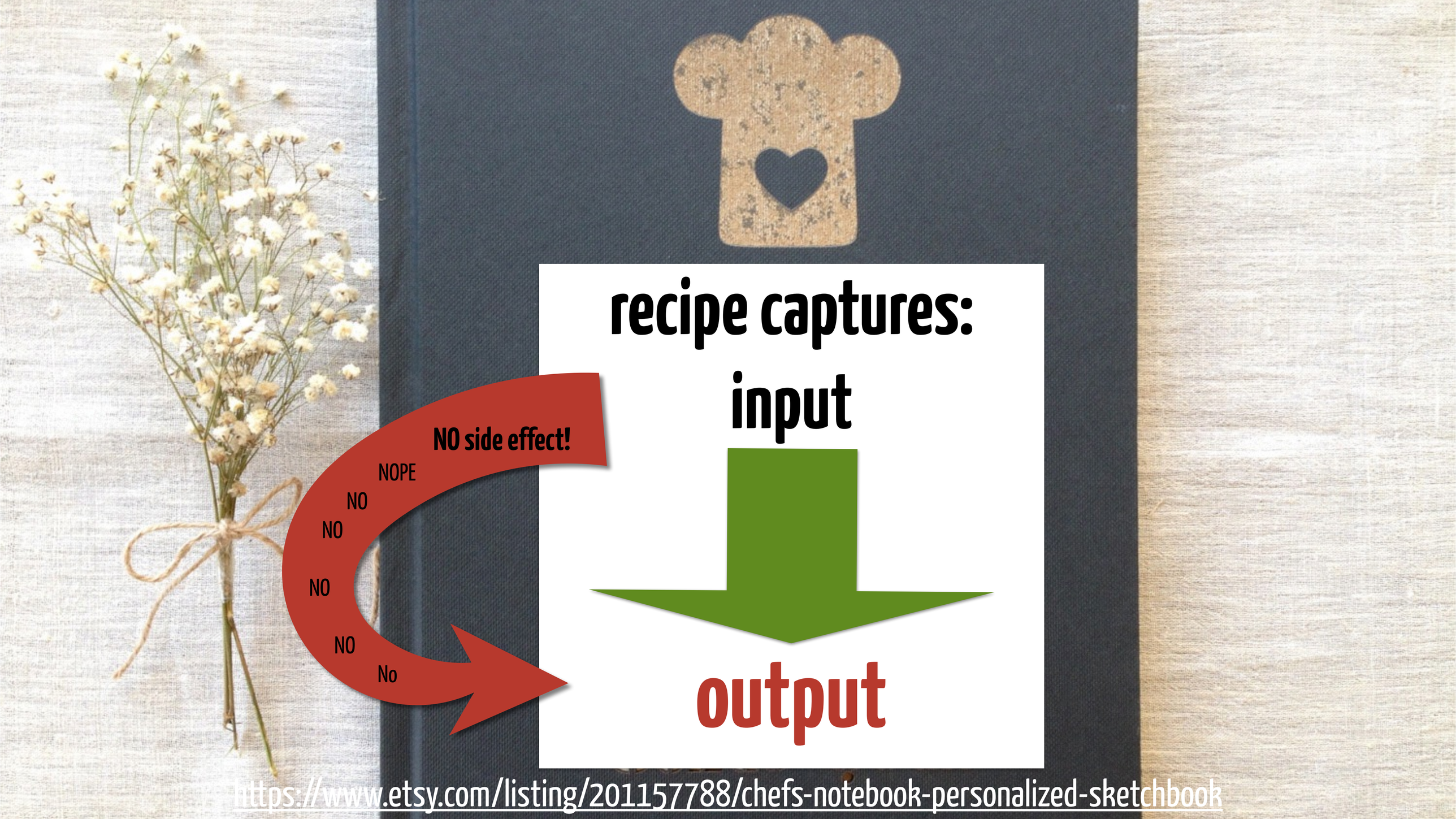
set of basic functions

specialized function solving our problem

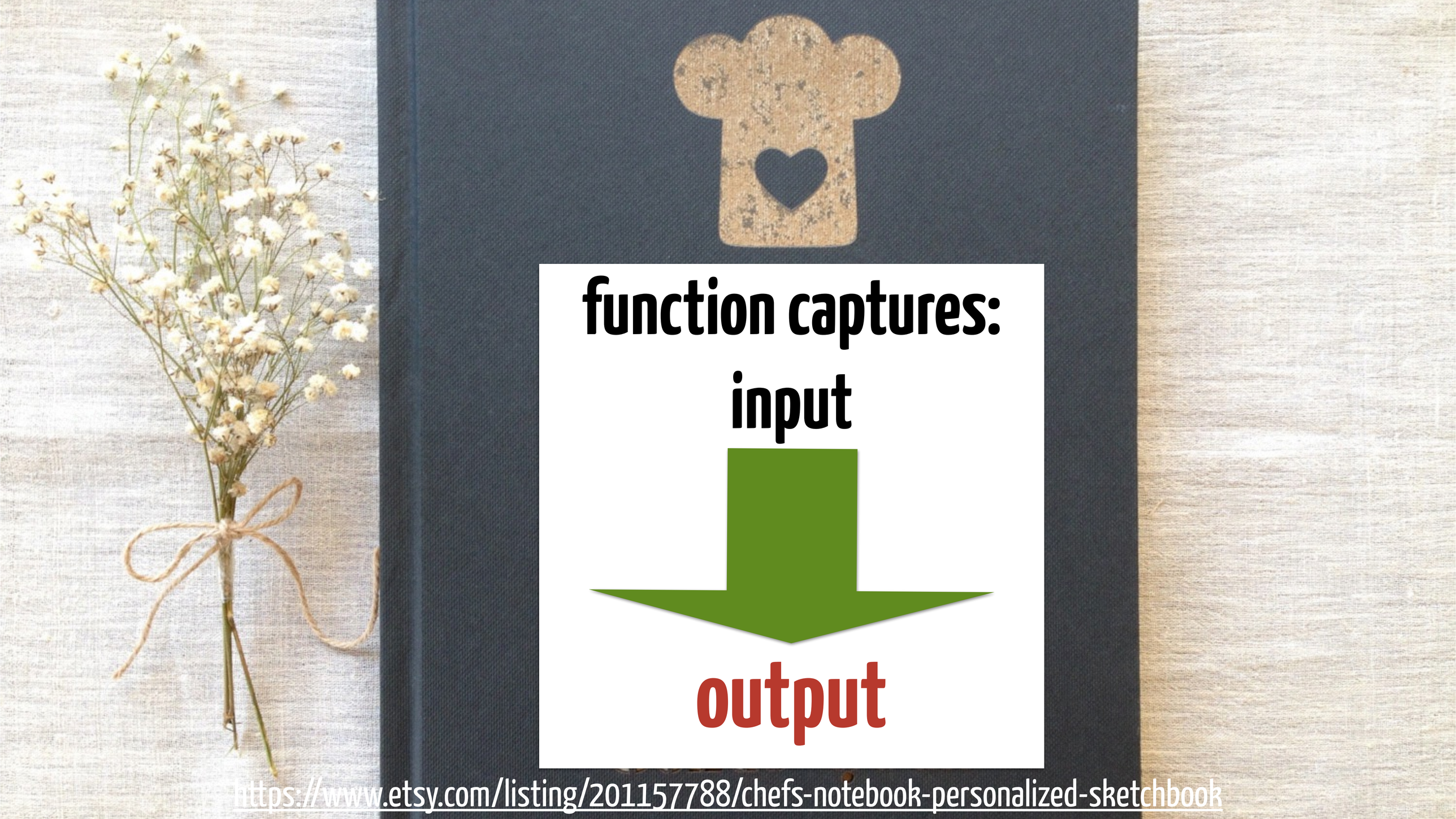
http://www.marchennes.de/wp-content/uploads/speisekammer-1.jpg



https://www.etsy.com/listing/201157788/chefs-notebook-personalized-sketchbook



https://www.etsy.com/listing/201157788/chefs-notebook-personalized-sketchbook



https://www.etsy.com/listing/201157788/chefs-notebook-personalized-sketchbook

I. Abstraction

"The art of programming is the art of organizing **complexity**, of mastering multitude and avoiding its bastard chaos

We all know that the only mental tool by means of which a very finite piece of reasoning can cover a myriad cases is called **abstraction**"

–E. W. Dijkstra

how can we use abstraction?

a variable abstracts from a value

```
var veggies = ['chickpea',
               'potato',
               'cauliflower'];
```

a function abstracts from a sequence

of statements

```
function roastSpices () {
  console.log('pick spices');
  console.log('use mortar');
  console.log('fry in pan');
}
```

a ???

abstracts from a function

a higher order function

abstracts from a function

```
spices.sort(sort_by('name'));
spices.sort(sort_by('color'));
spices.sort(sort_by('quantity’));
spices.sort(sort_by('compound’));
```

a module

provides a set of functions

levels of detail

Module

HigherOrderFunction

Function

Variable

building blocks

II. Functions

how is this

different?

Program

Module

HigherOrderFunction

Function

Block

Statement

Variable

Program

Module

HigherOrderFunction

Function

Block

Statement

Variable

Program

Module

HigherOrderFunction

Function

Block

Statement

Variable

isn't this

slow?

Closures

```
function localGreeting() {
  var captured = “Namaste!";
  return function() {
    console.log(captured);
  };
}
```

Curry

http://commons.wikimedia.org/wiki/File:Curry_Tree_Fruit.jpg

https://wiki.haskell.org/wikiupload/8/86/HaskellBCurry.jpg

1 function with

n arguments

curry

n functions with

1 argument

```
f = function (a,b,c) {
 return console.log(a,b,c);
}
```

```
> f('Haskell','B.','Curry')
Haskell B. Curry
```

```
fCurry = function(a) {
  return function(b) {
    return function(c) {
      return console.log(a,b,c);
    }
  }
}

> fCurry('Haskell')('B.')('Curry')
Haskell B. Curry
```

```
f = function (a,b,c) {
 return console.log(a,b,c);
}

> f('Haskell','B.','Curry')
Haskell B. Curry
```

```
fCurry = function(a) {
  return function(b) {
    return function(c) {
      return console.log(a,b,c);
    }
  }
}
```

```
> fCurry('Haskell')('B.')('Curry')
Haskell B. Curry
```

```
f = function (a,b,c) {
 return console.log(a,b,c);
}

> f('Haskell','B.','Curry')
Haskell B. Curry
```

```
fCurry = function(a) {
  return function(b) {
    return function(c) {
      return console.log(a,b,c);
    }
  }
}
```

```
> fCurry('Haskell')('B.')('Curry')
Haskell B. Curry
```

```
f = function (a,b,c) {
 return console.log(a,b,c);
}

> f('Haskell','B.','Curry')
Haskell B. Curry

fCurry = function(a) {
  return function(b) {
    return function(c) {
      return console.log(a,b,c);
    }
  }
}

> fCurry('Haskell')('B.')('Curry')
Haskell B. Curry
```

```
f = function (a,b,c) {
 return console.log(a,b,c);
}
```

```
> f('Haskell','B.','Curry')
Haskell B. Curry
```

```
fCurry = function(a) {
  return function(b) {
    return function(c) {
      return console.log(a,b,c);
    }
  }
}
```

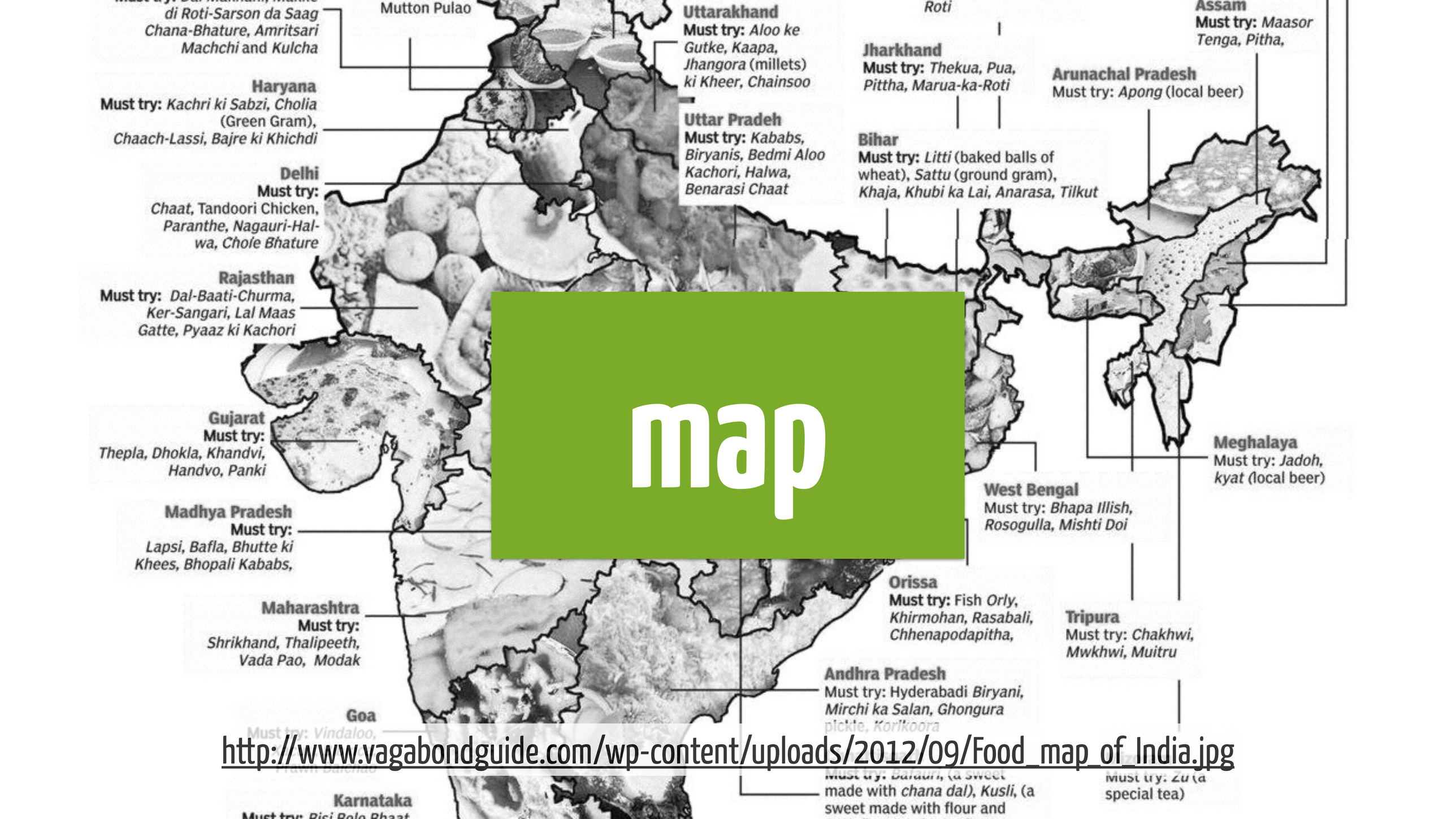
```
> fCurry('Haskell')('B.')('Curry')
Haskell B. Curry
```

"[..] well known wherever the Currys have lived has been the hospitality they have shown.

Cooking also played a role in the growth of interest in combinatory logic."

–J.P. Seldin, J.R. Hindley

Veggies,
map, reduce and filter



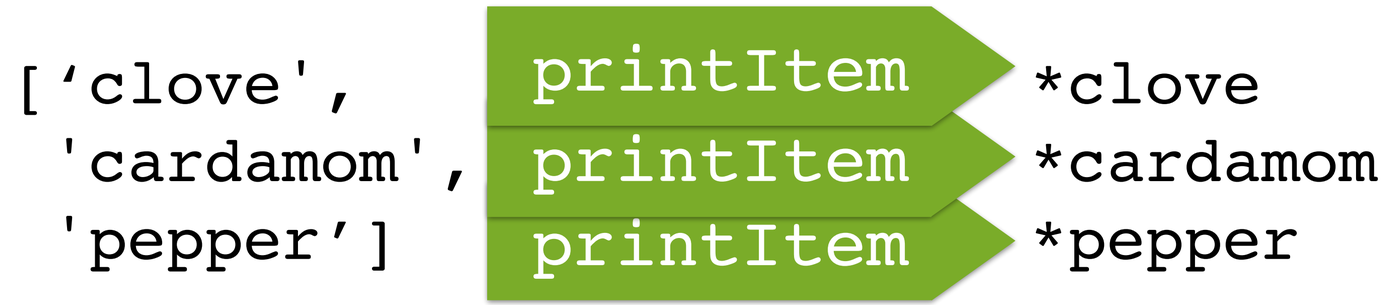
http://www.vagabondguide.com/wp-content/uploads/2012/09/Food_map_of_India.jpg

transform with map

```
console.log('Spices to roast:\n');
pick(5, spices).map(printItem);
```

transform with map

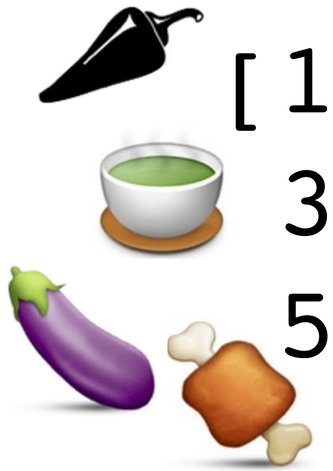
```
console.log('Spices to roast:\n');
pick(5, spices).map(printItem);
```



https://www.flickr.com/photos/notahipster/4250668672/

combine with reduce

```
function plus(a,b) { return a + b; }

console.log('Sum items in pantry:\n');
individualCounts.reduce(plus);
```



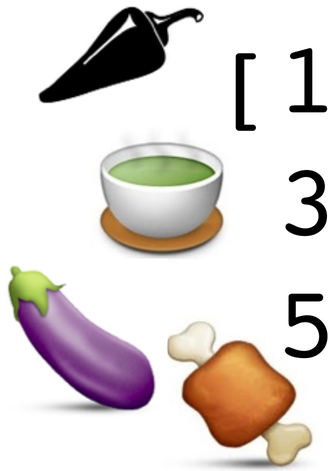
[10,
3,
5]

reduce

18

combine with reduce

```
function plus(a,b) { return a + b; }

console.log('Sum items in pantry:\n');
individualCounts.reduce(plus);
```



[10,
3,
5]

plus

13
5

plus

18

https://www.flickr.com/photos/dustinlinn3/52[illegible]28489/

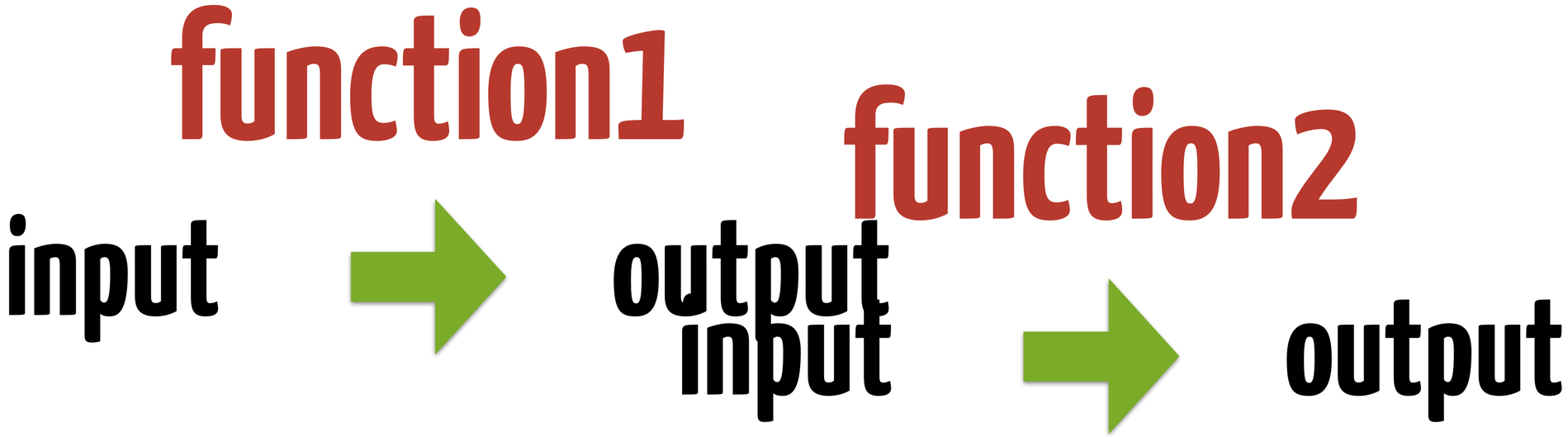
select with filter

```
function isInStock(item) {
  return item.quantity > 0;
}

console.log('What's in stock:\n');
veggies.filter(isInStock);
```

Composition

function

input  output


function1
input
output
function2
input
output

$$(g \circ f)(x) = g(f(x))$$

```
function compose(g, f) {
  return function (x) {
    return g(f(x));
  }
}
```

```
function count(x) {
  return x.quantity;
}

function chopInHalf(x) {
  return 2 * x;
}
```

```
> compose(chopInHalf,count)(okra)
50
```

Purely functional?

Point free style

curry + invisible arguments

```
mySort xs = sortBy compareSize xs
```

```
mySort    = sortBy compareSize
```

curry + invisible arguments

```
var recipe = compose(cook,
                     map(sliceAndDice),
                     selectIngredients);
```

functional program:
a machine for transforming data

functions are built from other functions

via compose & combine

currying allows to specialize

a generic function

III. Advanced Concepts

Recursion

http://commons.wikimedia.org/wiki/File:Romanesco_Broccoli_detail_-_(1).jpg

Cooking in a Monad

```
hell({
    parameter : someParameter,
    callback : function(status) {
        if (status == states.SUCCESS) {
          b(function(status) {
              if (status == states.SUCCESS) {
                 c(function(status){
                     if (status == states.SUCCESS) {
                         // Not an exaggeration. I have seen
                         // code that looks like this regularly.
                     }
                 });
              }
          });
        } elseif (status == states.PENDING {
          ...
        }
    }
});
```

http://stackoverflow.com/questions/25098066/what-is-callback-hell-and-how-and-why-rx-solves-it

artisanal ;

```
function foo (bar) {
  console.log(bar); return bar.length
}

foo("hallo");
> hallo
> 5
```

```
function foo (bar) {
        (function () { console.log(bar);  })();
return (function () { return bar.length; })();
}

foo("hallo");
> hallo
> 5
```

```
function foo (bar, fun_a, fun_b) {

            (function (as) { fun_a.call(this, as); })(bar);
    return (function (bs) {
                    return fun_b.call(this, bs); })(bar);

}

function length(s) { return s.length; }

foo('hallo', console.log, length);
> hallo
> 5
```

```
function bang(s) { console.log(s + '!'); }

> foo('this works', console.log,
    function () { return
   foo('like a semicolon', bang,
    length); });
this works
like a semicolon!
16
```

```
function bang(s) { console.log(s + '!'); }

> mbind('this works', console.log,
    function () { return
   mbind('like a semicolon', bang,
    length); });
this works
like a semicolon!
16
```

Step into the Monad

```
function Pot (contents) {
  this.contents = contents;
}
```

```
function Pot (contents) {
  this.contents = contents;
}

var a = new Pot(1);
```

```
function Pot (contents) {
  this.contents = contents;
}

var a = new Pot(1);
var b = new Pot(2);
```

```
function step (i) {
  return new Pot(i + 1);
}

var a = new Pot(1);
var b = step(a.contents);
```

```
function step (i) {
  return new Pot(i + 1);
}

var a = new Pot(1);
var b = step(a.contents);
```

Pot(value)

```
function step (i) {
  return i + 1;
}

var a = 1;
var b = step(a);
```

value

The Monad Pattern

```
Pot.prototype.mbind = function (f) {
  return f.call(this, this.contents);
};


var a = new Pot(1);
> { contents: 1 }

var b = a.mbind(step);
> { contents: 2 }

var c = b.mbind(step);
> { contents: 3 }

c.contents;
> 3
```

```
Pot.prototype.mbind = function (f) {
  return f.call(this, this.contents);
};
```

```
var a = new Pot(1);
> { contents: 1 }
```

value ➡ Pot(value)

```
var b = a.mbind(step);
> { contents: 2 }

var c = b.mbind(step);
> { contents: 3 }

c.contents;
> 3
```

Pot(value) ➡ Pot(value)

unit / return

value ➡ M (value)

mbind / >>= / bind

M (value) ➡ M (value)

What's for dessert?


Purely functional
goodness
https://www.etsy.com/shop/PaperPartyEvents

abstraction

very localized
understanding of code

Explicit

side effects
error handling

reusability

Das Curry-Buch

Funktional programmieren lernen mit JavaScript