webmasters akademie Nürnberg GmbH

(CC) Moureen - https://www.flickr.com/photos/amerune/9294639633

CleanCode by Point-Free Programming

@MarcoEmrich #CCD16

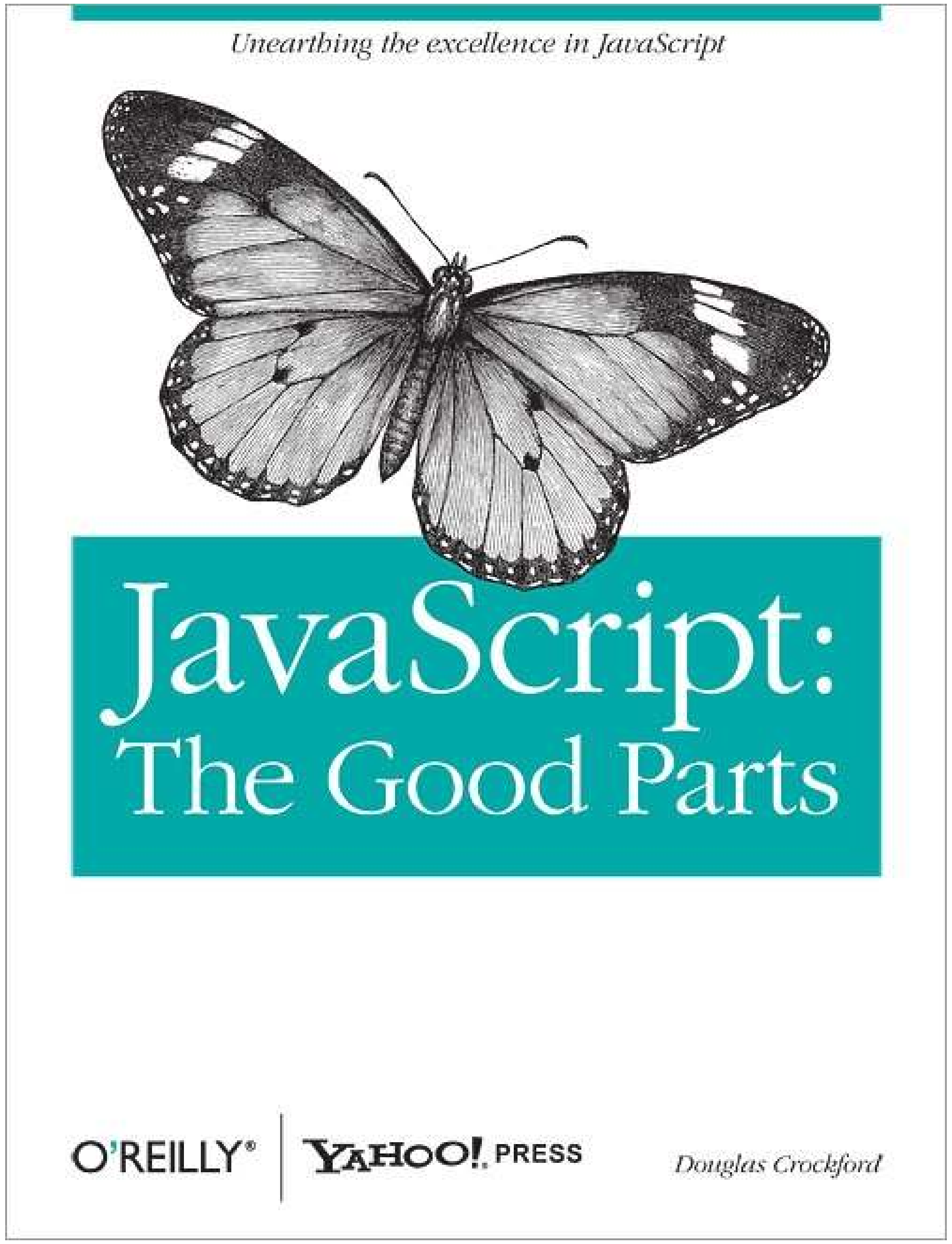
Example

String Calculator Kata

ROY OSHEROVE

'1,2,3' => 6

JavaScript



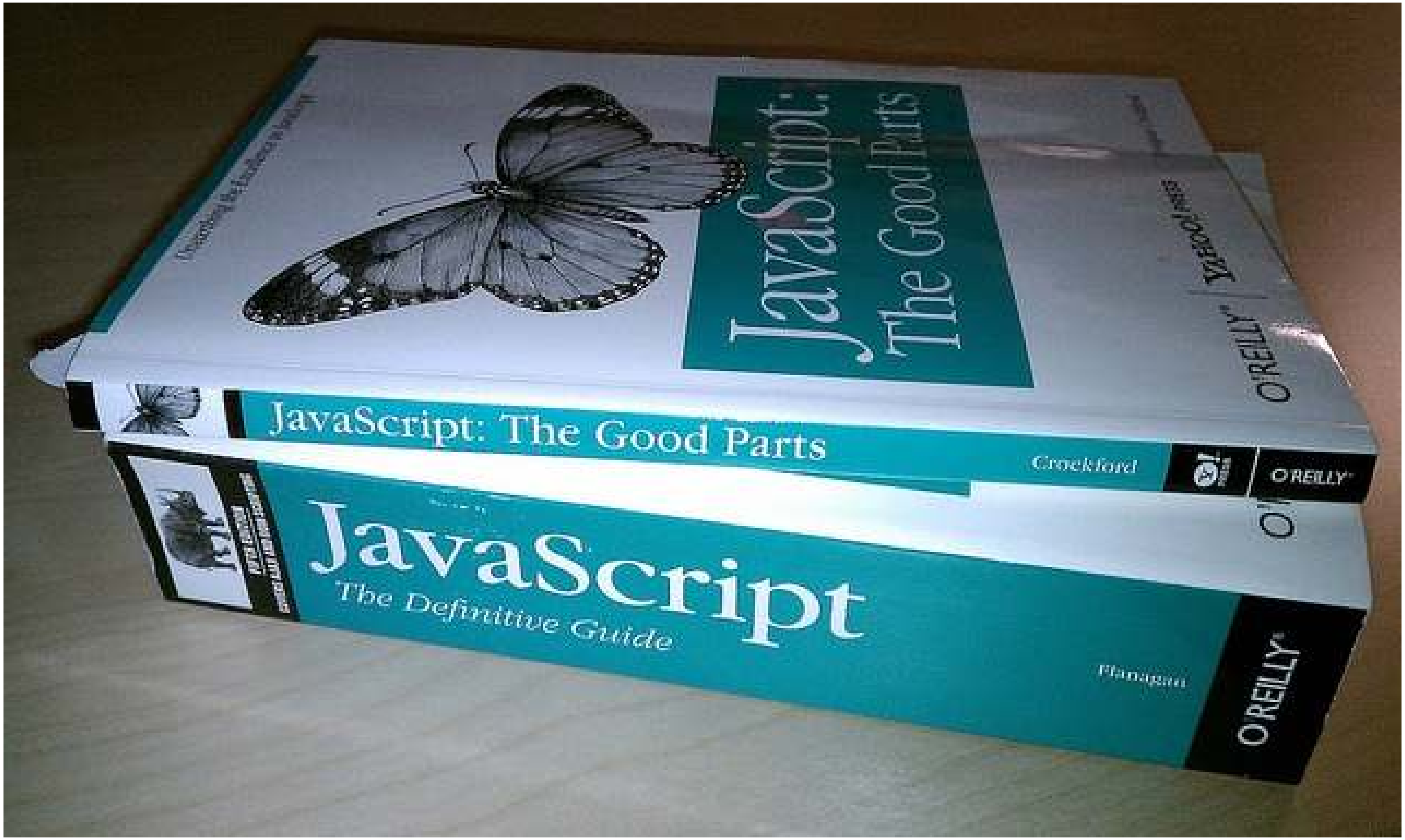
Unearthing the excellence in JavaScript

JavaScript: The Good Parts

O'REILLY® | YAHOO! PRESS

Douglas Crockford

The Good Parts



Source: http://www.michaelthelin.se

♥ JS ♥

String Calculator Kata

ROY OSHEROVE

'1,2,3' => 6

Classic Imperative

StringCalculator: Split

```
function stringCalc(str) {
  var parts = str.split(',');
  return parts;
}

stringCalc("1,1000,2");
```

StringCalculator: For

```
function stringCalc(str) {
  var parts, numbers;

  parts = str.split(',');

  for (i = 0; i < parts.length; ++i) {
    d(parts[i]);
  }

  return parts;
}

stringCalc("1,1000,2");
```

StringCalculator: Number

```javascript
function stringCalc(str) {
  var parts, number, i;

  parts = str.split(',');

  for (i = 0; i < parts.length; ++i) {
    number = Number(parts[i]);
    d(number);
  }

  return parts;
}

stringCalc("1,1000,2");
```

StringCalculator: Sum

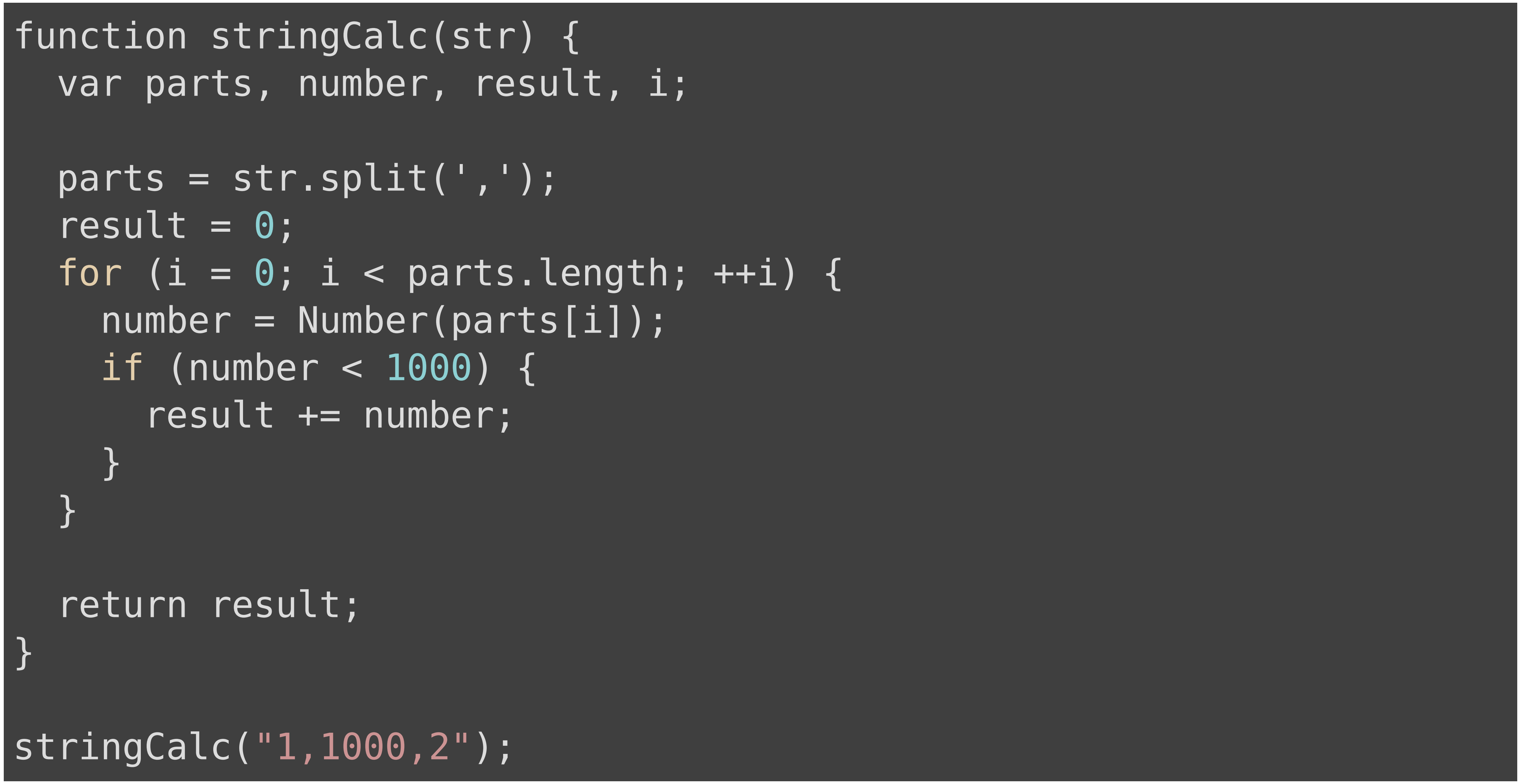
```
function stringCalc(str) {
  var parts, number, result, i;

  parts = str.split(',');
  result = 0;
  for (i = 0; i < parts.length; ++i) {
    number = Number(parts[i]);
    result += number;
  }

  return result;
}

stringCalc("1,1000,2");
```

New Requirement

ROY OSHEROVE

ignore >= 1000

StringCalculator: >= 1000



```
function stringCalc(str) {
  var parts, number, result, i;

  parts = str.split(',');
  result = 0;
  for (i = 0; i < parts.length; ++i) {
    number = Number(parts[i]);
    if (number < 1000) {
      result += number;
    }
  }

  return result;
}

stringCalc("1,1000,2");
```

Clean Code?

(CC) Sarah Laval - https://www.flickr.com/photos/smercury98/2446660754

(CC) Rafael Vianna Croffi - https://www.flickr.com/photos/rvc/8699750970

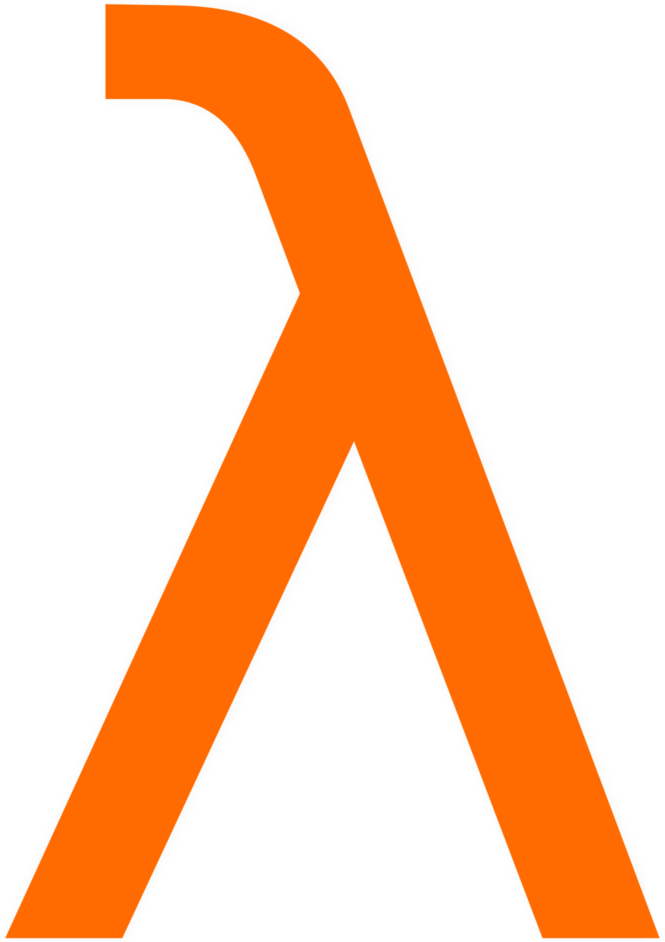
(CC) Stephen Pierzchala - https://www.flickr.com/photos/spierzchala/66232046

(CC) Maurizio - https://www.flickr.com/photos/maurizio-sorvillo/8739112730

Functional

Library

Photo: Michael D Beckwith (https://www.flickr.com/photos/118118485@N05/15817405853), CC BY-ND 2.0 λ



Lambda

Lamb

Photo: Susanne Nilsson
CC BY-SA 2.0 - https://www.flickr.com/photos/infomastern/19267372820

Grows Up to …

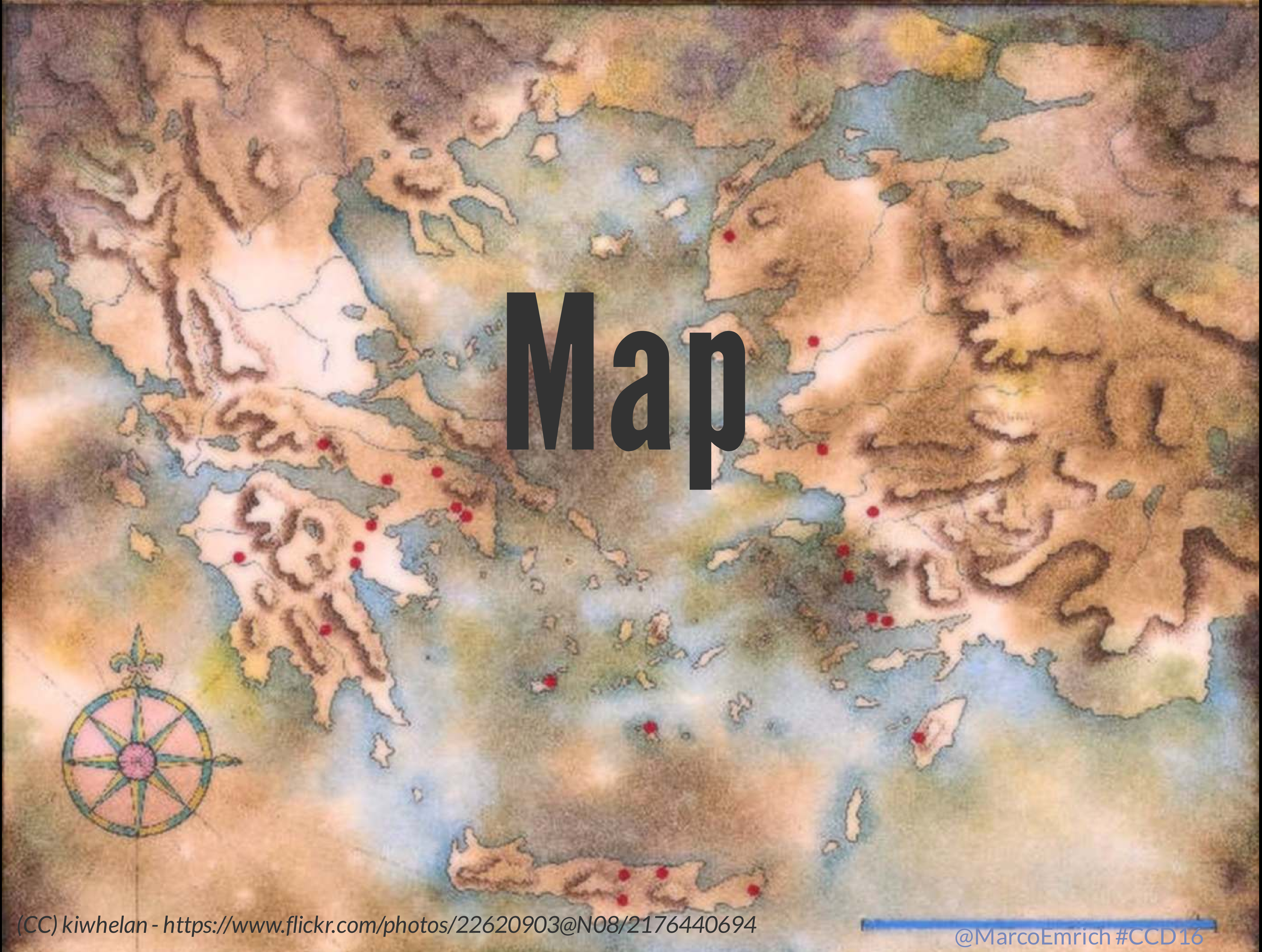
Ram

CC BY-ND 2.0 - https://www.flickr.com/photos/tattiehowker/7014970659

Ramda

Ramda

```
function stringCalc(str) {
  var parts = R.split(',', str);

  return parts;
}

stringCalc("1,1000,2");
```



Map

(CC) kiwhelan - https://www.flickr.com/photos/22620903@N08/2176440694

Map

```
R.map(n => 2 * n, [1, 2, 3])
```

Convert numbers

```
function stringCalc(str) {
  var parts, numbers;

  parts = R.split(',', str);
  numbers = R.map(Number, parts);
  return numbers;
}

stringCalc("1,1000,2");
```

Sum

```
R.sum( [1, 2, 3] )
```

Sum

```
function stringCalc(str) {
  var parts, numbers;

  parts = R.split(',', str);
  numbers = R.map(Number, parts);
  return R.sum(numbers);
}

stringCalc("1,1000,2");
```

Filter

(CC) Ralph Aichinger https://www.flickr.com/photos/sooperkuh/6360506353

Filter

```
R.filter(n => n < 1000, [1, 2, 3, 500, 1000, 1001, 2000])
```

StringCalculator

```
function stringCalc(str) {
  var parts, numbers, under1000s;

  parts = R.split(',', str);
  numbers = R.map(Number, parts);
  under1000s = R.filter(n => n < 1000, numbers);
  return R.sum(under1000s);
}

stringCalc("1,1000,2");
```

Clean Code?

(CC) Sarah Laval - https://www.flickr.com/photos/smercury98/2446660754

(CC) Wagner T. Cassimiro "Aranha" - https://www.flickr.com/photos/wagnertc/3859388854

(CC) Postmemes - https://www.flickr.com/photos/postmemes/15891128273

(CC) Maurizio - https://www.flickr.com/photos/maurizio-sorvillo/8739112730

Point-Free

!Pointless

Currying

(CC) Karsten Seiferlin - https://www.flickr.com/photos/timecaptured/6182975764

Curry

```
R.add(3, 4)
```

Curry

```
const add3 = R.add(3);

//d(typeof(add3));

add3(4)
```

Curry

```
const splitByComma = R.split(',');
splitByComma("1,2,3,4,5");
```

Curry

```
const splitByComma = R.split(',');

function stringCalc(str) {
  var numbers, under1000s, parts;

  parts = splitByComma(str);
  numbers = R.map(Number, parts);
  under1000s = R.filter(n => n < 1000, numbers);
  return R.sum(under1000s);
}

stringCalc("1,1000,2");
```

Curry

```
const splitByComma = R.split(',');
const mapToNumber = R.map(Number);
const filterUnder1000 = R.filter(n => n < 1000);

function stringCalc(str) {
  var numbers, under1000s, parts;

  parts = splitByComma(str);
  numbers = mapToNumber(parts);
  under1000s = filterUnder1000(numbers);
  return R.sum(under1000s);
}

stringCalc("1,1000,2");
```

Curry

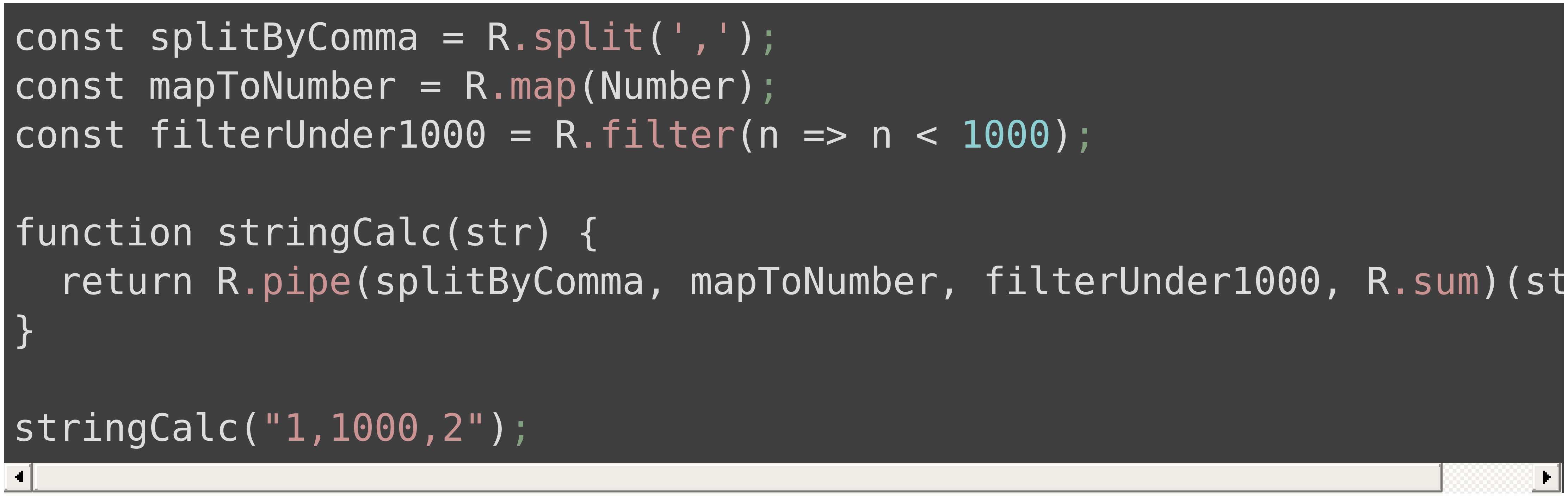
```
const splitByComma = R.split(',');
const mapToNumber = R.map(Number);
const filterUnder1000 = R.filter(n => n < 1000);

function stringCalc(str) {
  return R.sum(filterUnder1000(mapToNumber(splitByComma(str))));
}

stringCalc("1,1000,2");
```

((((((WTF?))))))

Pipeline

(CC) Moureen - https://www.flickr.com/photos/amerune/9294639633

Pipe



```
const splitByComma = R.split(',');
const mapToNumber = R.map(Number);
const filterUnder1000 = R.filter(n => n < 1000);

function stringCalc(str) {
  return R.pipe(splitByComma, mapToNumber, filterUnder1000, R.sum)(st
}

stringCalc("1,1000,2");
```

Pointfree

```
const splitByComma = R.split(',');
const mapToNumber = R.map(Number);
const filterUnder1000 = R.filter(n => n < 1000);

const stringCalc = R.pipe(
        splitByComma,
        mapToNumber,
        filterUnder1000,
        R.sum);

stringCalc("1,1000,2");
```

StringCalculator

```
const stringCalc = R.pipe(
  R.split(','),
  R.map(Number),
  R.filter(n => n < 1000),
  R.sum);

stringCalc("1,1000,2");
```

StringCalculator

```
//import { pipe, split, map, filter, sum } from 'ramda'

const stringCalc = pipe(
  split(','),
  map(Number),
  filter(n => n < 1000),
  sum);

stringCalc("1,1000,2");
```

StringCalculator - Comparison

```
const stringCalc = pipe(
  split(','),
  map(Number),
  filter(n => n < 1000),
  sum);
```

```
function stringCalc(str) {
  var parts, number, result, i;

  parts = str.split(',');
  result = 0;
  for (i = 0; i < parts.length; ++i)
    number = Number(parts[i]);
    result += number;
  }

  return result;
}
```

Clean Code?

(CC) Sarah Laval - https://www.flickr.com/photos/smercury98/2446660754

short and readable

OOP
meets
Point-Free

Vanilla JavaScript

```
const stringCalc = str => str
  .split(',')
  .map(Number)
  .filter(n => n < 1000)
  .reduce((a, b) => a + b);

stringCalc("1,1000,2");
```

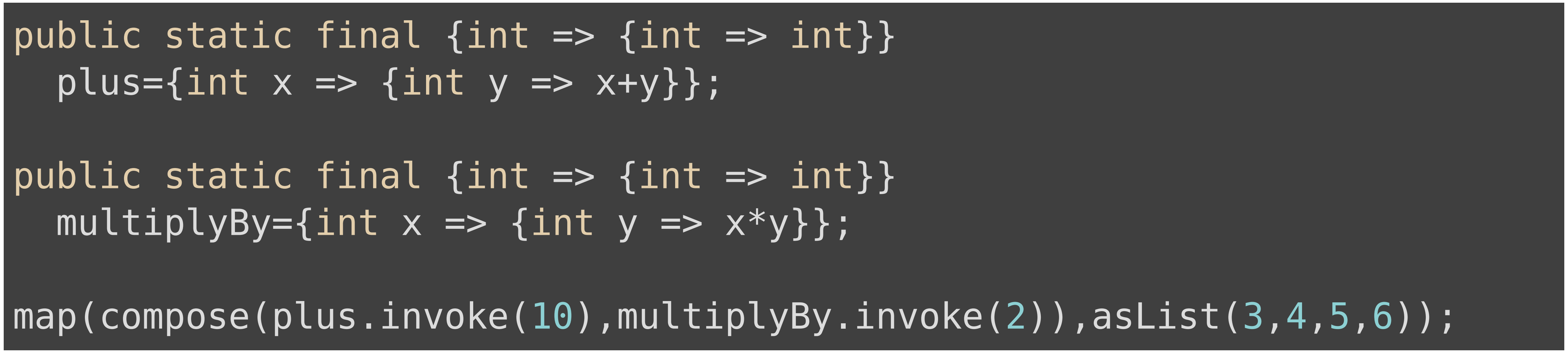
Other Languages

Java 8

```
map({int x => x*2}, asList(3,4,5,6,7));
```

Source: http://rickyclarkson.blogspot.de/2007/09/point-free-programming-in-java-7-beyond.html

Java 8



```
public static final {int => {int => int}}
  plus={int x => {int y => x+y}};

public static final {int => {int => int}}
  multiplyBy={int x => {int y => x*y}};

map(compose(plus.invoke(10),multiplyBy.invoke(2)),asList(3,4,5,6));
```

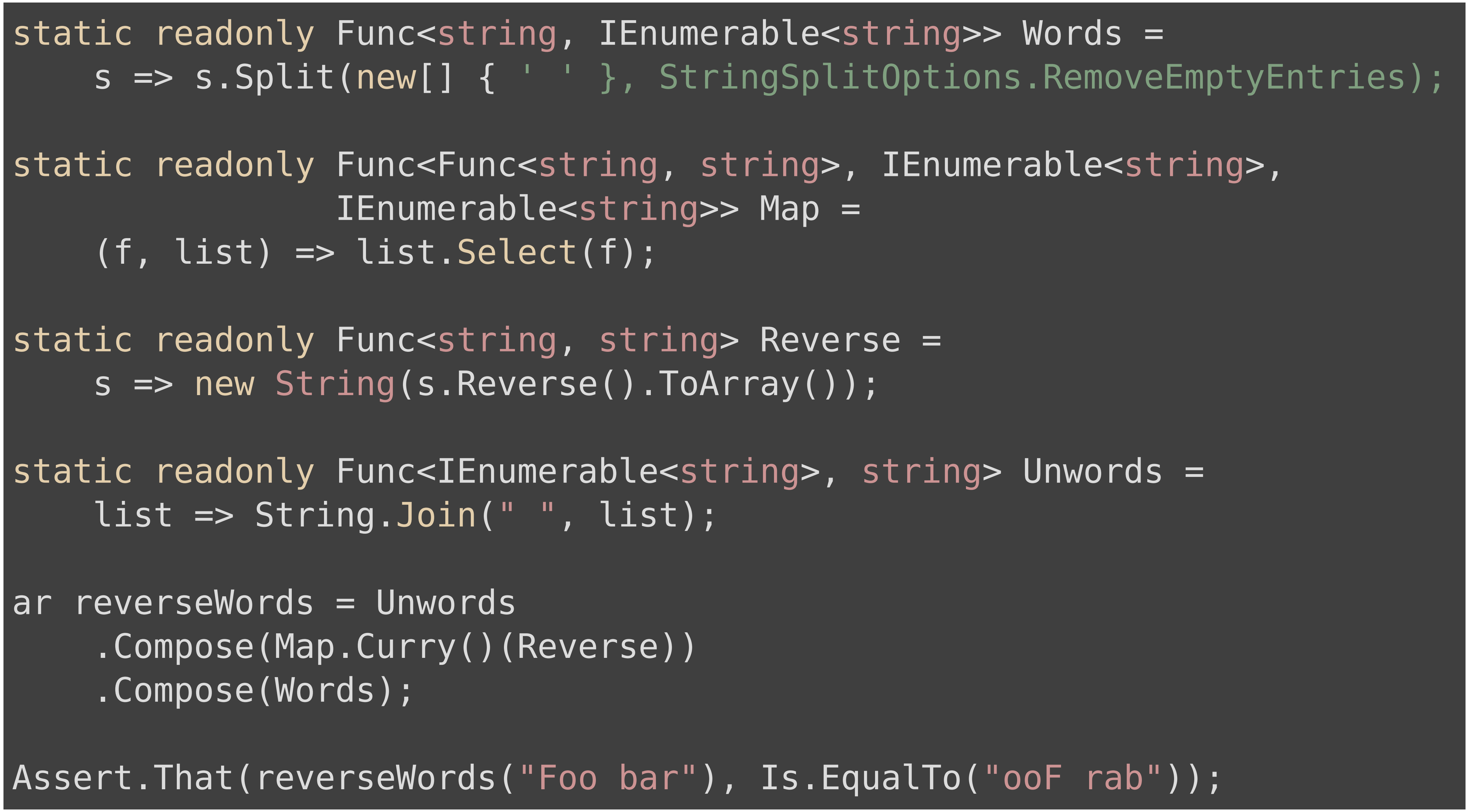
Source: http://rickyclarkson.blogspot.de/2007/09/point-free-programming-in-java-7-beyond.html

Java 8

```java
List<Student> students = persons.stream()
        .filter(p -> p.getAge() > 18)
        .map(Student::new)
        .collect(Collectors.toCollection(ArrayList::new));
```

Source: http://zeroturnaround.com/rebellabs/java-8-explained-applying-lambdas-to-java-collections

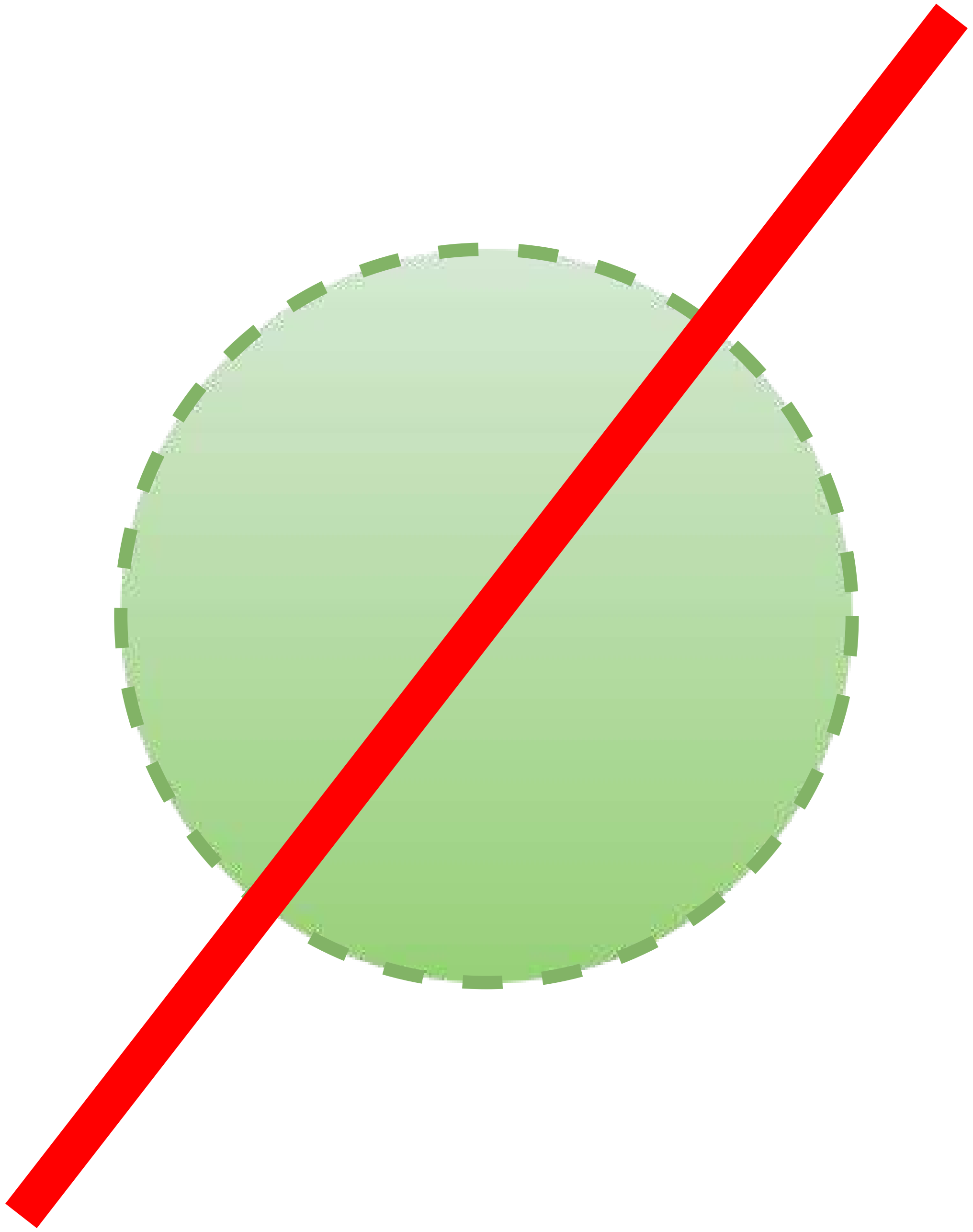
C#



```
static readonly Func<string, IEnumerable<string>> Words =
    s => s.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

static readonly Func<Func<string, string>, IEnumerable<string>,
                IEnumerable<string>> Map =
    (f, list) => list.Select(f);

static readonly Func<string, string> Reverse =
    s => new String(s.Reverse().ToArray());

static readonly Func<IEnumerable<string>, string> Unwords =
    list => String.Join(" ", list);

ar reverseWords = Unwords
    .Compose(Map.Curry()(Reverse))
    .Compose(Words);

Assert.That(reverseWords("Foo bar"), Is.EqualTo("ooF rab"));
```

Source: http://blog.leifbattermann.de/2015/06/04/function-composition-in-csharp

Becoming Point Free



Two Baby Steps

(CC) Bill G. - https://www.flickr.com/photos/billerr/1814657036

Step 1

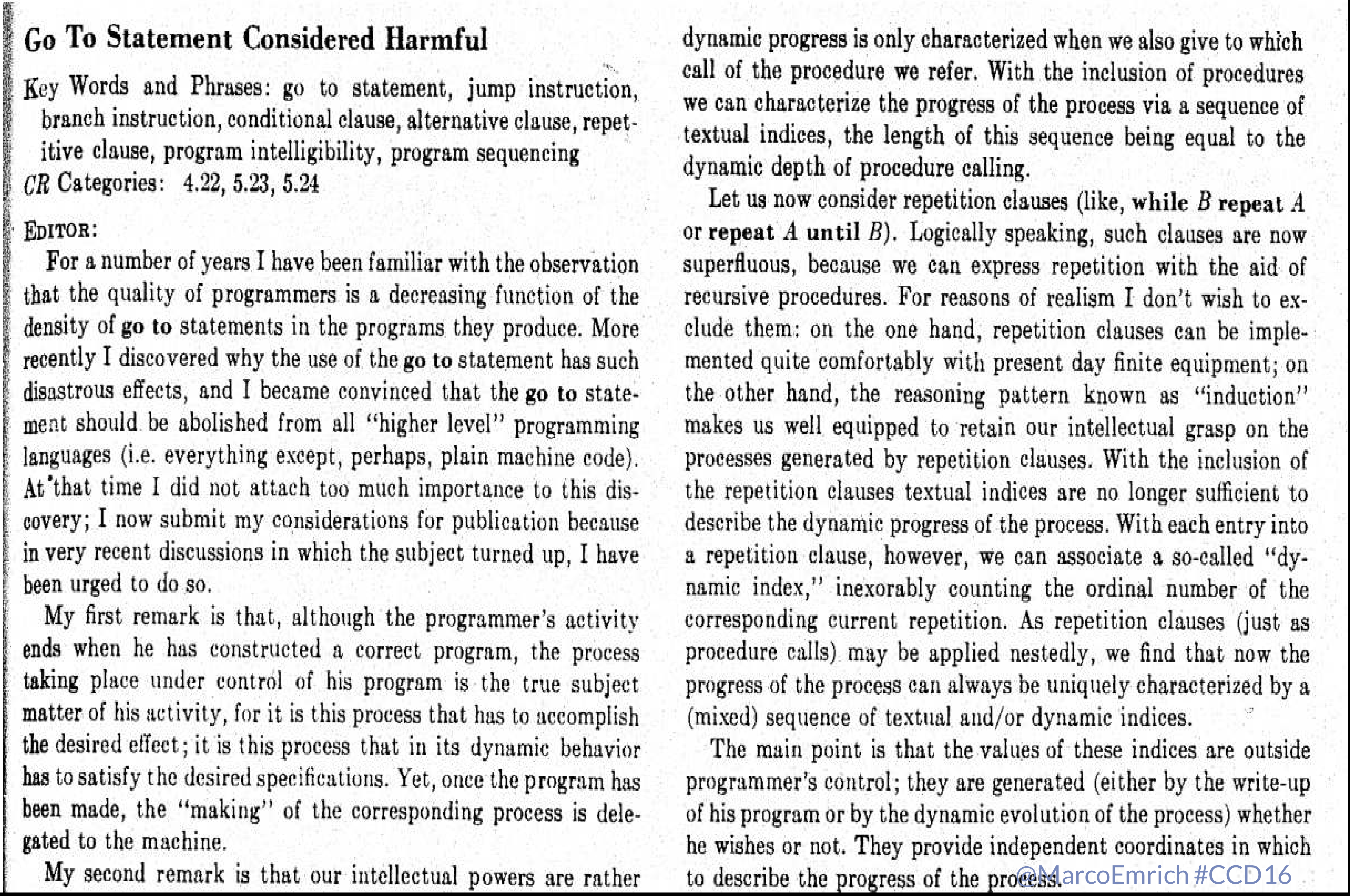
Replace »for« with High Order Functions

~~FOR~~ ⇨

Map
Filter
Reduce
…

»for« considered harmful?

Edgar Dijkstra: Go To Statement Considered Harmful



Go To Statement Considered Harmful

Key Words and Phrases: go to statement, jump instruction, branch instruction, conditional clause, alternative clause, repetitive clause, program intelligibility, program sequencing
CR Categories: 4.22, 5.23, 5.24

Editor:

For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of **go to** statements in the programs they produce. More recently I discovered why the use of the **go to** statement has such disastrous effects, and I became convinced that the **go to** statement should be abolished from all "higher level" programming languages (i.e. everything except, perhaps, plain machine code). At that time I did not attach too much importance to this discovery; I now submit my considerations for publication because in very recent discussions in which the subject turned up, I have been urged to do so.

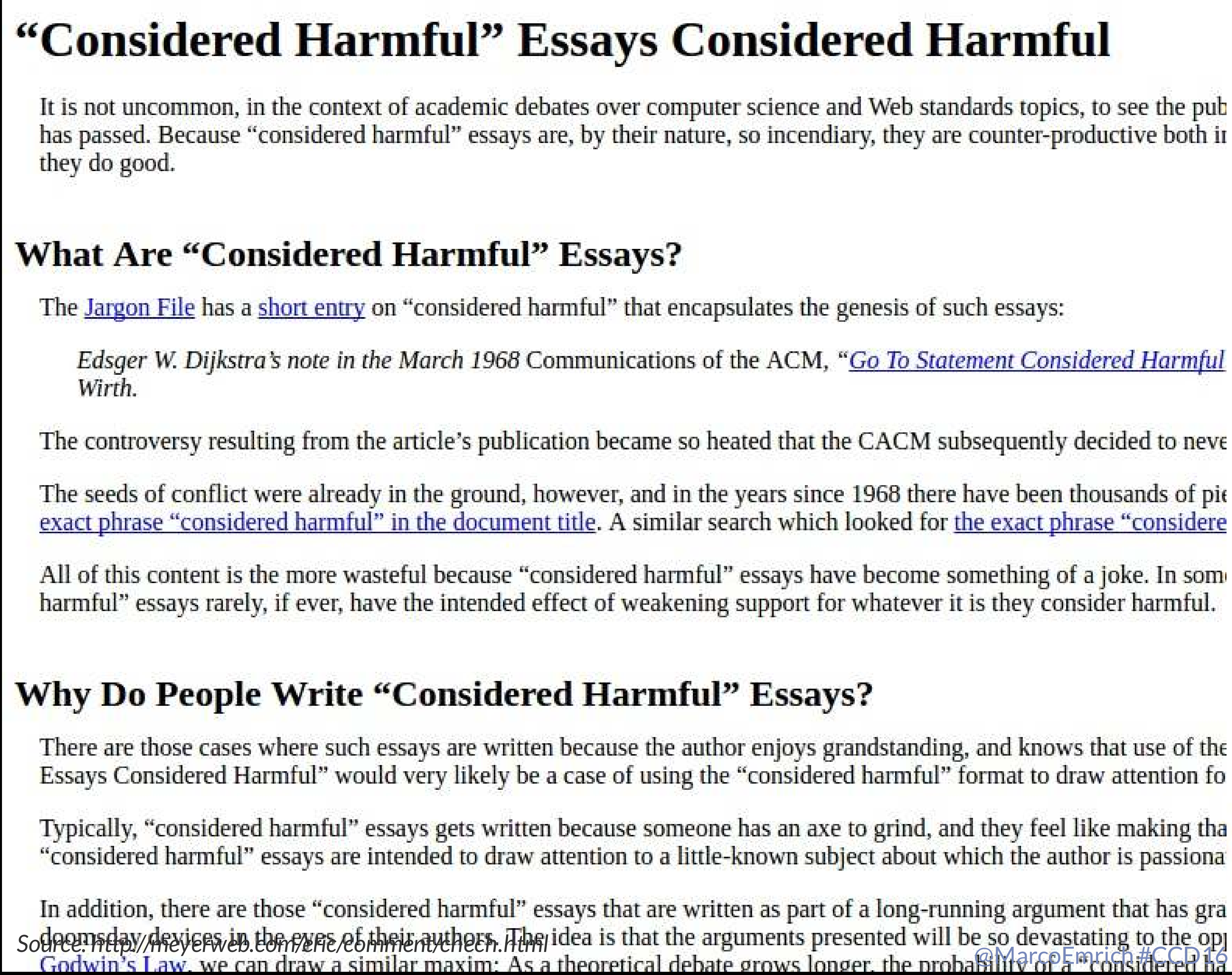
My first remark is that, although the programmer's activity ends when he has constructed a correct program, the process taking place under control of his program is the true subject matter of his activity, for it is this process that has to accomplish the desired effect; it is this process that in its dynamic behavior has to satisfy the desired specifications. Yet, once the program has been made, the "making" of the corresponding process is delegated to the machine.

My second remark is that our intellectual powers are rather

dynamic progress is only characterized when we also give to which call of the procedure we refer. With the inclusion of procedures we can characterize the progress of the process via a sequence of textual indices, the length of this sequence being equal to the dynamic depth of procedure calling.

Let us now consider repetition clauses (like, **while** *B* **repeat** *A* or **repeat** *A* **until** *B*). Logically speaking, such clauses are now superfluous, because we can express repetition with the aid of recursive procedures. For reasons of realism I don't wish to exclude them: on the one hand, repetition clauses can be implemented quite comfortably with present day finite equipment; on the other hand, the reasoning pattern known as "induction" makes us well equipped to retain our intellectual grasp on the processes generated by repetition clauses. With the inclusion of the repetition clauses textual indices are no longer sufficient to describe the dynamic progress of the process. With each entry into a repetition clause, however, we can associate a so-called "dynamic index," inexorably counting the ordinal number of the corresponding current repetition. As repetition clauses (just as procedure calls) may be applied nestedly, we find that now the progress of the process can always be uniquely characterized by a (mixed) sequence of textual and/or dynamic indices.

The main point is that the values of these indices are outside programmer's control; they are generated (either by the write-up of his program or by the dynamic evolution of the process) whether he wishes or not. They provide independent coordinates in which to describe the progress of the process.



“Considered Harmful” Essays Considered Harmful

It is not uncommon, in the context of academic debates over computer science and Web standards topics, to see the pub has passed. Because “considered harmful” essays are, by their nature, so incendiary, they are counter-productive both ir they do good.

What Are “Considered Harmful” Essays?

The Jargon File has a short entry on “considered harmful” that encapsulates the genesis of such essays:

> *Edsger W. Dijkstra’s note in the March 1968* Communications of the ACM, “*Go To Statement Considered Harmful Wirth.*

The controversy resulting from the article’s publication became so heated that the CACM subsequently decided to neve

The seeds of conflict were already in the ground, however, and in the years since 1968 there have been thousands of pie exact phrase “considered harmful” in the document title. A similar search which looked for the exact phrase “considere

All of this content is the more wasteful because “considered harmful” essays have become something of a joke. In som harmful” essays rarely, if ever, have the intended effect of weakening support for whatever it is they consider harmful.

Why Do People Write “Considered Harmful” Essays?

There are those cases where such essays are written because the author enjoys grandstanding, and knows that use of the Essays Considered Harmful” would very likely be a case of using the “considered harmful” format to draw attention fo

Typically, “considered harmful” essays gets written because someone has an axe to grind, and they feel like making tha “considered harmful” essays are intended to draw attention to a little-known subject about which the author is passiona

In addition, there are those “considered harmful” essays that are written as part of a long-running argument that has gra doomsday devices in the eyes of their authors. The idea is that the arguments presented will be so devastating to the opp Godwin’s Law, we can draw a similar maxim: As a theoretical debate grows longer, the probability of a “considered ha

Source: http://meyerweb.com/eric/comment/chech.html

@MarcoEmrich #CCD16

Replace »for« with High Order Functions ...where it makes sense :)

~~FOR~~ →

Map
Filter
Reduce
...

Step 2

Build Pipelines

(CC) Moureen - https://www.flickr.com/photos/amerune/9294639633

Enjoy Your Clean Code

(CC) Jo Anthony Fortugaleza - https://www.flickr.com/photos/30594175@N05/3277413297