ETH zürich

Algorithmen und Didaktik

Programmierunterricht

Chancen und Herausforderungen

Dennis Komm

Herzlich willkommen

Roadmap

1. Vergangenheit — Informatik- und Programmierunterricht
2. Gegenwart — Informatikunterricht in der Schweiz
3. Zukunft — Eine Vision des Informatikunterrichts von morgen
4. Stolpersteine auf dem Weg — Fehlvorstellungen im Programmierunterricht

Warum unterrichten wir Informatik?

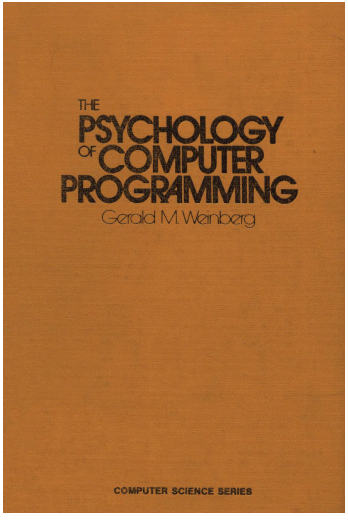
Informatik als Schulfach einzuführen ist keine neue Idee

Die Geschichte reicht mehr als 50 Jahre zurück

... mit einigen Rückschlägen (Informatik $\neq$ Anwendungskompetenzen)

Und so können wir heute noch immer von einer Pionierzeit sprechen

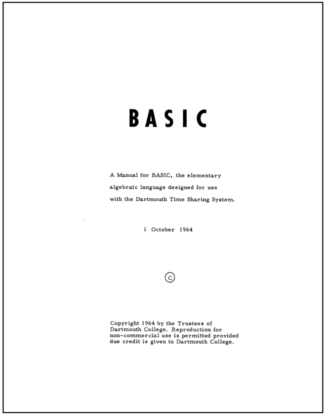
Warum unterrichten wir Informatik?



«This book has only one major purpose—to trigger the beginning of a new field of study: computer programming as a human activity, or, in short, the psychology of computer programming.»

Gerald M. Weinberg, 1971

Bereits vor Jahrzehnten kamen die ersten Programmiersprachen für die Lehre auf den Markt



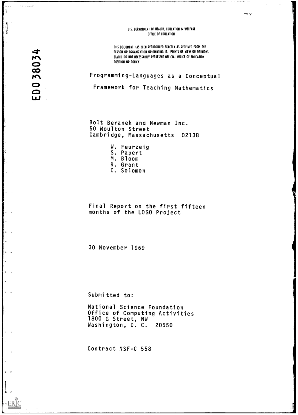
BASIC

A Manual for BASIC, the elementary algebraic language designed for use with the Dartmouth Time Sharing System.

1 October 1964

©

BASIC (1964)



ED0 38034

Programming-Languages as a Conceptual
Framework for Teaching Mathematics

Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, Massachusetts 02138

W. Feurzeig
S. Papert
M. Bloom
R. Grant
C. Solomon

Final Report on the first fifteen months of the LOGO Project

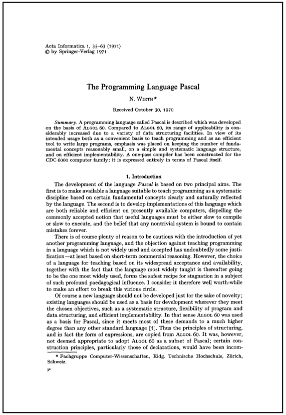
30 November 1969

Submitted to:

National Science Foundation
Office of Computing Activities
1800 G Street, NW
Washington, D. C. 20550

Contract NSF-C 558

LOGO (1967)



Acta Informatica 1, 35–63 (1971)

The Programming Language Pascal

N. Wirth*

Received October 30, 1970

Summary. A programming language called Pascal is described which was developed on the basis of Algol 60. Compared to Algol 60, its range of applicability is considerably increased due to a variety of data structuring facilities. In view of its intended usage both as a convenient basis to teach programming and as an efficient tool to write large programs, emphasis was placed on keeping the number of fundamental concepts reasonably small, on a simple and systematic language structure, and on efficient implementability. A one-pass compiler has been constructed for the CDC 6000 computer family; it is expressed entirely in terms of Pascal itself.

1. Introduction

The development of the language *Pascal* is based on two principal aims. The first is to make available a language suitable to teach programming as a systematic discipline based on certain fundamental concepts clearly and naturally reflected by the language. The second is to develop implementations of this language which are both reliable and efficient on presently available computers, dispelling the commonly accepted notion that useful languages must be either slow to compile or slow to execute, and the belief that any nontrivial system is bound to contain mistakes forever.

There is of course plenty of reason to be cautious with the introduction of yet another programming language, and the objection against teaching programming in a language which is not widely used and accepted has undoubtedly some justification—at least based on short-term commercial reasoning. However, the choice of a language for teaching based on its widespread acceptance and availability, together with the fact that the language most widely taught is thereafter going to be the one most widely used, forms the safest recipe for stagnation in a subject of such profound paedagogical influence. I consider it therefore well worth-while to make an effort to break this vicious circle.

Of course a new language should not be developed just for the sake of novelty; existing languages should be used as a basis for development wherever they meet the chosen objectives, such as a systematic structure, flexibility of program and data structuring, and efficient implementability. In that sense Algol 60 was used as a basis for Pascal, since it meets most of these demands to a much higher degree than any other standard language [1]. Thus the principles of structuring, and in fact the form of expressions, are copied from Algol 60. It was, however, not deemed appropriate to adopt Algol 60 as a subset of Pascal; certain construction principles, particularly those of declarations, would have been incom-

* Fachgruppe Computer-Wissenschaften, Eidg. Technische Hochschule, Zürich, Schweiz.

3*

PASCAL (1970)

Wo wir stehen

Die Situation an Schweizer Schulen

Die Situation in Schweizer Schulen

Volksschule KG–9

- Lehrplan 21 in deutschsprachigen Kantonen
- Modul **Medien und Informatik**

«École obligatoire» KG–9

- «Plan d'études romand» in der Romandie
- Modul **Éducation numérique**

- Formulierte Kompetenzen bzw. Objectifs d'Apprentissage für verschiedene Schulstufen
- Recht ehrgeizige Ziele bezüglich Lernzielen

Gymnasium / «Gymnase» 10–12

- Grundlagenfach bzw. «Disciplines Fondamentales» seit August 2024
- Möglichkeit eines Schwerpunktfachs bzw. «Option Spécifique»

«Kompetenzen» und «Objectifs d'Apprentissage»

MI 2.2a **[KG–2]** Die SuS können formale Anleitungen erkennen und ihnen folgen (z. B. Koch- und Backrezepte, Spiel- und Bastelanleitungen, Tanzchoreographien)

MI 2.2f **[3–6]** Die SuS können Programme mit Schleifen, bedingten Anweisungen und Parametern schreiben und testen

EN 22 **[7–9]** Die SuS können Programme mit Bedingungen und Schleifen in einer visuellen Programmiersprache erstellen und vergleichen, um einfache Probleme zu lösen

EN 32 **[7–9]** Die SuS können verschiedene Algorithmen zur Lösung desselben Problems vergleichen und die Lösungen bewerten

Rahmenlehrpläne für die gymnasiale Oberstufe

1.2.1 Die SuS können Probleme lösen, indem sie sie in Teilprobleme zerlegen

1.2.2 Die SuS können einfache Algorithmen zur Lösung von Problemen entwerfen oder sich künstlerisch mittels Programmierung ausdrücken

1.3.1 Die SuS können einen gut lesbaren und strukturierten Programmiercode schreiben und dokumentieren

3.3.1 Die SuS können verschiedene Cyber-Bedrohungen (z. B. Malware, Social Engineering) und Abwehrstrategien benennen, beschreiben und Vorsichtsmassnahmen erklären

Die Situation in Schweizer Schulen

Aber…

- «Modul» impliziert ein sehr kleines Stundengefäss
- Umsetzung nur interdisziplinär und mit gut ausgebildeten Lehrpersonen
- Insbesondere herausfordernd für Lehrpersonen der Volksschule
- Aber auch für Gymnasial-Lehrpersonen

Wo wir stehen

Die Situation an Schweizer Universitäten

Beispiel: ETH-Bachelor «Humanmedizin»

5. Sep-Dez	Notfallmedizin°	Blockwoche	Semesterleistung	2
	Pathologie	Blockw.+Sem.kurs	Semesterendprüfung	6
	Onkologie°		Semesterleistung	2
	Reproduktion°		Semesterendprüfung	4
	Früher Lebenszyklus°		Semesterendprüfung	2
	Später Lebenszyklus°		Semesterleistung	1
	Rheumatologie°		Semesterendprüfung	2
	Ethik & Recht und Kommunikation°		Semesterleistung	4
	Interprofessionelle Versorgungsketten°		Semesterleistung	3
	Ultraschall-Grundkurs°		Semesterleistung	1
	Medizintechnik I*		Semesterleistung	3
	Informatikgrundlagen für Humanmedizin*		Semesterleistung	2
6. Feb-Mai	Psychiatrie & Computational Psychiatry°	Blockwoche	Semesterleistung	2
	Psychosomatische & psychosoziale Medizin°	Blockwoche	Semesterleistung	2
	Teamarbeit, Interprofessionalität, Karriere°	Blockwoche	Semesterleistung	2
	Krankenbett°	Blockwoche	Semesterleistung	2
	Differentialdiagnostik°	Blockwoche	Semesterleistung	2
	Medizinische Bildgebung II*	Blockwoche	Semesterleistung	2
	Data Science for Medicine*	Blockwochen	Semesterleistung	4
	Medizintechnik II*	Blockwoche	Semesterleistung	2
	Translationale Tiermodelle*	Blockwoche	Semesterleistung	1
	Translationales Forschungspraktikum*	Blockwochen	Semesterleistung	8

Beispiel: ETH-Bachelor «Humanmedizin»

- Insgesamt **6 ECTS Informatik / Data Science** (zusätzlich zu Grundlagen in Statistik)
- Aktuell angenommene Informatik-Grundlagen: Keine
- Ziele im fünften Semester:
  - Einführung in Python
  - Algorithmen und Datenstrukturen «light»
  - «Computational Thinking»
- Ziele im sechsten Semester:
  - Verwendung von speziellen Python-Librarys
  - Machine-Learning-Grundlagen
  - Anwendungen in der Humanmedizin

Wohin wir wollen

Eine Vision, wie Informatikunterricht aussehen wird

Ein «Shift» nach unten

Primarschule

- Erste Programmiererfahrung
- Grundlagen wie binäre Suche etc.
- Bewusstsein für Fehlvorstellungen

Sekundarstufe I

- Programmierkonzepte
- Eine formalere Sicht auf Algorithmen
- «Computational Thinking»

Sekundarstufe II

- Fortgeschrittene Programmierkonzepte
- Algorithmen und Datenstrukturen
- Gesellschaftliche Auswirkungen

Universitäten

- Algorithmen im Fachbereich
- Data Science, Machine Learning etc.
- «Real World»-Beispiele

Ein «Shift» nach unten

Wir leben in einer **Pionierzeit**:
Schweizer Schulen haben «Mandat», Informatik vom Kindergarten bis Klasse 12 zu unterrichten

- **Spiral-Curriculum:** Führe Konzepte früh ein, iteriere mit steigender Komplexität
- Berücksichtige Alltagsbezug der SuS
- Aber fokussiere auf «first Principles» statt auf «Blackboxes»
- Unterrichte Informatik interdisziplinär (auch im Gymnasium)

Was im Weg ist

Fehlvorstellungen um Informatik

Fehlvorstellungen um Informatik

Programmierunterricht – Chancen und Herausforderungen – Dennis Komm – Fachschaftstag Bern – 3.12.2024 – 13 / 31

Was ist eine Variable?

Fehlvorstellung 1

Was ist eine Variable?

Mathematik-Aufgabe. Berechne x:

$$\begin{aligned} & 20 = 2 \cdot x + 2 \\ \iff \quad & 18 = 2 \cdot x \\ \iff \quad & 9 = x \\ \iff \quad & x = 9 \end{aligned}$$

- Aber x war eigentlich die ganze Zeit 9
- . . . auch schon in der ersten Zeile
- In diesem Sinne ist x keine «Variable» (sondern eine «Unbekannte»)

Was ist eine Variable?

Mit einer solchen Intuition des Begriffs «Variable» ergibt der folgende Code keinen Sinn

```
1 x = 1
2 print(x)
3 x = 5
4 print(x)
5 x = 10
6 print(x)
```

- In der Progammierung ist eine Variable tatsächlich «variabel»
- Dies ist der Grund für einige Fehlvorstellungen
- Mathematische Sätze sind **statische Wahrheiten**
- Objekte in der Programmierung sind zeitabhängig
- ➡ Ein Programm ist eine statische Beschreibung eines **dynamischen Prozesses**

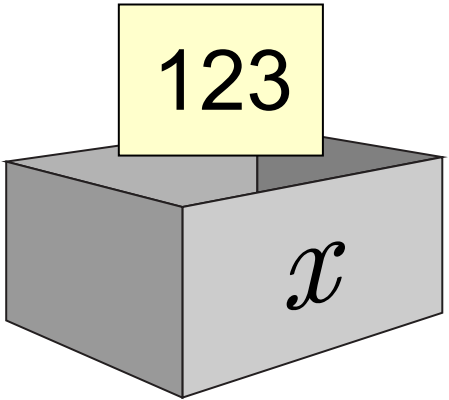
Was ist eine Variable?

SuS verstehen Variablen oft wie in der Mathematik

Was ist die Ausgabe des folgenden Programms?

```
1 x = 5
2 y = x * x
3 x = 8
4 print(8 * 8)
```

Was ist eine Variable?



Probleme

- Wie viele Zahlen passen in eine Variable?
- Warum kann keine Formel in einer Variablen gespeichert werden?
- Ist eine Variable nach Gebrauch leer?

Das Gleichheitszeichen

Fehlvorstellung 2

Welche Bedeutung hat das Gleichheitszeichen?

Mathematik-Aufgabe 2. Berechnen Sie z:

$$
\begin{aligned}
& x = 15 \\
& y = x + 6 \\
& z = 2 \cdot y \\
\Longrightarrow \quad & z = 2 \cdot (x + 6) \\
\Longrightarrow \quad & z = 2 \cdot (15 + 6) \\
\Longrightarrow \quad & z = 2 \cdot 21 \\
\Longrightarrow \quad & z = 42
\end{aligned}
$$

- In der Mathematik bedeutet «$y = x + 6$», dass y durch $x + 6$ **ersetzt** werden kann
- Wenn immer wir ein y vorfinden, können wir stattdessen $x + 6$ schreiben

Welche Bedeutung hat das Gleichheitszeichen?

Dieses Konzept auf den folgenden Code anzuwenden ergibt keinen Sinn

```
1 x = 0
2 0 = 0 + 1
3 print(0)
```

```
1 x = 0
2 x = x + 1
3 print(((x + 1) + 1) + 1)
```

Und wenn wir schon dabei sind . . .

- die Gleichung $x = x + 1$ ergibt mathematisch auch nur wenig Sinn
- und wenn $x = 0$ gilt, dann gilt auch $0 = x$, was in wiederum in der Programmierung keinen Sinn ergibt

Welche Bedeutung hat das Gleichheitszeichen?

In der Programmierung weist «=» den Wert rechts der Variablen links zu

- In der Mathematik ist $x = x + 1$ eine Gleichung
- In der Programmierung ist $x = x + 1$ eine Zuweisung
- Mathematik und Programmierung haben nicht dieselbe Sprache
- Es gibt Ausnahmen
  - Pascal (ALGOL) verwendet “:=”
  - Logo using “`make`”

Welche Bedeutung hat das Gleichheitszeichen?

. . . und das verwirrt von Zeit zu Zeit auch Expertinnen und Experten

An attempt to make a change in this way is suspicious, to say the least, so there was a lot of interest in what the attempted change was. The actual patch confirmed all suspicious; the relevant code was:

```
+       if ((options == (__WCLONE|__WALL)) && (current->uid = 0))
+                       retval = -EINVAL;
```

It looks much like a standard error check, until you notice that the code is not testing `current->uid` - it is, instead setting it to zero. A program which called `wait4()` with the given flags set would, thereafter, be running as root. This is, in other words, a classic back door.

The resulting vulnerability, had it ever made it to a deployed system, would have been a locally-exploitable hole. Some sites have said that the hole would have been susceptible to remote exploits, but that is not the case. An attacker would need to be able to run a program on the target system first.

Der «Linux Backdoor Attempt» von 2003

Schleifen

Fehlvorstellung 3

Schleifen

- Der «Lehrplan 21» stellt Schleifen vor Variablen
- Jedoch brauchen `for`- und `while`-Schleifen Variablen, um Sinn zu ergeben
- Deswegen gibt es in Logo die `repeat`-Schleife

```
repeat 4 [
  forward 100
  right 90
]
```

- Übernommen in TigerJython

```
repeat 4:
  forward(100)
  right(90)
```

- Mit dem `repeat`-Block können Schleifen schon im Kindergarten thematisiert werden

Der Superbug

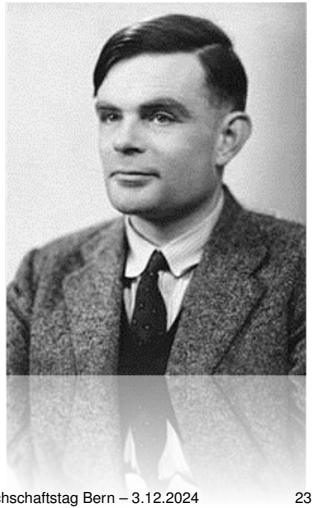
Die wahrscheinlich grösste Fehlvorstellung

Was muss ein Computer können?

. . . nicht wirklich viel

Turingmaschine [Alan Turing, 1936]

- Multiplikation ist nur wiederholte Addition
- Addition ist wiederum nur wiederholtes Erhöhen um Eins
- Wenige Grundoperationen reichen
- Turing-Vollständigkeit



Nagut, aber was kann ein Computer denn eigentlich?

- Der Computer ist eine «Blackbox», insbesondere für Anfängerinnen und Anfänger
- Es ist in keiner Weise klar, was der Computer «versteht» oder «kann»
- Anfängerinnen und Anfänger neigen dazu, mit ihm wie mit einem Menschen zu interagieren
- In Zeiten von Siri, Alexa und ChatGPT ist das nicht überraschend

Das kann in der Praxis beobachtet werden, z. B. «Natürlich enthält eine Variable mit dem Namen `maximum` einen maximalen Wert. Der Computer weiss das, weil es klar ist.»

MI 2.2e [5–6] SuS verstehen, dass ein Computer nur vordefinierte Anweisungen ausführen kann und dass ein Programm eine Abfolge von solchen Anweisungen ist.

Die «Notional Machine»

Mache die Maschine so einfach wie möglich

Ausführende Maschine ➧ konzeptuelle Maschine («Notional Machine»)

- Der Zustand der Maschine sollte sichtbar sein
  - ➧ «Glasbox» statt «Blackbox»
- Damit ist sie einfach zu beherrschen
- Damit ist sie aber auch beschränkt und kann nur wenige Probleme direkt lösen
  - ➧ Bereits das Lösen einfacher Probleme erfordert **Problemlösefähigkeiten**

Turtlegrafik

Reptilien vs. Superbugs

Turtlegrafik

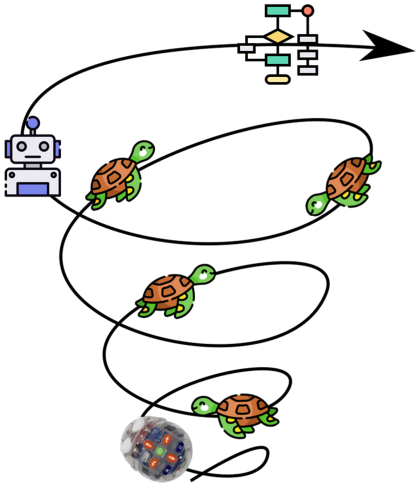
- Programmiere «die Turtle» statt des Computers
- Wenige Befehle, um sich über den Bildschirm zu bewegen und eine Linie zu zeichnen
- Wenige (implizite) Variablen beschreiben den Zustand

```
1 to rectangle
2 setpencolor blue
3 repeat 4 [
4     forward 200
5     right 90
6 ]
7 end
```

Turtlegrafik

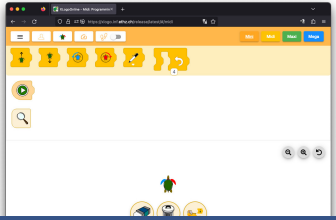
- Die Turtle ist eine greifbare «Notional Machine»
- Man kann sich einfach mit ihr identifizieren
- Die Befehle entsprechen unmittelbar beobachtbaren Aktionen, die von der Turtle ausgeführt werden, z. B. `forward`, `backward`, `left`, `right`
- Mit der Turtle entsteht eine einfache nicht-technische Sprache, z. B. das Definieren eines Befehls als «der Turtle ein neues Wort beibringen»
- Die Turtle kann während der Ausführung beobachtet werden

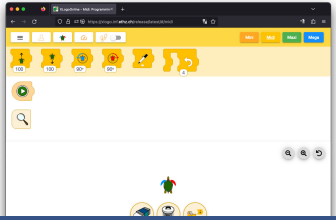
Ein Spiral-Curriculum mit der Turtle



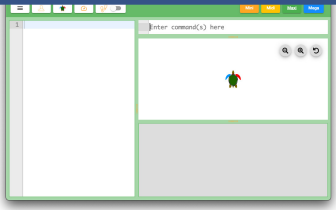
- WebTigerPython ohne Turtle
- Turtle / Roboter mit WebTigerPython
- Turtle / Roboter mit textbasiertem Logo
- Turtle / Roboter mit blockbasiertem parametrisiertem Logo
- Turtle / Roboter mit blockbasiertem Logo

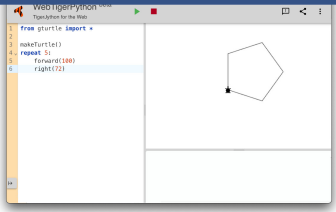
Ein Spiral-Curriculum mit der Turtle





Visit `https://xlogo.inf.ethz.ch` and `https://webtigerpython.ethz.ch`





Und dann noch etwas…

Co-Pilots und Co.

DOI:10.1145/3570220 Matt Welsh

Viewpoint

The End of Programming

The end of classical computer science is coming, and most of us are dinosaurs waiting for the meteor to hit.

CAME OF AGE in the 1980s, programming personal computers such as the Commodore VIC-20 and Apple][e at home. Going on to study computer science (CS) in college and ultimately getting a Ph.D. at Berkeley, the bulk of my professional training was rooted in what I will call "classical" CS: programming, algorithms, data structures, systems, programming languages. In Classical Computer Science, the ultimate goal is to reduce an idea to a program written by a human—source code in a language like Java or C++ or Python. Every idea in Classical CS—no matter how complex or sophisticated, from a database join algorithm to the mind-bogglingly obtuse Paxos consensus protocol—can be expressed as a human-readable, human-comprehensible program.

When I was in college in the early 1990s, we were still in the depths of the AI Winter, and AI as a field was likewise dominated by classical algorithms. My first research job at Cornell University was working with Dan Huttenlocher, a leader in the field of computer vision (and now Dean of the MIT Schwarzman College of Computing). In Huttenlocher's Ph.D.-level computer vision course in 1995 or so, we never once discussed anything resembling deep learning or neural networks—it was all classical algorithms like Canny edge detection, optical flow, and Hausdorff distances. Deep learning was in its infancy, not yet considered mainstream AI, let alone mainstream CS.

Of course, this was 30 years ago, and a lot has changed since then, but one thing that has not really changed is that CS is taught as a discipline with data structures, algorithms, and programming at its core. I am going to be amazed if in 30 years, or even 10 years, we are still approaching CS in this way. Indeed, I think CS as a field is in for a pretty major upheaval few of us are really prepared for.

Programming will be obsolete. I believe the conventional idea of "writing a program" is headed for extinction, and indeed, for all but very specialized applications, most software, as we know it, will be replaced by AI systems that are *trained* rather than *programmed*. In situations where one needs a "simple" program (after all, not everything should require a model of hundreds of billions of parameters running on a cluster of GPUs), those programs will, themselves, be generated by an AI rather than coded by hand.

I do not think this idea is crazy. No doubt the earliest pioneers of computer science, emerging from the (relatively) primitive cave of electrical engineering, stridently believed that all future computer scientists would need to command a deep understanding of semiconductors, binary arithmetic, and microprocessor design to understand software. Fast-forward to today, and I am willing to bet good money that 99% of people who are writing software have almost no clue how a CPU actually works, let alone the physics underlying transistor design. By extension, I believe the computer scientists of the future will be so far removed from the classic definitions of "software" that they would be hard-pressed to reverse a linked list or implement Quicksort. (I am not sure I remember how to implement Quicksort myself.)

AI coding assistants such as CoPilot are only scratching the surface of what I am describing. It seems totally obvious to me that *of course* all programs in the

Communications of the ACM

Jensen Huang says kids shouldn't learn to code — they should leave it up to AI

News By Mark Tyson published February 25, 2024

But this isn't the first time a tech exec has predicted the death of coding.

Comments (121)

Tom's Hardware

Co-Pilots und Co.

- Was ist die Aufgabe von Schule?
- Was ist die Aufgabe von Informatikunterricht?
- Was ist die Aufgabe von Programmierunterricht?

Vielen Dank für die Aufmerksamkeit