Form over Function

Teaching Beginners How to Construct Programs

Michael Sperber

Collaborators:
Marcus Crestani, Martin Gasbichler,
Herbert Klaeren, Eric Knauel
@ University of Tübingen

Back at the Ranch ...

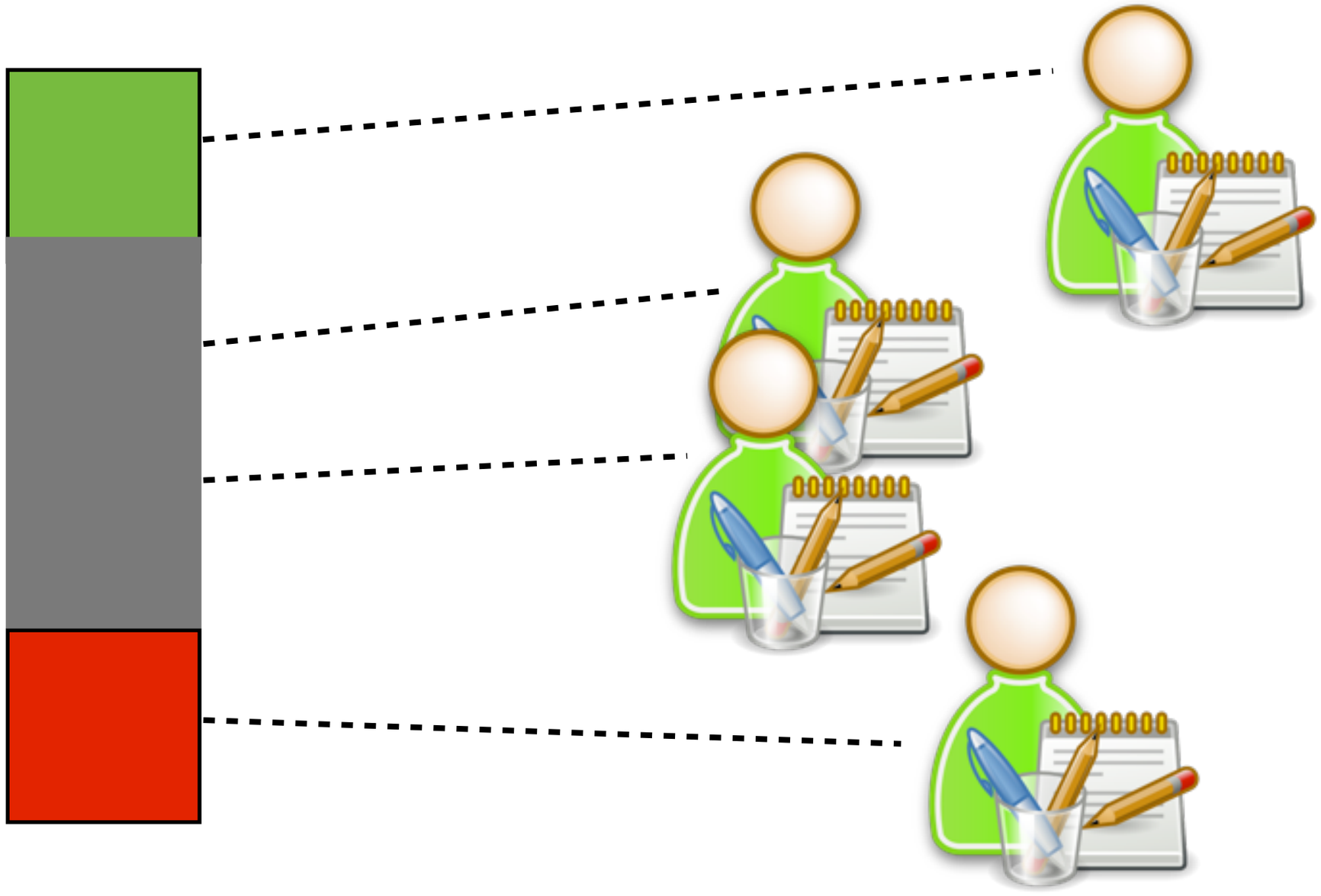
MORE THAN 600,000 COPIES SOLD

CHILDREN: THE CHALLENGE

The Classic Work on Improving Parent-Child Relations—Intelligent, Humane, and Eminently Practical

RUDOLF DREIKUS, M.D.,
with Vicki Soltz, R.N.

Volker Claus's Trick



College announces investigation

Inappropriate collaboration alleged on a take-home final

File photo by Rose Lincoln/Harvard Staff Photographer

"We take academic integrity very seriously because it goes to the heart of our educational mission," said Michael D. Smith, dean of the Faculty of Arts and Sciences. "Academic dishonesty cannot and will not be tolerated at Harvard."

mindset

THE NEW PSYCHOLOGY OF SUCCESS

HOW WE CAN LEARN TO FULFILL OUR POTENTIAL

*parenting
*business
*school
*relationships

"Will prove to be one of the most influential books ever about motivation."
—Po Bronson, author of *NurtureShock*

CAROL S. DWECK, Ph.D.

So How Is This About Scheme?

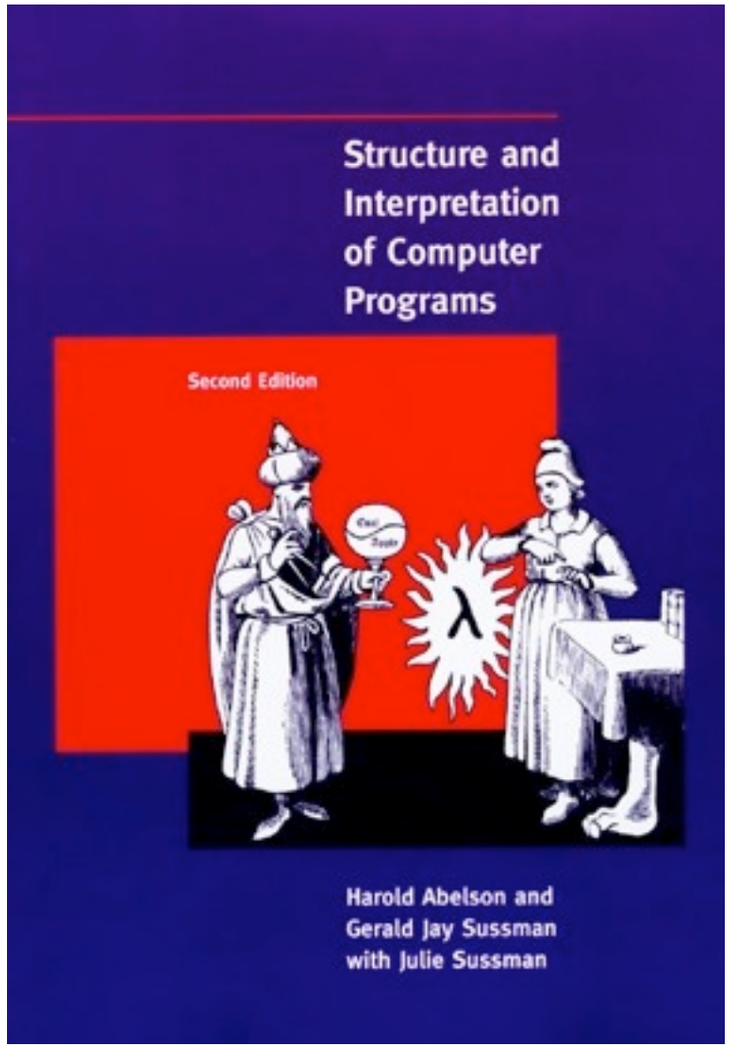
U   ?

Self-Deception

- plagiarism
- stronger students are more vocal
- strong students teach themselves

Scheme Is Great For Beginners

- Syntax
- Size
- Functional
- Easy to transition to X



Scheme Is Great For Beginners

- Syntax
- Size
- Functional
- Easy to translate to X

Wrong

Questions!

What Is Important to You?

Dictionary

sys•tem•at•ic | ˌsistəˈmatik |

adjective

done or acting according to a fixed plan or system; methodical: *a systematic search of the whole city.*

DERIVATIVES

sys•tem•at•i•cal•ly | -ik(ə)lē |adverb,

sys•tem•a•tist | ˈsistəməˌtist |noun

ORIGIN early 18th cent.: from French ***systématique***, via late Latin from late Greek ***sustēmatikos***, from ***sustēma*** (see **SYSTEM**) .

Geometric Shapes

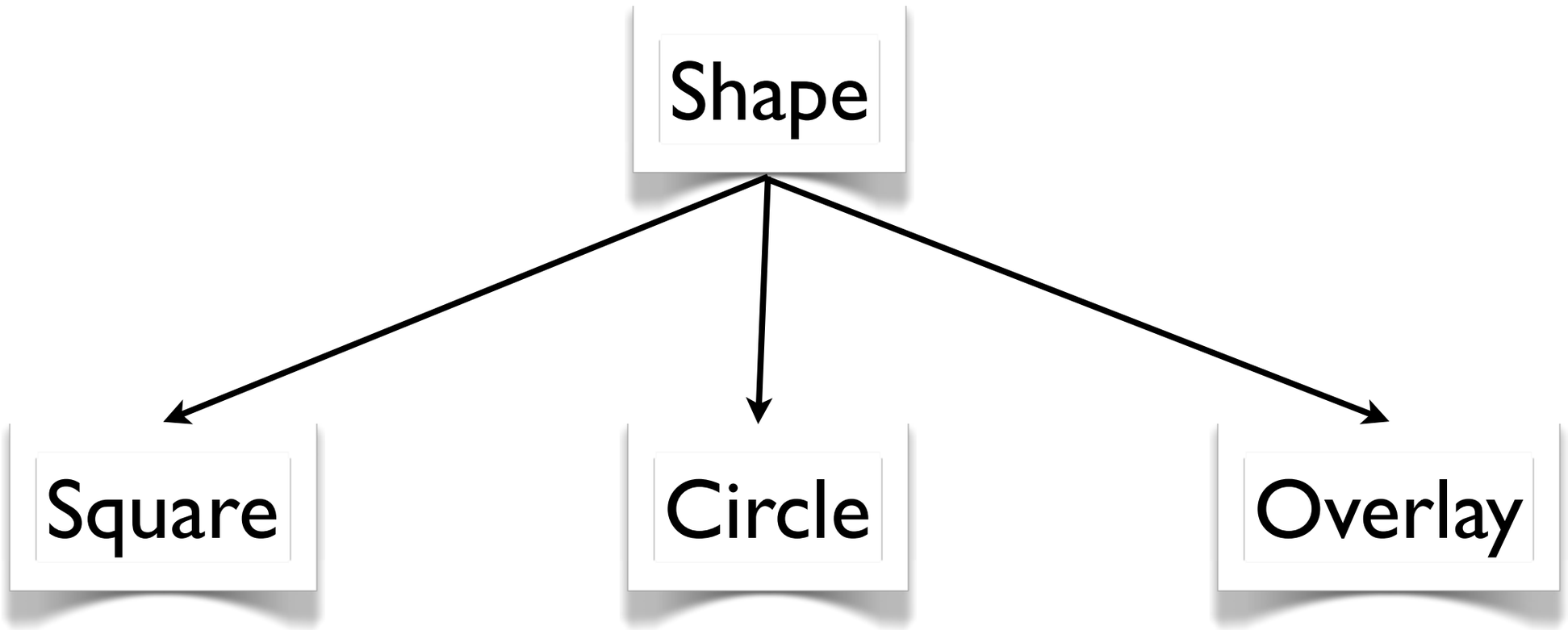
A geometric shape is one of the following:

- square (parallel to axes)
- circle
- overlay of 2 geometric shapes

Geometric Shapes

Implement geometric shapes! Write a program that allows creating geometric shapes und to check whether a given point is inside our outside a geometric shape!

Geometric Shapes



Design Recipes

How to organize the composition. Sometimes, a particular assignment will not exactly fit into this outline form, but, generally, the form can be used as a guide to check against to be certain you are putting together your composition correctly.

I. Introduction (usually is 1 paragraph in length)

A. Attention Step

B. Background Information

1. any information required for an understanding of the thesis statement. For exampl
2. when a paper is analyzing a story, include its title, author, and some brief plot
etc. information.

C. Thesis Statement

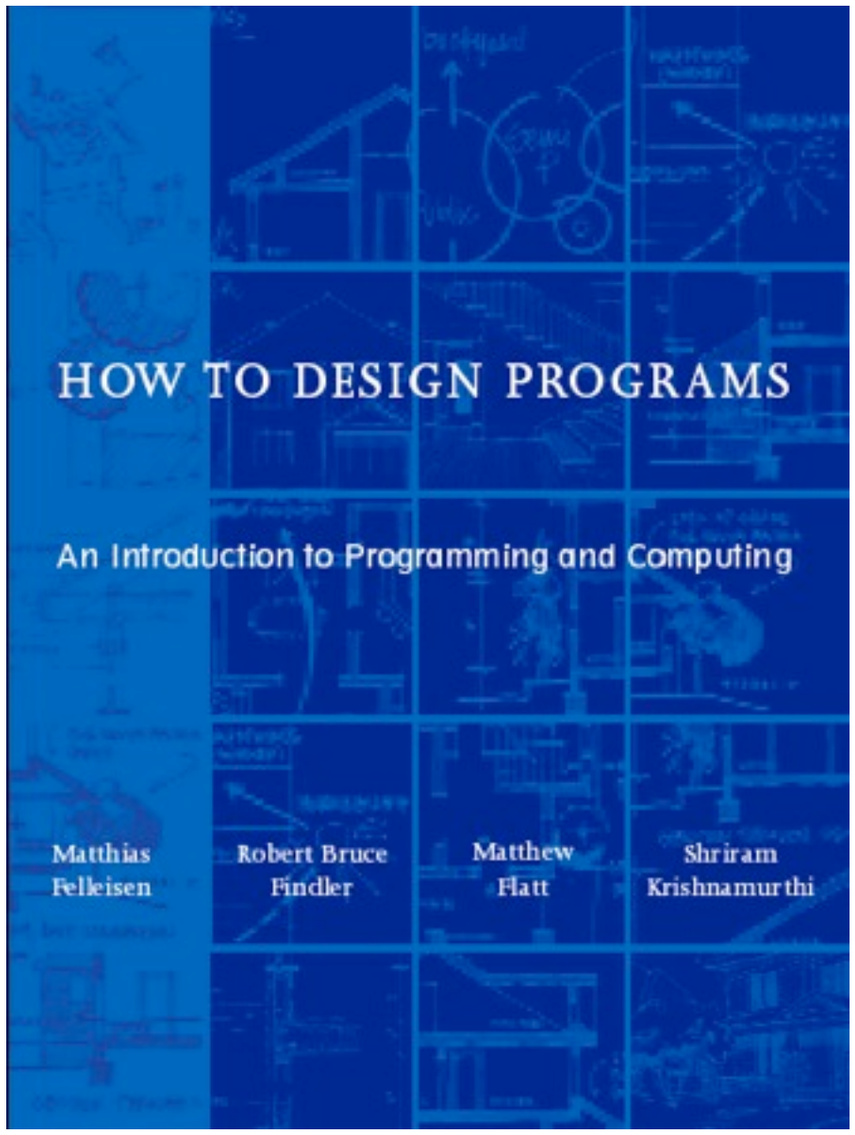
1. purpose
2. scope
   a.
   b.
   c.
3. direction

II. Body (usually is 3 paragraphs, with each paragraph developing one of the areas of the thesis)

A. First Area of Scope (usually one paragraph)

1. transition
2. topic sentence
3. further explanation/clarification of the topic sentence
4. amplification of the topic sentence
   a. examples, details, proofs, quotes, etc., that support the topic sentence in
   b. some way

U


HOW TO DESIGN PROGRAMS
An Introduction to Programming and Computing
Matthias Felleisen
Robert Bruce Findler
Matthew Flatt
Shriram Krishnamurthi

?

Data Analysis

- shapes
- squares
- circles
- overlays
- points
- (2-dimensional plane)

Mixed Data

A geometric shape is one of the following:

- a circle
- a square
- an overlay

Composite Data

A circle has:

- center
- radius

Design Recipe

"When your data analysis contains composite data, identify the signatures of the components. Then write a data definition starting with the following:

```
; An x consists of / has:
; - field1 (sig1)
; ...
; - fieldn (sign)
```

Design Recipe

Then translate the data definition into a record definition:

```
(define-record-procedures sig
  constr pred?
  (select1 ... selectn)
```

Design Recipe

Also write a constructor signature of the following form:

```
(: constr (sig1 ... sign -> sig))
```

Also, write signatures for the predicate and the selectors:

```
(: pred? (any -> boolean))
(: select1 (sig -> sig1))
...
(: selectn (sig -> sign))
```

Circles

```
; A circle consists of:
; - center (point)
; - radius (real)
(define-record-procedures circle
  make-circle circle?
  (circle-center circle-radius))
(: make-circle (point real -> circle))
(: circle? (any -> boolean))
(: circle-center (circle -> point))
(: circle-radius (circle -> real))
```

Composite Data

A square consists of:

- lower left corner
- size

Squares

```
; A square consists of:
; - lower left corner (point)
; - size / edge length (real)
(define-record-procedures square
  make-square square?
  (square-corner square-size))
(: make-square (point real -> square))
(: square? (any -> boolean))
(: square-corner (square -> point))
(: square-size (square -> real))
```

Composite Data with Self Reference

On overlay consists of:

- a geometric shape
- and another geometric shape

Overlays

```
; An overlay consists of:
; - a geometric shape “on top” (shape)
; - a geometric shape “on bottom” (shape)
(define-record-procedures overlay
  make-overlay overlay?
  (overlay-top-shape overlay-bot-shape))
(: make-overlay (shape shape -> overlay))
(: overlay? (any -> boolean))
(: overlay-top-shape (overlay -> shape))
(: overlay-bot-shape (overlay -> shape))
```

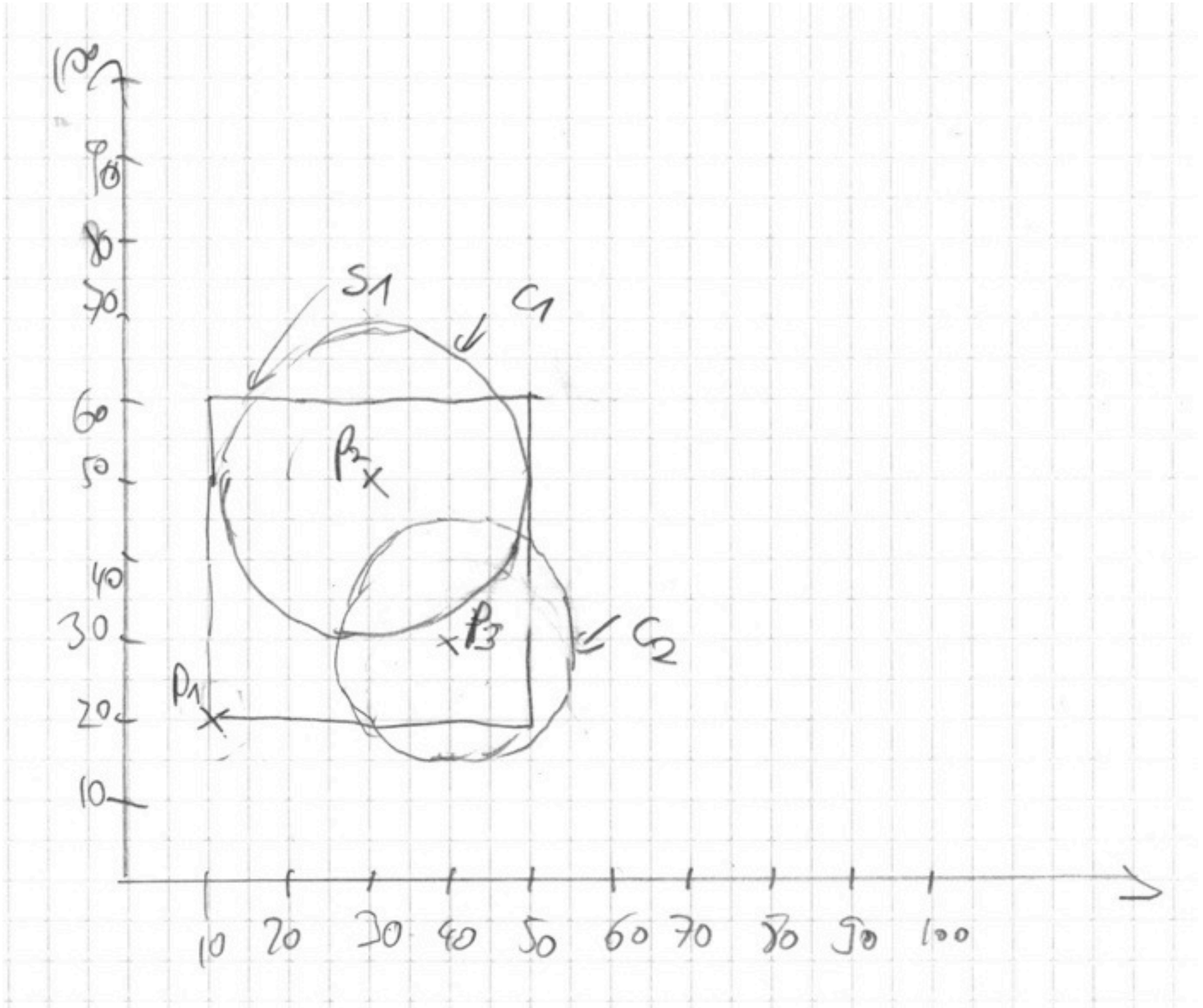
Points

```
; A point consists of:
; - x coordinate (real)
; - y coordinate (real)
(define-record-procedures point
  make-point point?
  (point-x point-y))
(: make-point (real real -> point))
(: point? (any -> boolean))
(: point-x (point -> real))
(: point-y (point -> real))
```

Geometric Shapes

```
; A geometric shape is one of the following:
; - a circle (circle)
; - a square parallel to the axes (square)
; - on overlay of two geometric figures (overlay)
(define shape
  (signature
   (mixed circle
          square
          overlay)))
```

Examples



Examples

```
(define p1 (make-point 10 20))   ; Point at X=10, Y=20
(define p2 (make-point 30 50))   ; Point at X=30, Y=50
(define p3 (make-point 40 30))   ; Point at X=40, Y=30
(define s1 (make-square p1 40))  ; Square w/ corner at p1, size 40
(define c1 (make-circle p2 20))  ; Circle around p2, radius 20
(define o1 (make-overlay c1 s1)) ; Overlay of circle und square
(define c2 (make-circle p3 15))  ; Circle around p3, radius 10
(define o2 (make-overlay o1 c2)) ; Overlay of o1 and c2
```

First Steps

short description

signature

```
; is a point within a shape?
(: point-in-shape? (point shape -> boolean))

(check-expect (point-in-shape? p2 c1) #t)
(check-expect (point-in-shape? p3 c2) #t)
(check-expect (point-in-shape? (make-point 51 50) c1) #f)
(check-expect (point-in-shape? (make-point 11 21) s1) #t)
(check-expect (point-in-shape? (make-point 49 59) s1) #t)
(check-expect (point-in-shape? (make-point 9 21) s1) #f)
(check-expect (point-in-shape? (make-point 11 19) s1) #f)
(check-expect (point-in-shape? (make-point 51 59) s1) #f)
(check-expect (point-in-shape? (make-point 49 61) s1) #f)

(check-expect (point-in-shape? (make-point 40 30) o2) #t)
(check-expect (point-in-shape? (make-point 0 0) o2) #f)
```

examples

Template

```
(define point-in-shape?
  (lambda (p s)
    ...))
```

Skeleton

```
(define point-in-shape?
  (lambda (p s)
    ... p ... s ...
    ... (point-x p) ... (point-y p) ...
    (cond
      ((circle? s) ...)
      ((square? s) ...)
      ((overlay? s) ...))))
```

More Skeleton

```
(define point-in-shape?
  (lambda (p s)
    ... p ... s ...
    ... (point-x p) ... (point-y p) ...
    (cond
      ((circle? s)
       ... (circle-center s) ... (circle-radius s) ...)
      ((square? s)
       ... (square-corner s) ... (square-size s) ...)
      ((overlay? s)
       ... (overlay-top-shape s) ... (overlay-bot-shape s) ...)))))
```

More Skeleton

```
(define point-in-shape?
  (lambda (p s)
    ... p ... s ...
    ... (point-x p) ... (point-y p) ...
    (cond
      ((circle? s)
       ... (circle-center s) ... (circle-radius s) ...)
      ((square? s)
       ... (square-corner s) ... (square-size s) ...)
      ((overlay? s)
       ... (point-in-shape? p (overlay-top-shape s))
       ... (point-in-shape? p (overlay-bot-shape s)) ...)))))
```

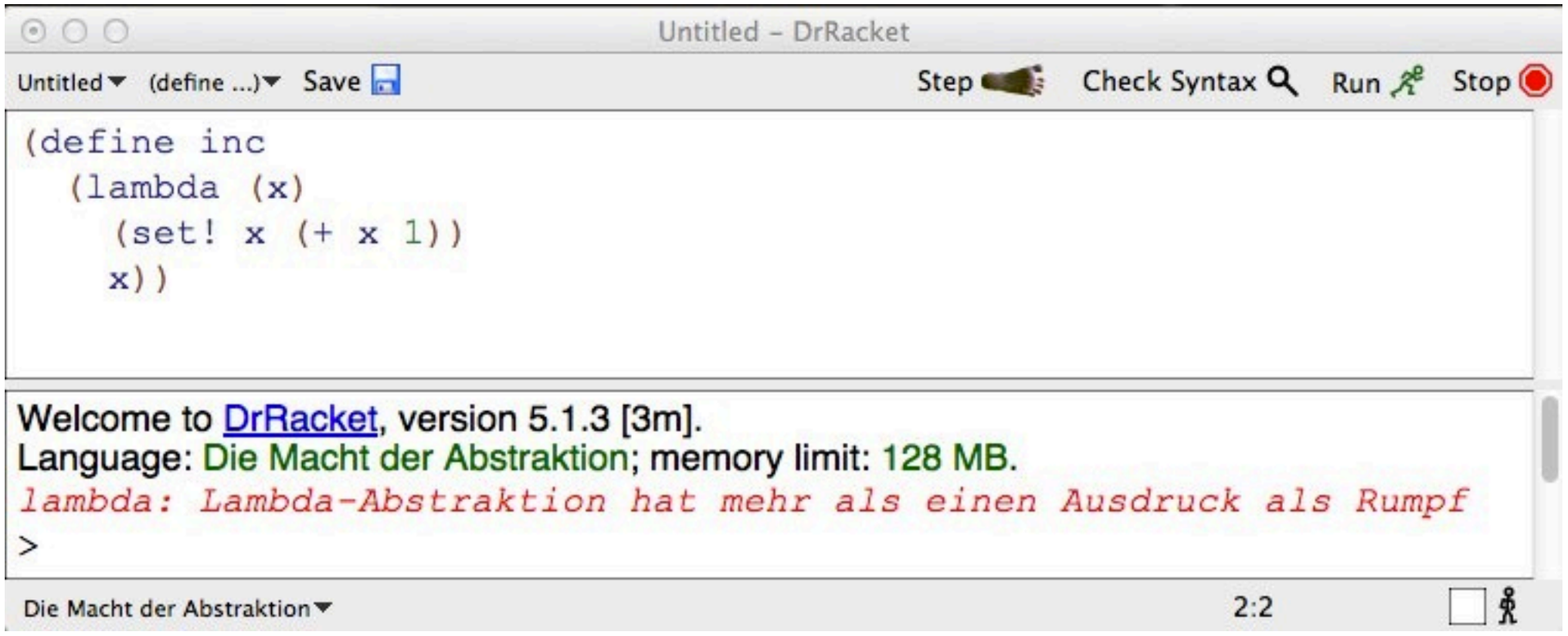
Definition

```
(define point-in-shape?
  (lambda (p s)
    (cond
      ((circle? s)
       (<= (distance p (circle-center s))
           (circle-radius s)))
      ((square? s)
       (and (>= (point-x p) (point-x (square-corner s)))
            (<= (point-x p) (+ (point-x (square-corner s))
                               (square-size s)))
            (>= (point-y p) (point-y (square-corner s)))
            (<= (point-y p) (+ (point-y (square-corner s))
                               (square-size s)))))
      ((overlay? s)
       (or (point-in-shape? p (overlay-top-shape s))
```

Refinement

```
(define point-in-shape?
  (lambda (p s)
    (cond
      ((circle? s)
       (<= (distance p (circle-center s))
           (circle-radius s)))
      ((square? s)
       (let ((corner (square-corner s)))
         (let ((cx (point-x corner))
               (cy (point-y corner))
               (size (square-size s))
               (x (point-x p))
               (y (point-y p)))
           (and (>= x cx)
                (<= x (+ cx size))
                (>= y cy)
                (<= y (+ cy size))))))
      ((overlay? s)
       (or (point-in-shape? p (overlay-top-shape s))
           (point-in-shape? p (overlay-bot-shape s)))))))
```

Enforcement

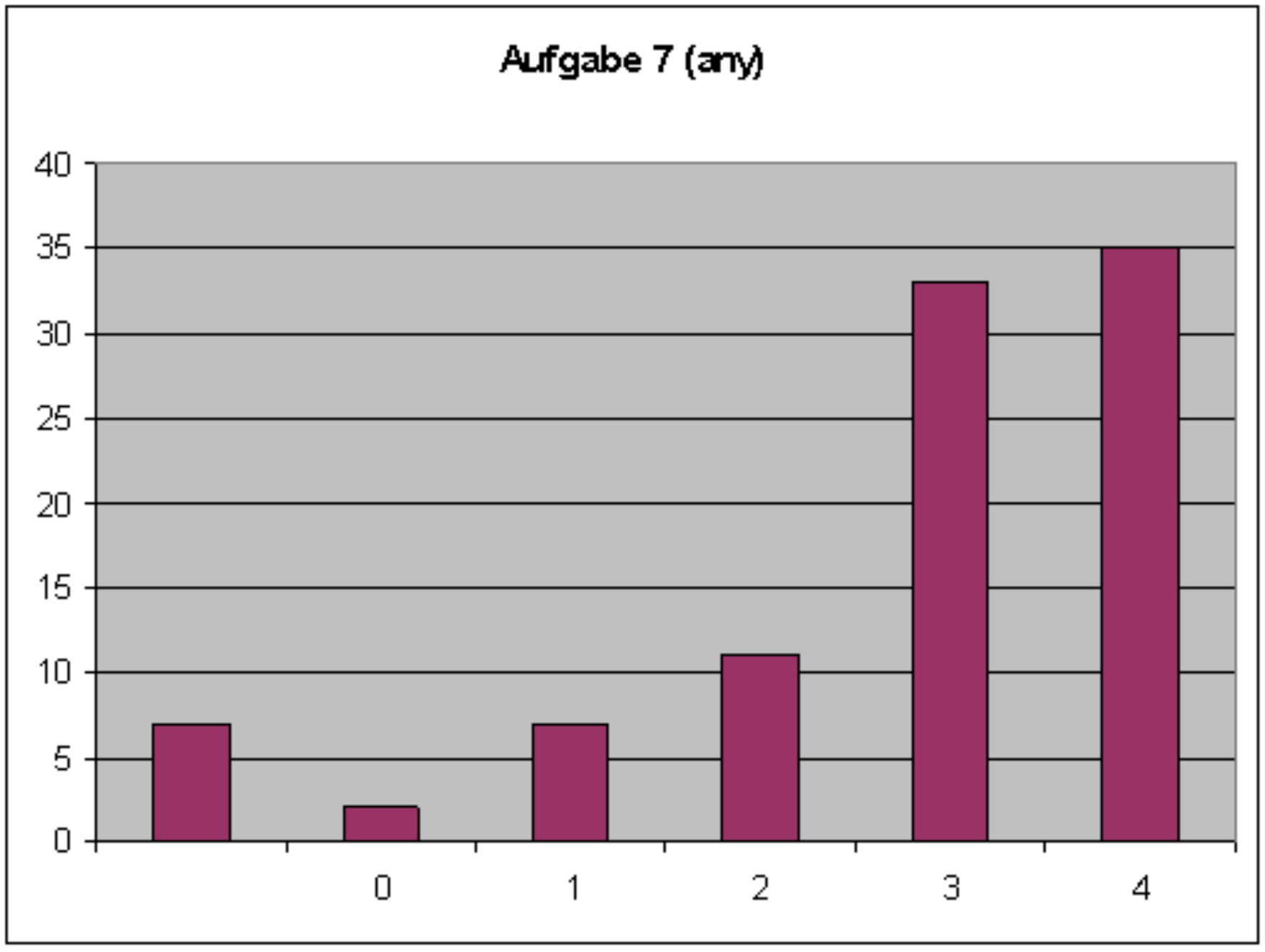


Enforcement

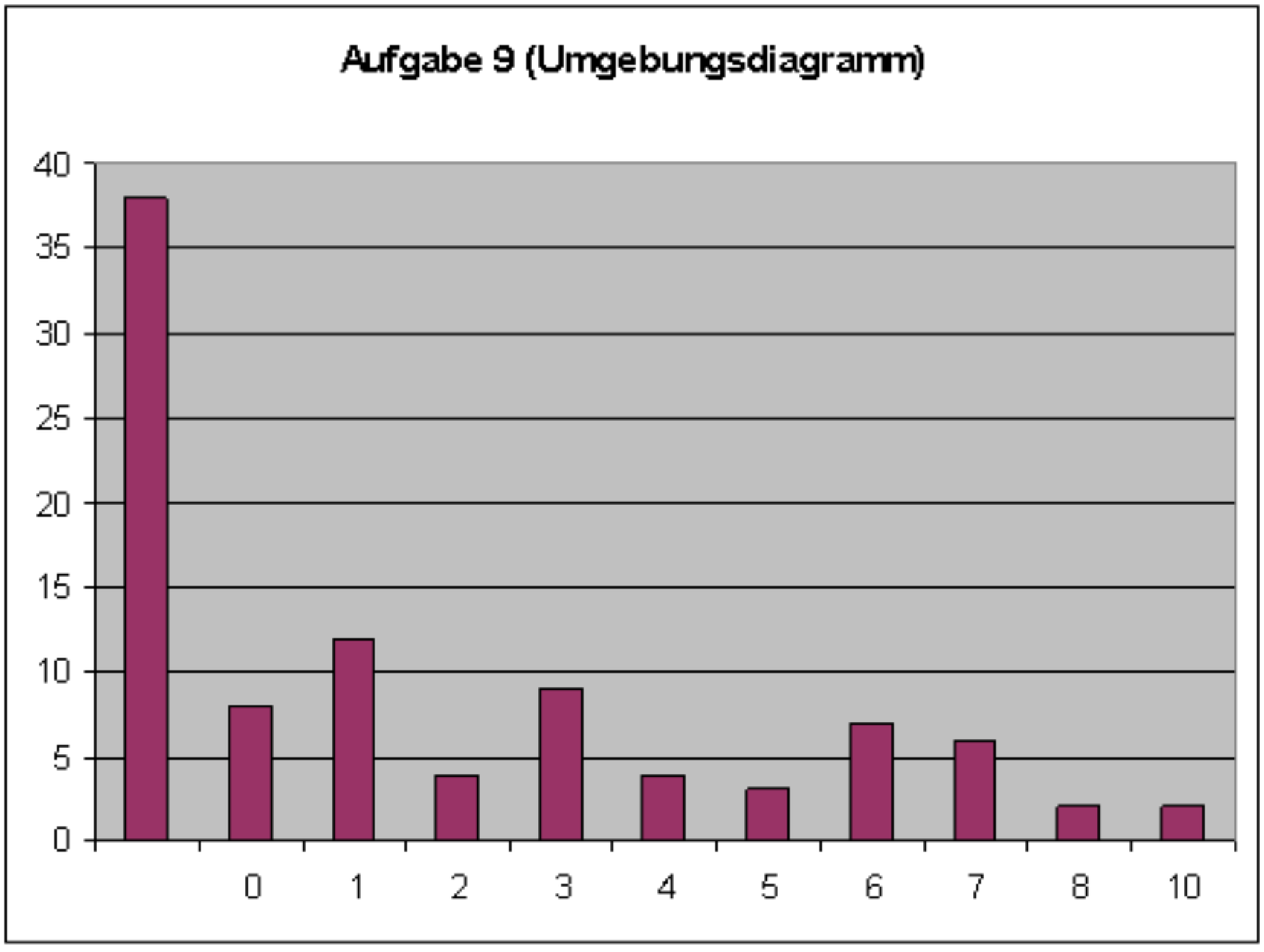
Enforcement

U   ?

Measure

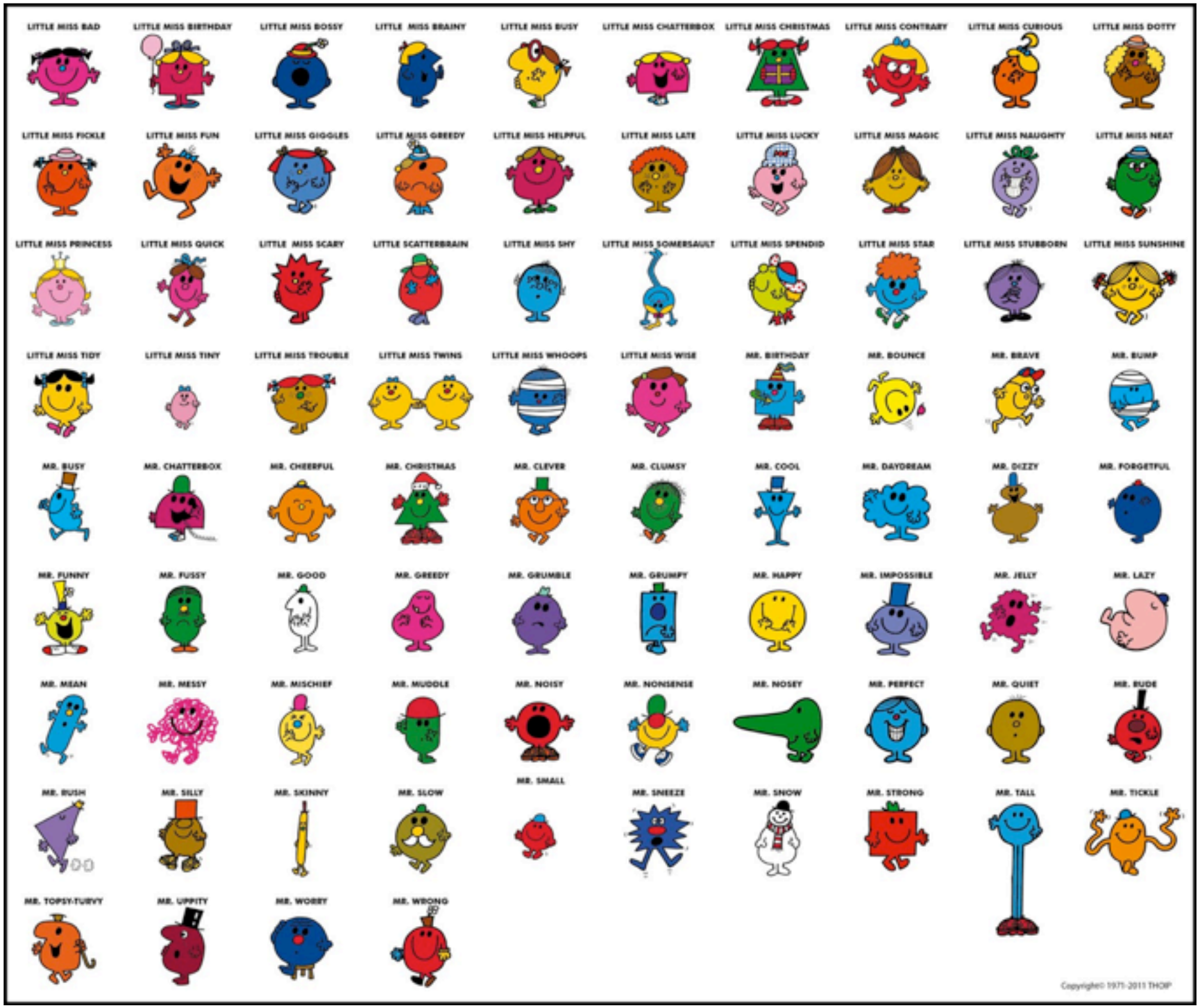


Observe & Measure



Form

How Many Forms?



How Many Forms?

Scheme

... is our business!

Practice

So Why Again Is Scheme Important?

Signature violations

list-dmda.scm – DrRacket

list-dmda.scm ▾ (define ...) ▾ Save | Step | Check Syntax | Run | Stop

```
(: foo-list (natural -> (list number)))
(define foo-list
  (lambda (n)
    (cond
      ((= n 0) empty)
      ((> n 0) (cons "foo" (foo-list (- n 1)))))))

(check-expect (foo-list 0) empty)
(check-expect (foo-list 2)
              (list "foo" "foo"))
```

```
Ran 2 tests.
All tests passed!

2 signature violations.


Signature violations:
                    got "foo" in list-dmda.scm, line 6, column 27 , signature
                    to blame: procedure in list-dmda.scm, line 3, column 2
                    got "foo" in list-dmda.scm, line 9, column 14 , signature
                    to blame: procedure in list-dmda.scm, line 3, column 2
```

Hide | Undock

Die Macht der Abstraktion ▾ 3:2

Properties

```
(check-property
 (for-all ((a number)
           (b number))
   (= (+ a b) (+ b a))))
```

Properties

```
(: commutativity
   ((%a %a -> %b) signature -> property))
(define commutativity
  (lambda (op sig)
    (for-all ((a sig)
              (b sig))
      (expect (op a b) (op b a)))))
```

Images


frogger.rkt – DrRacket
frogger.rkt
(define ...)
Step
Check Syntax
Run
Stop
(define field
)
; Grafiken für den Frosch
(define frog-up)
(define frog-down)
(define frog-left)
(define frog-right)
; Grafiken für die Autos und Trucks
(define car1)
(define car2)
(define car3)
(define car4)
(define car5)
Die Macht der Abstraktion custom
18:0

Why Not Start With Types?

```
data Tool1 = ...

data ToolState1 = ...

data Tool2 = ...

data ToolState2 = ...

data Tool = Tool1 | Tool2

data ToolState = ToolState1 | ToolState2
```

Summary

- Don’t love Scheme.
- Your students don’t have to love you.
- Only program what you can explain.
- Observe & measure.
- Kill your darlings.
- Fall in love with Scheme all over.

Collaboration Record