

Growing Up Software Development

Michael Sperber
@sperbsen

All Grown Up Now

- graduated U Tübingen 1994
- PhD 2001
- wrote some books
- freelance developer 2003-2010
- now: CEO, Active Group GmbH
- software from social pedagogy to semiconductor fabrication



Flick
Lambsdorff
DIE ANKLAGE

DAS GROSSE GESCHÄFT



COMPUTER IM KINDERZIMMER

SPIEGEL *Titel*



Kinder im Computer-Laden*: „Wir lassen die spielen, das sind nämlich unsere Kunden“

„Computer – das ist wie eine Sucht“

Das Geschäft mit dem neuesten Spielzeug der Elektronikindustrie kommt kurz vor Weihnachten in Deutschland erstmals so richtig in Schwung. Mehr als 200 000 Heimcomputer sollen dieses Jahr verkauft werden. Für

Jugendliche und Kinder ist der Umgang mit den Geräten, die vielen Erwachsenen noch unheimlich sind, eine der wichtigsten Freizeitbeschäftigungen geworden. „Der Durchbruch“, meldet die Branche, „hat sich vollzogen.“

1983

Das „Computer-Centrum“ im Karstadt-Kaufhaus an der Münchner Fußgängerzone steht wie eine Bastion in

Wohlgefallig betrachten die Verkäufer das bunte Getümmel in dem eleganten Centrum. Wir lassen die hier spielen“

der für das „Computer-Center“ zuständige Abteilungsleiter Peter Hinz, „wurden weit übertroffen.“

1986



Computer Science Club Programs An Exciting Year

The Computer Science Club is open to any student attending Radford. It helps students to understand the operation and programming of computers. This club also assists members in college planning.

Besides club meetings, there was a computer science class, which was taught by President **Michael Sperber**. Michael stated "It is not a normal class. Everyone is very active in it."

Other officers were Vice-president **Andre Stechert**, Secretary **Johanna Monsour** and Treasurer **Matt Labruyere**. **Mr. Ginoza** served as the club advisor.

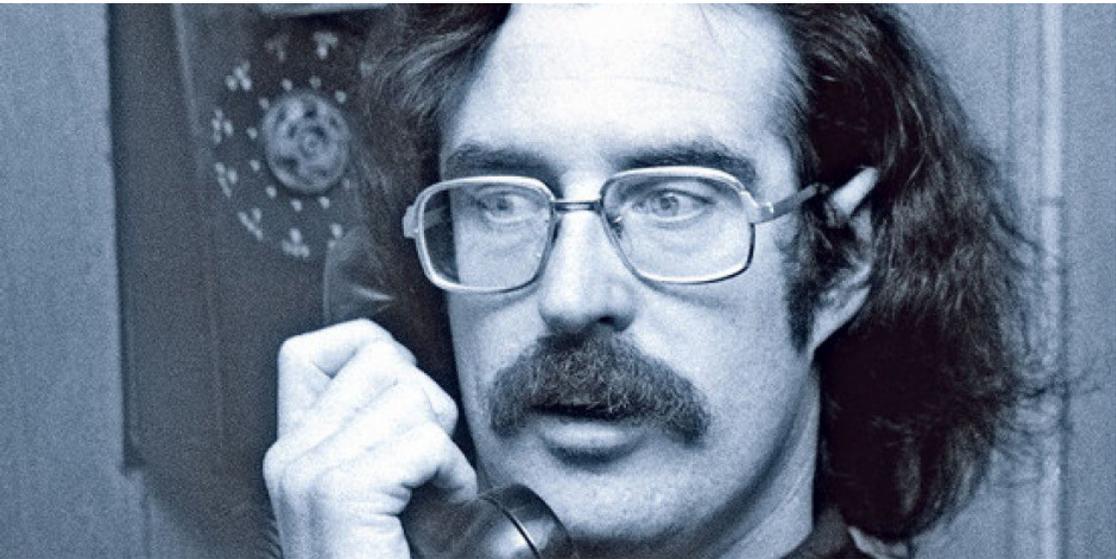


1988

Me, Teaching

- AP Computer Science 1987/1988
Radford High School
- Intro Programming for CS Students 1999-2011
University of Tübingen
- Intro Programming for Humanities Majors 1997-1999
- Training for Active Group 2012-
- (coworkers, friends, children)

Steven Levy:
Hackers -
Heroes of the Computer Revolution,
Dell Publishing, 1984, 1994.



The former "Captain Crunch" was in a bad way. Apparently certain authorities had objected to his willingness to share phone company secrets with anyone who bothered to ask; FBI agents trailed him and, according to his accounts of the incident, planted an informer who talked him into a blue-box escapade while agents waited to bust him. For this second conviction, he was sentenced to a brief jail term, and incarceration did not agree with the normally contentious Captain, a person taken to screaming like a six-foot-tall hyena if someone lit a cigarette twenty feet away from him. After his release, he needed legitimate work badly, and Woz got him hired as a consultant, designing a telephone interface board, something that would plug into one of the Apple's expansion slots to allow you to connect the phone to your computer.

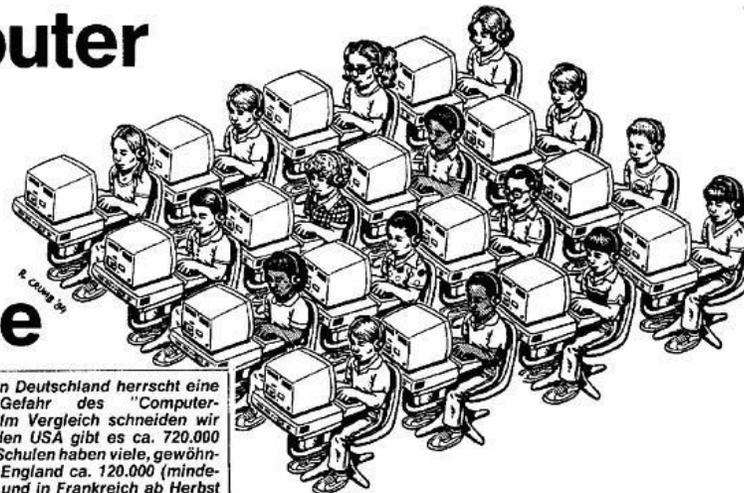
Draper happily worked on the board. The people at Apple were amused by his programming style, which alternated bursts of brilliance with bizarre pedantic detours. Draper was a "defensive" programmer. Chris Espinosa, who had the unenviable task of trying to keep an eye on the unpredictable Captain, would later explain: "Say you're writing a program and you discover you've done something wrong, like every time you try to use the program, a button pops up. Most programmers go in, analyze their program, find out what causes the button to pop up and cure it so it doesn't do that. Draper would go in and code around the button so when the bug occurs, the program knows it's made an error and fixes it, rather than avoiding the error in the first place. The joke is, if Draper were writing math routines for addition and he came up with the answer $2 + 2 = 5$, he would put a clause in the program, if $2 + 2 = 5$, then that answer is 4. That's generally the way he writes programs."

But while the hackers at Apple were amused that the strange style of John Draper was turning out a featureful product, the people in charge of the business end of Apple were not. The business end of Draper's business was not what they needed. Apple



1985

Computer und Schule



Fast alle sind sich einig: in Deutschland herrscht eine "Bildungskrise". Die Gefahr des "Computer-Analphabetismus" droht. Im Vergleich schneiden wir wirklich nicht gut ab. In den USA gibt es ca. 720.000 Rechner in Schulen (teure Schulen haben viele, gewöhnliche wenige oder keine), England ca. 120.000 (mindestens 2-3 in jeder Schule) und in Frankreich ab Herbst '85 120.000 in Schulen und Unis.

In der Bundesrepublik sollen es 1985 100.000 Comps werden. Neben dem Gerangel der Hersteller um Verkaufszahlen an Schulen treten große Probleme bei der Ausbildung am Computer auf. Zum einen fehlen die Lehrer, zum anderen ist unklar, was überhaupt gelernt werden soll. Es gibt kaum ausgebildete "Computerlehrer". Die meisten haben sich autodidaktisch oder in Fortbildungskursen vorbereitet. Ihr Wissen geben sie meist lehrplanlos weiter, oft auf Geräten, mit denen sinnvolles Arbeiten kaum möglich ist. Es gibt noch den Homo Freakus, der entweder schon seit Jahren mit Computern und Artverwandtem arbeitet und so mit großem Vorwissen in den Unterricht geht oder der sich zusammen mit seinen Schülern in die Materie einarbeitet. Letzterer benötigt eine gesunde Portion Selbstvertrauen. Sich vor eine Klasse zu stellen und zugeben zu müssen, erheblich weniger zu wissen als die Schüler ist nicht ganz einfach. Der Typ, der genau weiß, daß die grölende Masse mehr weiß als er selber, und trotzdem so tut, als wenn er mehr wüßte, muß scheitern, denn eine Klasse hat ein sehr feines Gespür für die Fähigkeiten der Lehrer. Noch dazu in der Situation, in der gleichzeitig in den Heimen der Kids eine technologische Revolution abläuft. Entweder bekommen die Gören den Computer zu Weihnachten, Geburtstag etc. oder Vater entdeckt das Kind im Manne und legt sich einen Rechner zu, auf dem er nicht arbeiten kann, bis sein 10-jähriger Sohn es ihm beigebracht hat.

Zurück zur Schule. Noch wird im Informatikunterricht eine Programmiersprache gelehrt (BASIC, Pascal oder Elan). Da treten die ersten Probleme auf. Die Gelehrten reißen sich die Haare aus im Streit um die "richtige" Sprache. Spaghetti-BASIC läuft auf fast jeder Maschine vom besseren Taschenrechner bis hin zu Großrechnern wie VAX und Cray. Pascal ist so ordentlich deutsch und informatikgerecht. Elan, die Sprache für die Schule, ist DIE die einzig wahre? LOGO, die Sprache für das Hackerbaby ab drei Monate! Oder Forth, die Programmiersprache für individuelle Compilererweiterungen ohne Baugenehmigung? Mit der Sprache C geht alles fast überall schnell und übersichtlich, meint die TUNIX-Fraktion. Glaubt man einem berühmten Hamburger Informatik-Professor, so kann man fast alles unterrichten, bloß BASIC nicht. Wer das gelernt habe, könne für den Rest seines Lebens kein vernünftiges Programm mehr schreiben.

So schnell wird sich dieser Streit nicht legen. Unabhängig davon ist zu bedenken: Diplomatenkinder an Europaschulen wie in Brüssel haben von klein auf Computerunterricht. Grund: Sonst wären die Engländer dort gegenüber den Kindern zu Hause benachteiligt, siehe oben. Bei uns ist die Diskussion zu Computerunterricht ein paar Jahre zurück: in der Oberstufe.

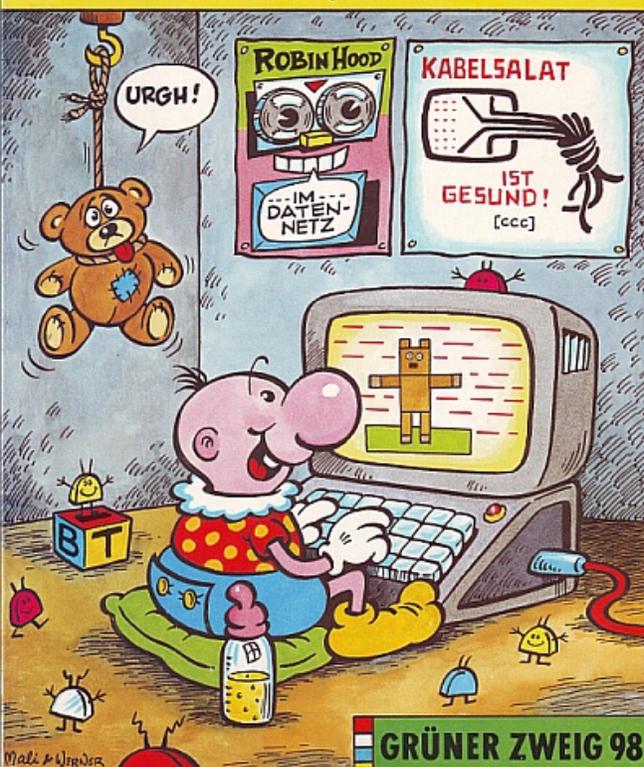
Bevor Computerwissen vermittelt wird, sollte Maschineschreiben gelehrt werden, dabei sind Computer praktische Helfer. Mit Einfinger-Adlersuchsystem dauert auch das Schreiben eines Vierzeilenprogramms zu lange. Die Schüler müssen sowieso – wegen des unleserlichen Gekrakels namens Schrift – ihre Referate mit der Schreibmaschine schreiben. Da bietet sich ein Textprogramm an. Damit ist es leichter, eine wissenschaftliche Arbeitsweise zu lernen. Dazu gehört nunmal das Erstellen und Ändern von Texten, Korrekturen, Fußnoten etc. als ein Stück Vorbereitung auf die Uni. Stoßen Schüler nach der Entlassung aus der Schule auf eine offene Stelle (Wunder), werden sie in der Regel mit Rechnern konfrontiert. Zu etwa 90-95% müssen sie dann mit Programmen von der Stange arbeiten, d. h. mit Standardsoftware. Was soll dann das Erlernen von Computerfachchinesisch, verbunden mit einem Abschalteneffekt bei allen Nichtmathematikern?

Eine Möglichkeit wäre es, nach einer kurzen Einführung in die grundlegende Bedienung und Funktionsweise von Comps, sich mit dem Erlernen von Standardsoftware zu beschäftigen: wie bediene ich Wordstar, wie gehe ich mit Multiplan um, wie baue ich meine dBase-Adressdatei...

Dazu ist die intime Kenntnis einer Programmiersprache nicht notwendig! Auch im Gemeinschaftskundeunterricht sind automatische Datenverarbeitung und Medien wichtige Themen, die leider von den wenigsten Lehrern verstanden werden. Die Gesellschaft verändert sich durch die neuen Techniken einfach zu schnell.

Extremisten wollen Comps in alle Fächer hineinpressen. Sinnvoll ist der Einsatz im naturwissenschaftlichen Unterricht zum Beispiel bei der Darstellung von Funktionen, Vektoren, Ebenen und Räumen im Mathe-Unterricht. Bildschirmdarstellung kontra Kurvenlineal entspricht Taschenrechner kontra Rechenschieber und Logarithmentabelle. Per Programm lassen sich leichter Änderungen an Kurvenzügen vornehmen als mit dem





zu, auf dem er nicht arbeiten kann, bis sein 10-jähriger Sohn es ihm beigebracht hat.

Zurück zur Schule: Noch wird im Informatikunterricht eine Programmiersprache gelehrt (BASIC, Pascal oder Elan). Da treten die ersten Probleme auf. Die Gelehrten reißen sich die Haare aus im Streit um die "richtige" Sprache. **Spaghetti-BASIC** läuft auf fast jeder Maschine vom besseren Taschenrechner bis hin zu Großrechnern wie VAX und Cray. Pascal ist so ordentlich deutsch und informatikgerecht. Elan, die Sprache für die Schule, ist DIE die einzig wahre? LOGO, die Sprache für das Hackerbaby ab drei Monate! Oder Forth, die Programmiersprache für individuelle Compilererweiterungen ohne Baugenehmigung? Mit der Sprache C geht fast alles fast überall schnell und übersichtlich, meint die TUNIX-Fraktion. Glaubt man einem berüchtigten Hamburger Informatik-Professor, so kann man fast alles unterrichten, bloß BASIC nicht. Wer das gelernt habe, könne für den Rest seines Lebens kein vernünftiges Programm mehr schreiben.



Mirai Botnet!

DDoS-For-Hire Service

Country	Index	# of servers
🇺🇸 United States	West	33,400 (17)
🇲🇽 Mexico	West	20,000 (10)

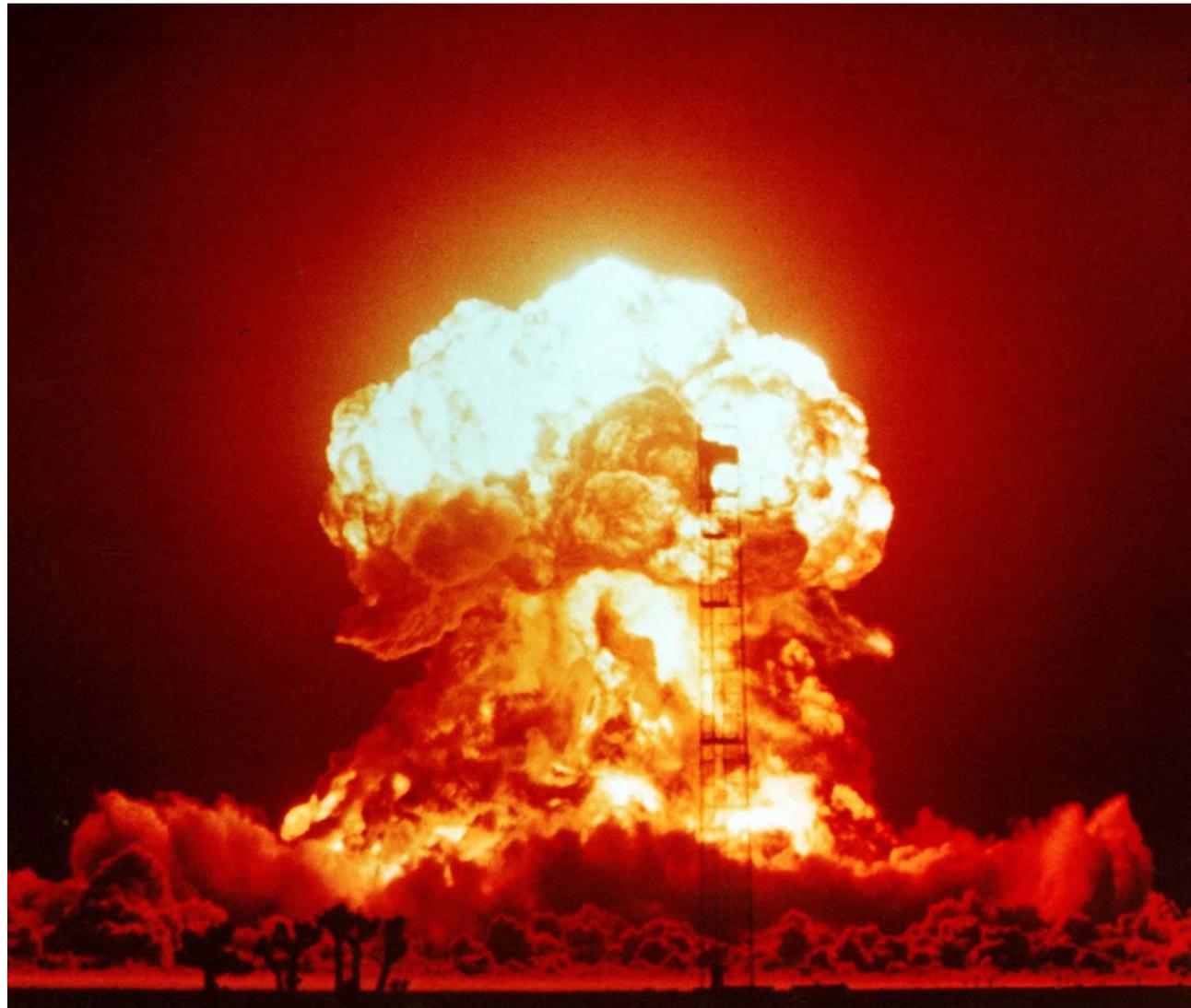
HeartBleed



<http://heartbleed.com/>

EQUIFAX

CloudBleed



Btx ist sicher!

Computer-Club profitiert vom Leichtsinne einer Sparkasse

Da hatte doch vor kurzem eine Gruppe junger Leute in Hamburg, sie nennt sich "Chaos Computer Club" (CCC) einen spektakulären Auftritt. Die Hamburger Computer-Freaks (oder "Hacker") hatten nach ihren Angaben zufällig sowohl die geheime Anschlußkennung als auch das persönliche Kennwort eines Btx-Teilnehmers, der Hamburger Sparkasse, beim Eingeben von Btx-Seiten entdeckt. Nun meldeten sich die "Computer-Chaoten" gegenüber der Post als "Haspa" und riefen ihre eigene, kostenpflichtige Btx-Seite ab, bis die "Haspa" beim "CCC" mit 135 000 Mark in der Kreide stand.

Im ersten Augenblick sind wir, aber auch alle Medien, dem CCC auf den Leim gegangen. Ja wir wollten ihm schon fast gratulieren, daß er für uns einen Fehler im System entdeckt hätte.

Es gab zwar einen Programmfehler der bereits behoben ist, aber "in gar keinem Fall", so unsere Btx-Experten hätten jedoch die Anschlußkennungen und das persönliche Kennwort gleichzeitig auf dem Schirm des Chaos-Clubs erscheinen können! Beide werden systembedingt in verschiedenen Bereichen gespeichert und können nie gemeinsam auftreten! Notwendig ist natürlich, daß die Btx-Teilnehmer ihre Zugangskennungen sorgfältig verwahren. Wenn nicht, dann könnten sie auch gleich ihre Scheckkarte und Schecks offen herumliegen lassen.

Unsere Experten sind sicher, daß der CCC die Anschlußkennung einer freizügig geschalteten mobilen Btx-Station und das persönliche Kennwort der Haspa "ausgespäht" hat.

Übrigens, an das Geld der Haspa und ihrer Kunden wäre der CCC nie herangekommen.

Fazit: Btx ist sicher!

Heim Verlag
ATARI[®] ST

1986

Michael Sperber



Digitale Bildung



Mit BOB3 werden die technischen Grundlagen der IT vermittelt. Die Frage "Wie funktioniert das?" wird für den digitalen Bereich beantwortet, indem die Schüler genau auf die Weise programmieren lernen, wie das Computer-Experten tun. Dadurch bekommen die Schüler ein grundlegendes Verständnis dafür, wie Computer 'denken' und welche Möglichkeiten bzw. Einschränkungen sich daraus ergeben. Dadurch können sie Risiken und Potentiale von neuen, digitalen Technologien besser beurteilen und bewerten.

2017

Individuelles Lernen



Das ProgBob Tutorial setzt als didaktischen Ansatz auf einen individualisierten Lernprozess. Das ist gerade in diesem Themenbereich wichtig, da die Schüler über sehr inhomogene Voraussetzungen verfügen. Manche Schüler haben zum ersten Mal Kontakt mit Maus und Tastatur, andere Schüler bringen schon von zu Hause erste 'Programmier'-Kenntnisse mit. Um diesen Umständen gerecht zu werden können die Schüler die Lektionen in ihrem jeweils eigenen Tempo erarbeiten und im weiteren Verlauf auch nach ihren Interessen auswählen.

Gamification



Durch ein hohes Maß an 'Gamification', ohne dabei jedoch auf die notwendige Tiefe für eine realitätsnahe, textbasierte Programmierung zu verzichten, wird die Motivation der SuS auf hohem Niveau aufrecht erhalten. Das Tutorial verwendet verschiedene Arten der Wissensvermittlung, zwischen denen in den Lektionen immer wieder gewechselt wird. Quizfragen, Reproduktions- und Transferaufgaben, beobachten und ausprobieren, in den späteren Tutorials auch ganze Problemlösungen, wechseln sich immer wieder ab. Als positive Anreize bekommen die Schüler nicht nur virtuelle Medaillen und funktionierende Programme, sondern auch das Gefühl, dass sie diese Leistungen selbst erbracht haben.

Medals



```
1 #include <BOB3.h>
2
3 void setup() {
4     delay(1000);
5
6     // Zähler um eins erhöhen und wieder speichern
7     int boot_counter = recall();
8     boot_counter = boot_counter+1;
9     remember(boot_counter);
10
11     for (int i=0; i<boot_counter; i++) {
12
13
14
15
16     }
17     delay(1000);
18 }
19 }
20
21
22 void loop() {
23     // bleibt erstmal leer...
24 }
25
```

__info__**The Sentinel** - der Wächter! BOB3 soll mit dem Sentinel-Programm Dinge

bewachen können, zum Beispiel eine Packung Kekse, deine Zimmertür oder dein Smartphone...



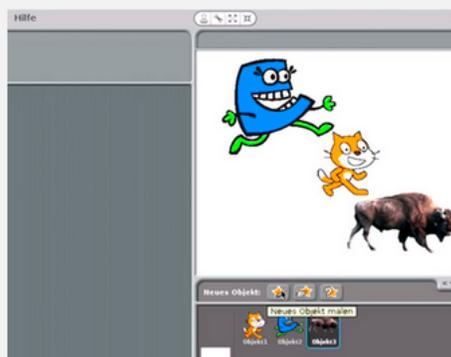
Dazu beobachtet BOB3 mit seinem IR-Sensor den zu überwachenden Bereich: Immer wenn sich der Wert des IR-Sensors zu sehr ändert, schlägt BOB3 Alarm, indem er rot mit seinen Augen blinkt!

Programmieren für Teenager

Früh übt sich. Die spielerische Herangehensweise motiviert und macht Spaß



Eine Einführung in Raspberry Pi



Programmieren lernen mit Scratch



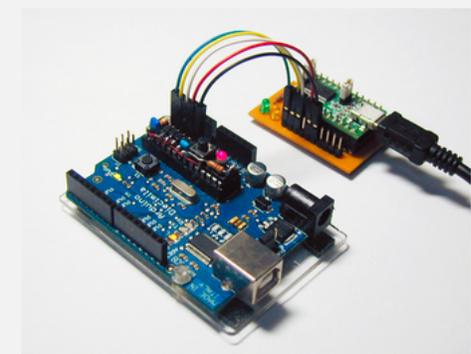
Minecraft mit Python erweitern

start-coding.de

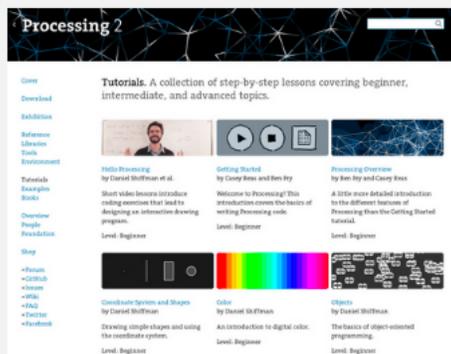
2017

Lernangebote für alle Erwachsene

Früh übt sich. Die spielerische Herangehensweise motiviert und macht Spaß



Arduino: Der einfache Einstieg in die Elektronik.



Processing: eine grafische Programmiersprache.



Experimentiere mit Sound, Musik und Code.

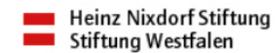


GEFÖRDERT VOM

Bundesministerium für Bildung und Forschung



Eine Initiative des Bundesministeriums für Bildung und Forschung



Heinz Nixdorf Stiftung
Stiftung Westfalen

Gesellschaft für Informatik

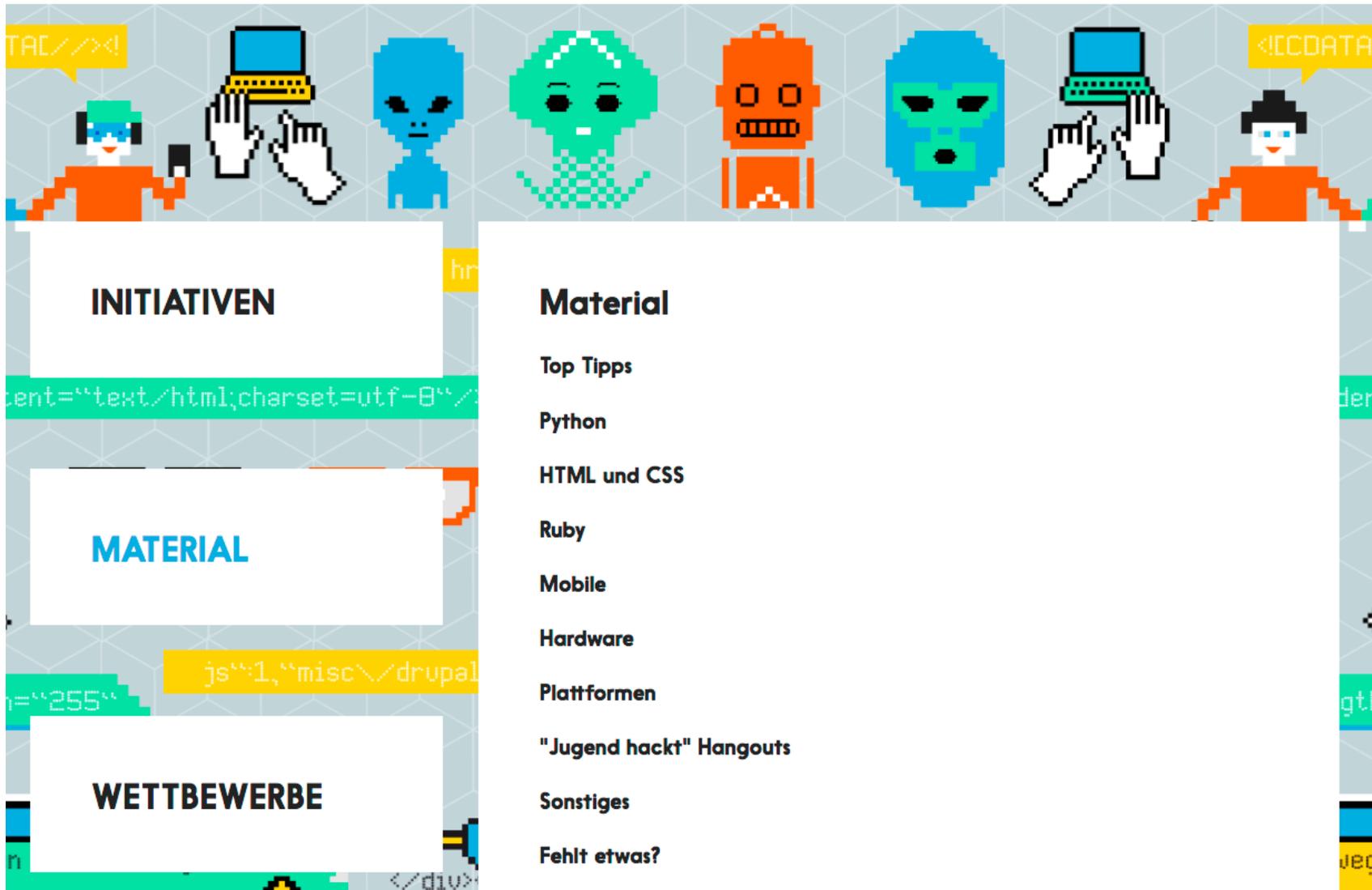




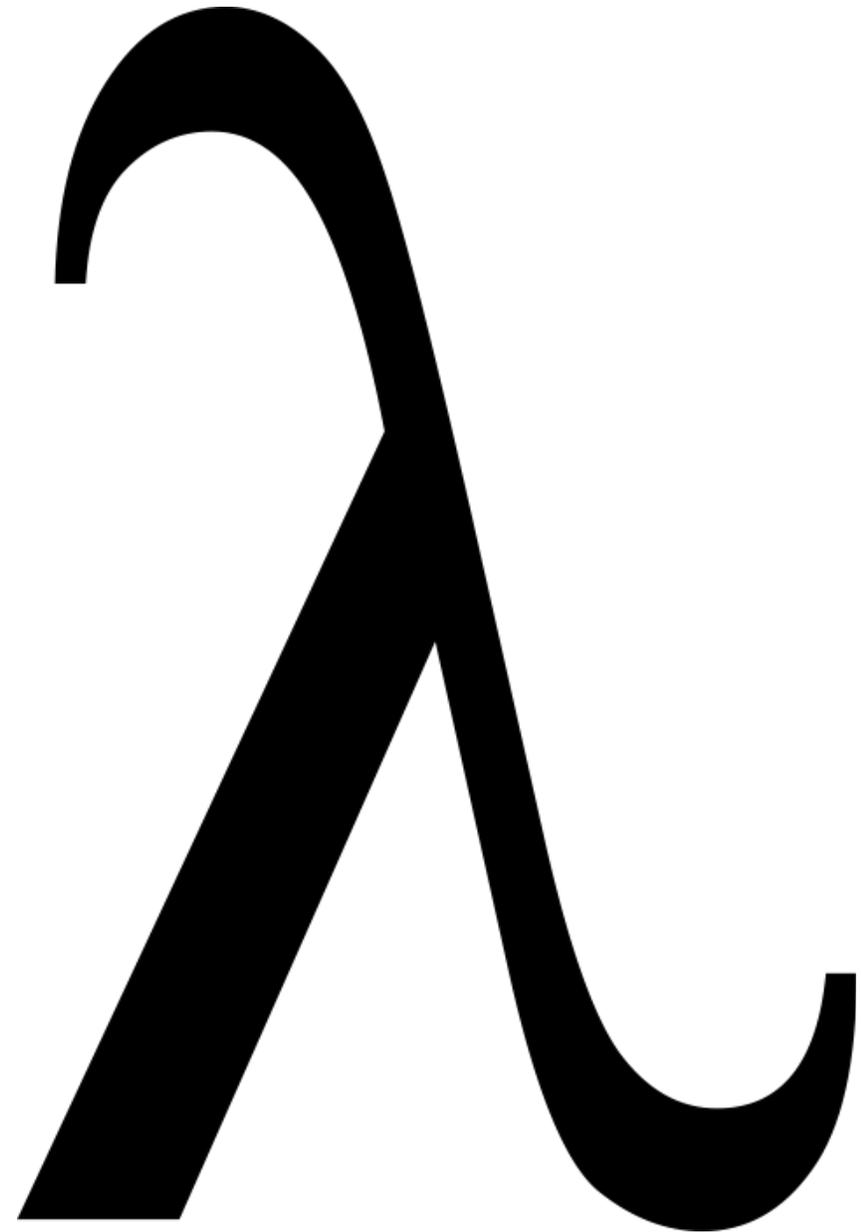
Mit Code die Welt verbessern

[WAS IST...?](#) [EVENTS](#) [HELLO WORLD](#) [PROJEKTE](#) [MITMACHEN](#) [FAQ](#) [TEAM](#) [PARTNER](#)

2017



Teaching is Easy...





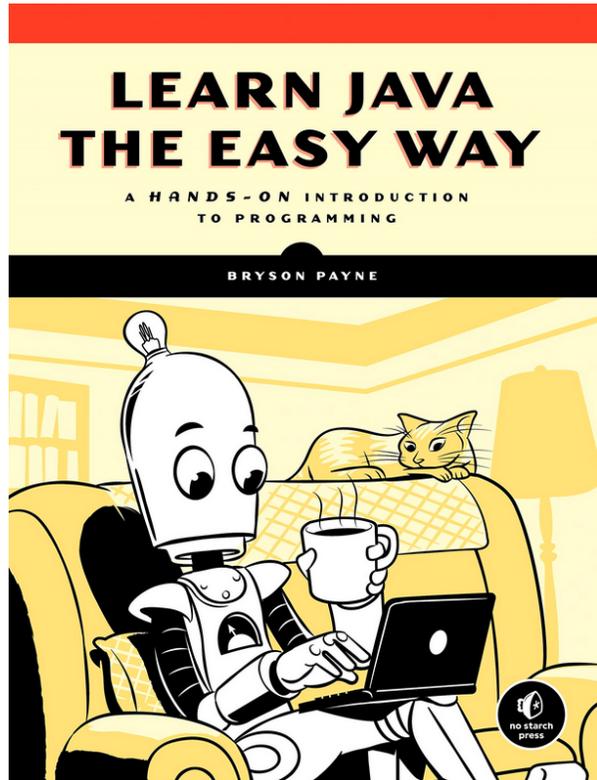
≠





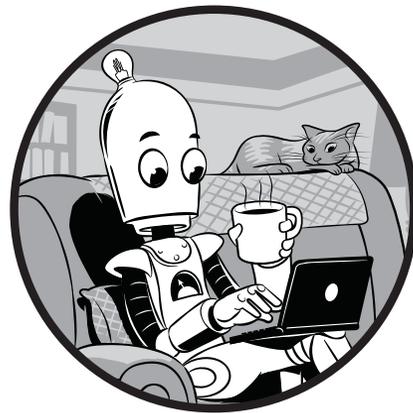
Teaching by Example

2



BUILD A HI-LO GUESSING GAME APP!

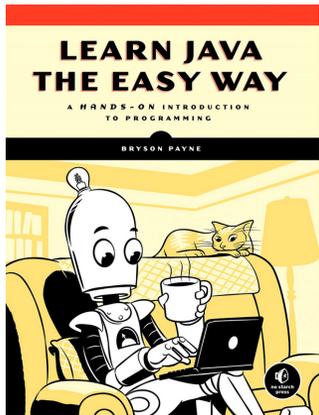
2017



Let's begin by coding a fun, playable game in Java: the Hi-Lo guessing game. We'll program this game as a *command line*

Teaching by Example

To create a while loop, we need to insert a while statement before the last three lines of code and then wrap the three lines for guessing inside a new pair of braces, as follows:



```
int guess = 0;
while(guess != theNumber)
{
    System.out.println("Guess a number between 1 and 100:");
    guess = scan.nextInt();
    System.out.println("You entered " + guess + ".");
}
}
}
```



1987

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 example
2. when a paper is analyzing a story, include its title, author, and some brief plot etc.

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 some way
 - b.
 - c.
5. concluding sentence

B. Second Area of Scope (usually one paragraph--developed in same manner as first body ¶)

- 1s - 1. a. (1.)
2. (2.)
1s - 3. b. (1.)
4. c. (2.)
- 1s - 5. c. (1.)
(2.)

9 sent.

C. Third Area of Scope

- 1.
- 2.
- 3.
4. a.
b.
c.
- 5.

III. Conclusion (usually one paragraph in length)

A. Summary (Contains a transition and goes over the main points of the paper with the idea that the end is near)

- 1.
- 2.
- 3.

B. Ending Statement (usually is some judgment/opinion about the main idea of the paper. May be more than one sentence)

- a.
- b.
- c.

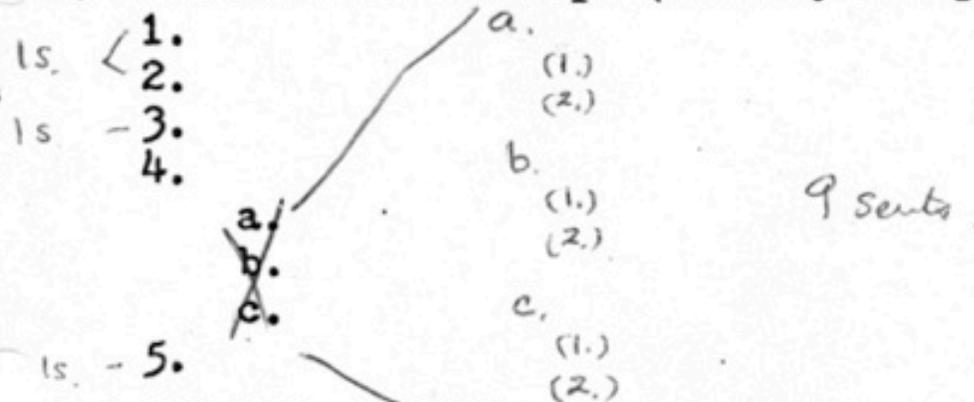
3. direction

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
 - c. {
 - etc.
5. concluding sentence

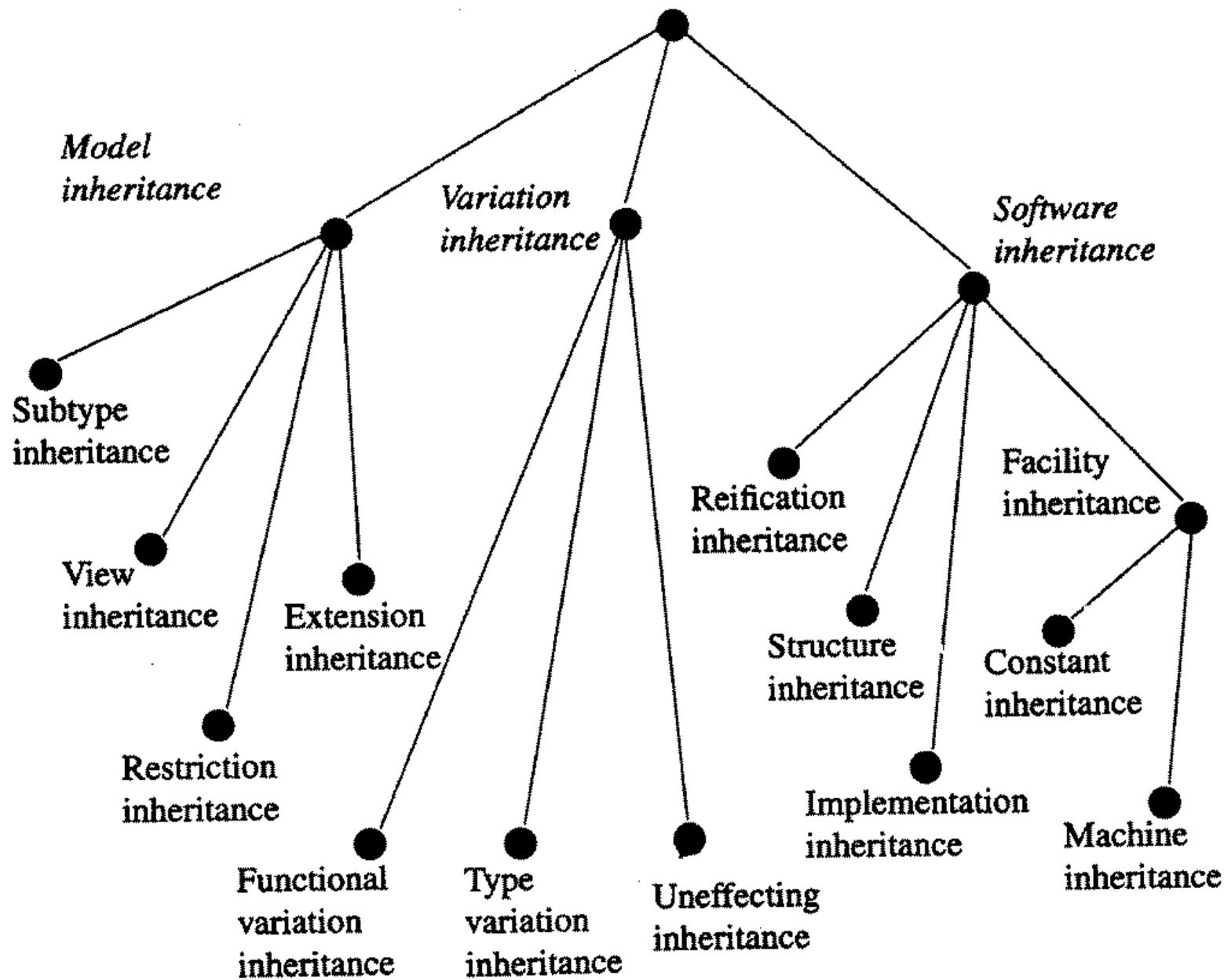
B. Second Area of Scope (usually one paragraph--developed in same manner as first body ¶)



C. Third Area of Scope

- 1.
- 2.
- 3.
- 4.

Valid use of inheritance

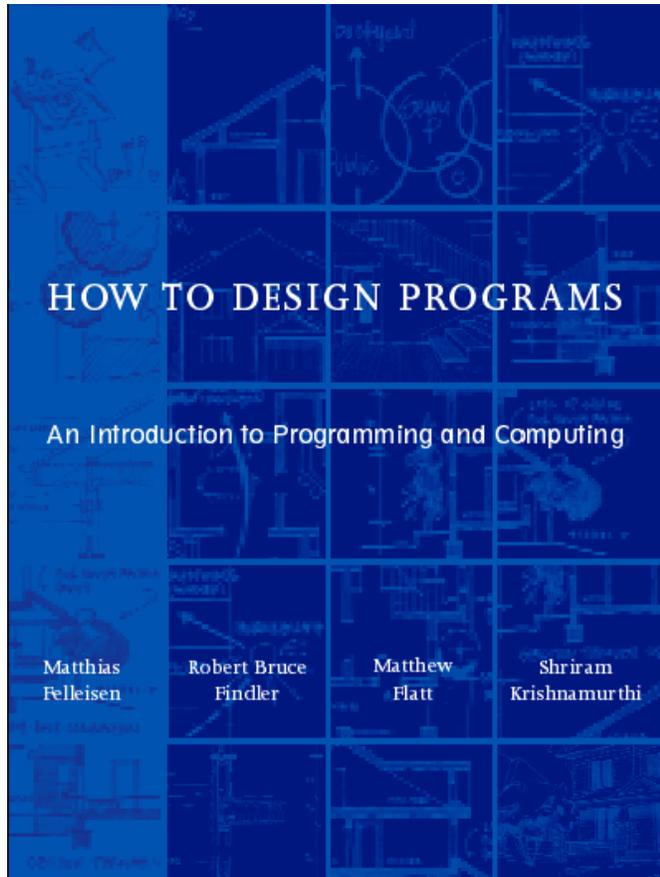


Classification of the valid categories of inheritance

Bertrand Meyer:
OO Software Construction,
2nd edition, 1997



Systematic Methods



dein
Programm

DeinProgramm ist ein Projekt zur Anfängerausbildung im Programmieren, das seit 1999 an der Universität Tübingen und anderswo entwickelt wird. Die entstandenen didaktischen Konzepte, Software und Materialien wurden in zahlreichen Anfängerausbildungen erprobt und kontinuierlich verbessert.

sys•tem•at•ic | ,sistə'matik |

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 | 'sistəmə,tist | noun

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

DESIGN

RECIPES



Armadillo

has following properties:

- alive or dead
- weight

**COMPOUND
DATA**

DrRacket

animals.rkt - DrRacket

animals.rkt (define ...) [Save]

Check Syntax [Magnifying Glass] [Checkmark] Step [Play] Run [Play] Stop [Red Square]

```
; An armadillo
; - is either dead or alive
; - has a weight
(define-record-procedures dillo
  make-dillo ; constructor
  dillo? ; predicate
  (dillo-alive? ; boolean
   dillo-weight)) ; getters, one for each field
```

Language: Die Macht der Abstraktion.
All 35 tests passed!
> |

Die Macht der Abstraktion ▾

4:2 274.23 MB [Memory Usage] [Gnome Icon]

Examples

```
(define d1 (make-dillo #t 10))  
  ; armadillo, alive, 10kg  
(define d2 (make-dillo #f 12))  
  ; armadillo, dead, 12kg
```

Signatures

```
(: make-dillo (boolean number -> dillo))
```

```
(: dillo-alive? (dillo -> boolean))
```

```
(: dillo-weight (dillo -> number))
```

Life on the Texas Highway

```
; run over an armadillo
(: run-over-dillo (dillo -> dillo))

(check-expect (run-over-dillo d1)
              (make-dillo #f 10))
(check-expect (run-over-dillo d2)
              (make-dillo #f 12))

(define run-over-dillo
  (lambda (d)
    ...))
```

Template

```
(: run-over-dillo (dillo -> dillo))
```

```
(define run-over-dillo  
  (lambda (d)  
    (make-dillo ... ..)  
    ... (dillo-alive? d) ...  
    ... (dillo-weight d) ...)))
```

Definition

```
(define run-over-dillo  
  (lambda (d)  
    (make-dillo #f  
                (dillo-weight d))))
```

Rattlesnakes

```
; A rattlesnake has the following properties:  
; - thickness  
; - length  
(define-record-procedures rattlesnake  
  make-rattlesnake  
  rattlesnake?  
  (rattlesnake-thickness  
   rattlesnake-length))  
  
(: make-rattlesnake (number number -> rattlesnake))  
(: rattlesnake-thickness (rattlesnake -> number))  
(: rattlesnake-length (rattlesnake -> number))
```

Life on the Texas Highway

```
; run over a rattlesnake
```

```
(: run-over-rattlesnake  
  (rattlesnake -> rattlesnake))
```

```
(check-expect (run-over-rattlesnake r1)  
              (make-rattlesnake 0 150))
```

```
(check-expect (run-over-rattlesnake r2)  
              (make-rattlesnake 0 200))
```

Diverse Life on the Texas Highway

An animal is one of the following:

- armadillo
- rattlesnake



**MIXED
DATA**

Diverse Life on the Texas Highway

```
(define animal  
  (signature  
    (mixed  
      dillo  
      rattlesnake) ) )
```

... and its End

```
; run over an animal
```

```
(: run-over-animal (animal -> animal))
```

```
(check-expect (run-over-animal d1)  
              (make-dillo #f 10))
```

```
(check-expect (run-over-animal r1)  
              (make-rattlesnake 0 150))
```

Function Accepting Mixed Data

```
(define run-over-animal  
  (lambda (a)  
    (cond  
      ((dillo? a)  
       ...)  
      ((rattlesnake? a)  
       ...))))
```

Function Accepting Mixed Data

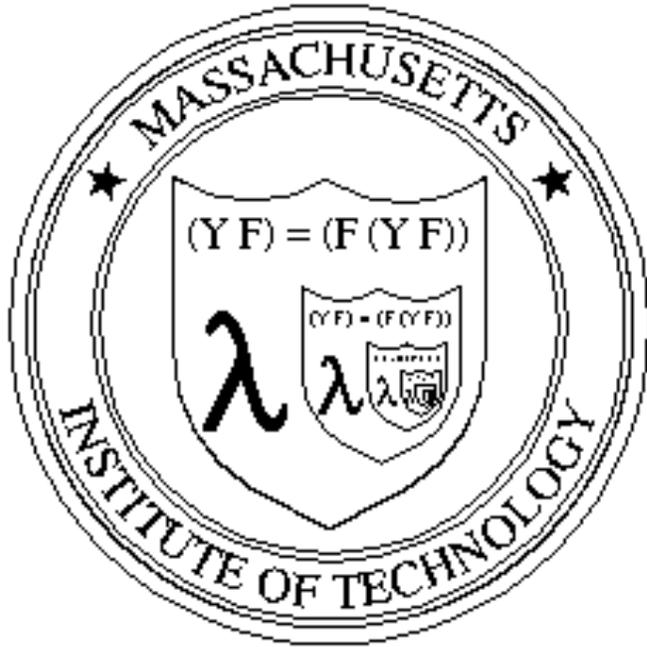
```
(define run-over-animal  
  (lambda (a)  
    (cond  
      ((dillo? a)  
       (run-over-dillo a))  
      ((rattlesnake? a)  
       (run-over-rattlesnake a))))))
```



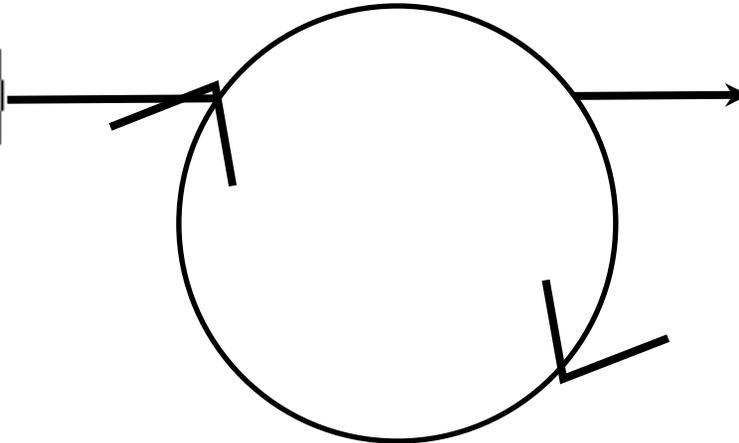


Teaching

observe
improve
repeat



arbitrary
starting point



Program by Design

- Design Recipes
- Programming Language(s) designed for learners
- Programming Environment designed for learners



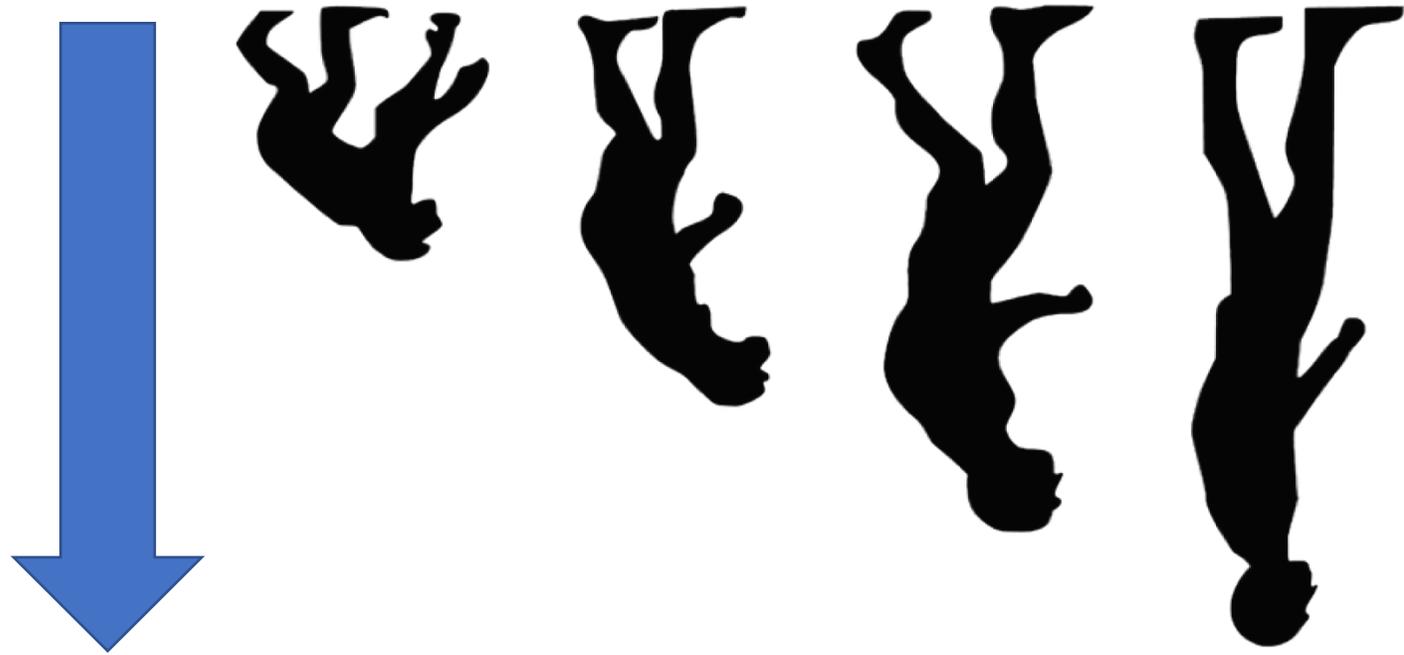
DeinProgramm ist ein Projekt zur Anfängerausbildung im Programmieren, das seit 1999 an der Universität Tübingen und anderswo entwickelt wird. Die entstandenen didaktischen Konzepte, Software und Materialien wurden in zahlreichen Anfängerausbildungen erprobt und kontinuierlich verbessert.

Program
by
Design

Program by Design combines motivated learning with our “hello world” program games, a scalable approach to this curriculum, and university levels (research, house corporate training) fundamental problems in programming languages.

Progression

- safe languages / runtimes
- functional languages
- systematic programming
- type-driven programming
- type-assisted programming







1 + 1 = 2