Informatics Europe: 19th European Computer Science Summit
AI and the Future of Informatics Education Workshop
Session 2: The impact of AI on Informatics curricula and educational practices

Should we still teach "Programming"?

Thomas R. Gross

Department Informatik
ETH Zürich

Copyright (c) Thomas Gross 2023
All rights reserved.
Disclaimer

- I do not know the answer(s).
  - Maybe more than one answer

- These comments are my personal view and not cleared with my institution or a (national) Informatics Association
  - Based on research and teaching at several institutions
    - Design and implementation of compilers, programming tools, networking and computer systems
  - Teaching the last 8 years "Introduction to Programming" to 1st semester computer science students at ETH Zurich
You must have heard or seen ...
You must have heard or seen ...
A number of minor issues

- How good/relevant are these tools for "Introduction to Programming"?

- Have we reached the end of programming?

- Do these tools make teaching programming {unnecessary | trivial}?
Programming?

«Educators, generals, dieticians, psychologists, and parents program. Armies, students, and some societies are programmed.»

Alan Perlis

(Foreword to «Structure and Interpretation of Computer Programs», H. Abelson and G. J. Sussman, 1985)
Programming?

- Programming ⊂ Software Engineering
- Software Engineering ⊂ Programming
Programming?

- Related to "Computational Thinking"
  - Permeates different groups
  - Reflection

- High-school curricula
  - Scratch, Logo and other block programming languages
  - Other languages (from Python to Java to ....)

- Data analysis for domain experts

- (University-level) programming as foundation for system design
(University-level) programming and I2P

- I2P: Introduction to Programming
- Concepts, not syntax
- Foundation for various subfields
- Must be mastered to appreciate wide range of topics (AI, networking, security, program verification, computer vision, user interfaces, ...)

11
Programming (in the context of CS education)

«Programming as universal activity» by Vinton Cerf, CACM March 2016, vol 59(3) p 7

- Analyzing problems
- Breaking them down into manageable parts
- Finding solutions
- Integrating the results
Summary I2P@ETH

- Programming as foundation for system design
- Imperative programming (w/ class-based OOP)
- Practical slant ("Programming+")
  - Testing (JUnit)
  - Debugging, Design/partitioning
  - Leverage IDE
- Optional practice problems (some new each year)
Overview

- How good/relevant are LLM tools (programmer assistants) for "Introduction to Programming"?
  - I2P as part of the ETH Computer Science B.Sc. program
  - Focus on one tool: GitHub Copilot
Copilot – friend or foe?

- General-purpose languages (e.g., Algol 60, FORTRAN IV) replaced assembly language programming
- (Compile-time checking of) Types raised programming level
- Object-oriented languages provide facilities to structure (larger) programs
- Parallel program generators avoid dealing with low-level details

How can Copilot help?
<table>
<thead>
<tr>
<th>Java</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Warmup-1</strong></td>
<td>Simple warmup problems to get started</td>
</tr>
<tr>
<td></td>
<td>(solutions available)</td>
</tr>
<tr>
<td><strong>String-1</strong></td>
<td>Basic string problems -- no loops</td>
</tr>
<tr>
<td><strong>Logic-1</strong></td>
<td>Basic boolean logic puzzles -- if else</td>
</tr>
<tr>
<td></td>
<td>&amp; &amp;</td>
</tr>
<tr>
<td><strong>String-2</strong></td>
<td>Medium String problems -- 1 loop</td>
</tr>
<tr>
<td><strong>Array-1</strong></td>
<td>Basic array problems -- no loops.</td>
</tr>
<tr>
<td><strong>Logic-2</strong></td>
<td>Medium boolean logic puzzles -- if else</td>
</tr>
<tr>
<td></td>
<td>&amp; &amp;</td>
</tr>
<tr>
<td><strong>String-3</strong></td>
<td>Harder String problems -- 2 loops</td>
</tr>
</tbody>
</table>

**New...**

| Map-1       | Basic Map get()/put(), no loops        |
| Map-2       | Maps with bulk data and loops          |
| Functional-1| Functional mapping operations on lists |
|             | with lambdas                           |
| Functional-2| Functional filtering and mapping       |
|             | operations                              |
countCode

Return the number of times that the string "code" appears anywhere in the given string, except we'll accept any letter for the 'd', so "cope" and "cooe" count.

countCode("aaacodebb") → 1
countCode("codexxcode") → 2
countCode("cozexxcope") → 2
countCode

// Return the number of times that the
// string "code" appears anywhere in the
// given string, except we'll accept any
// letter for the 'd', so "cope" and "cooe"
// count.
public static int countCode(String str){
    int count = 0;
    for(int i = 0; i < str.length() - 3; i++){
        if(str.substring(i, i + 2).equals("co") &&
           str.charAt(i + 3) == 'e'){
            count++;
        }
    }
    return count;
}
Copilot

- Picked good name for method
- Needed help with method parameter

- Summary: ✔️
Others reported similar observation

Copilot was trained on public repositories and popular collections of programming problems
  - Copilot will know the answers to these problems
  - Also textbook problems like TowersOfHanoi, Fibonacci, ....

Copilot needed sometimes help with the structure
  - Method and method parameter type(s)
  - First suggestion often a "literal parameter"
Copilot performance for other tasks

- "New" (not from a repository or textbook)
  - One task/week, with start in Week 4 or 5
    - Comes with small (sample) test suite
    - Students should write tests
  - Increasing difficulty/size
  - Optional for students but must be done within a week to be graded

- Step 1: Feed task description to Copilot
Tasks 2018

<table>
<thead>
<tr>
<th>Week</th>
<th>Size (Words)</th>
<th>Topic</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>197</td>
<td>String-Addition: loops, arrays</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>456</td>
<td>Tool Rental: classes, arrays, iteration, JUnit</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>344</td>
<td>Valleys &amp; Hills: arrays, data analysis, I/O</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>146</td>
<td>String Interleaving: recursion</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>151</td>
<td>List Reversal: references, working w/ classes</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>163</td>
<td>Class Puzzle: inheritance ((w/o \text{classes or driver}))</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>600</td>
<td>Desk Calculator: inheritance, recursion</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>302</td>
<td>Data Analysis (FIFA): ArrayList(&lt;..&gt;), Map(&lt;..&gt;)</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>194</td>
<td>Sublist Palindrome: Set(&lt;List&lt;..&gt;&gt;), exceptions</td>
<td>100</td>
</tr>
</tbody>
</table>
Copilot

Your AI pair programmer
Copilot – hints and suggestions

- Hints and interactions
  - Informal classification
  - Not a strict hierarchy
  - Identify "highest level" for hint

- Rough classification
  - Experiments by G. Ponti

- Preliminary
Hints – summary

- Many tasks require hints
  - That is expected ...

- Appropriate hints require knowledge of programming concepts
# Copilot – 2018 hints and suggestions

<table>
<thead>
<tr>
<th>Week</th>
<th>Size (Words)</th>
<th>Topic</th>
<th>Total Required</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>197</td>
<td>String-Addition: loops, arrays</td>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>456</td>
<td>Tool Rental: classes, arrays, iteration, JUnit</td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>6</td>
<td>344</td>
<td>Valleys &amp; Hills: arrays, data analysis, I/O</td>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>7</td>
<td>146</td>
<td>String Interleaving: recursion</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>151</td>
<td>List Reversal: references, working w/ classes</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>9</td>
<td>163</td>
<td>Class Puzzle: inheritance (w/o classes or driver)</td>
<td>12</td>
<td>100</td>
</tr>
<tr>
<td>10</td>
<td>600</td>
<td>Desk Calculator: inheritance, recursion</td>
<td>5</td>
<td>100</td>
</tr>
<tr>
<td>11</td>
<td>302</td>
<td>Data Analysis (FIFA): ArrayList&lt;..&gt;, Map&lt;..&gt;</td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>12</td>
<td>194</td>
<td>Sublist Palindrome: Set&lt;List&lt;..&gt;&gt;, exceptions</td>
<td>0</td>
<td>100</td>
</tr>
</tbody>
</table>
Hints – summary

- Many tasks require hints
  - That is expected ...

- Appropriate hints require knowledge of programming concepts
  - There is no free lunch
  - Unless your task description matches completely
Acknowledgments

- Michael Faes, Gaurav Parthasarathy, Felix Wolf
  - Co-authors of programming problems

- Giacomo Ponti
  - Experiments with Copilot
Discussion

- Copilot is not an automatic program generator
- Copilot seems to expect a fairly skilled pair programmer
  - May use advanced language features
  - Solution to early problem uses classes and generics
- Copilot encourages good programming practice
  - Comments, good examples in real-life setting
- Student must (still) understand unit tests, carefully read problem description
  - Text understanding crucial skill, adversary reading/analysis
  - Requirements analysis
It’s Like GPT-3 but for Code—Fun, Fast, and Full of Flaws
The end of programming?

The Premature Obituary of Programming

By Daniel M. Yellin
Communications of the ACM, February 2023, Vol. 66 No. 2, Pages 41-44
10.1145/3555367

It's the end of programming as we know it -- again
Issues outside of scope

- Use of LLM tools to support teaching
  - Task development, individual feedback, grading, ...
- Security implications
- IP issues
- ... and many more
Should we teach programming? (My take)

- Depends on the student population/B.S. program design
- All students? High school? Kindergarten?
  - Programming the "Latin of the 21st century"?
  - Is Scratch/Logo/... the best way to educate (future) digital citizens?
- System designers and implementors
  - No alternative yet
Should we teach programming? (Your input)

- Should the answer depend on the student’s level
  - What are good levels?
  - For each group, what is the answer?
- What should be the focus of ”Programming”?
  - Text understanding? Analytical/Critical thinking?
  - Testing
  - System design and analysis
  - Use of tools
Related aspects – Teaching Programming

- Student practice a big problem
  - Must synthesize solution
  - Partition problems into sub-problems, find solutions, integrate results
- Many students may underestimate importance of practice (and effort required)
  - Evidence that practice is crucial
  - Exam: 2 hrs live programming plus 40 min written test (foundations)
Concluding remarks

- LLM Tools (Copilot) – the end of programming?
  - Not really for B.S. CS students that design future systems
  - Assistance – helpful for skilled programmer, challenging for others
  - Students must still master concepts

- Copilot implications (for I2P)
  - Even more emphasis on testing & debugging & requirements analysis

- Challenges and research issues
  - Student engagement: may ignore practice (even more than now)
  - Interface (human) programmer <-> tool (programmer assistant)