CS1
V2.0
Motivation

Shifting from Teacher-centered to Student-centered learning

Identifying variation within the student cohort

Implementing programming exercises to foster theory

 Increasing focus on imperative programming
Introductory Programming

166 Students
15 ECTs
7 Weekly assignments
2 Assignments
3 Tests
1 Four-week project
(Objects-first with Java)
Introductory Programming

Stakeholders question students basic imperative competencies.

Are they capable of composing basic constructs?

[ Nicolajsen, S. M., Understanding Programming Languages as an Advanced Beginner. ]
Introductory Programming

Increase opportunities of *learn-by-doing*
(Shifting from Teacher-centered to Student-centered learning)

Understand student diversity in terms of experience
(Identifying variation within the student cohort)

Systematically design exercises to strengthen understanding of PL constructs
Increasing focus on imperative programming
Shifting from Teacher-centered to Student-centered learning

Lectures contain multiple (different) techniques and theory.

( \( T_A, T_B, \ldots, T_n \) )

Labs (or exercises) following iterates these.

( \( E_{A1-X}, E_{B1-Y}, \ldots, E_{n-Z} \) )
Shifting from Teacher-centered to Student-centered learning

A week

Lecture
Exercise

A week

Lecture
Exercise
LiveCoding

A week

Mixed

Mixed
LiveCoding
Exercise
Shifting from Teacher-centered to Student-centered learning

\[
\begin{array}{c}
T_A \\
T_B \\
\ldots \\
E_{A1-n} \\
T_{B1-n} \\
\ldots \\
\end{array}
\]
Shifting from Teacher-centered to Student-centered learning

while

\[ E_{A1} \ldots E_{An} \]

for

\[ E_{B1} \ldots E_{Bn} \]
Shifting from Teacher-centered to Student-centered learning (in practice)

Typically 1-2 exercises
Alone and in combination

Use most fitting
Or combine
What are the effects?
*What do you feel about the format compared to other courses? (N=50)*

- Easier to learn: 26%
- Confrontation: 32%
- Fast testing: 5%
- Gives breaks: 13%
- Prefer exercises: 4%
- Useful: 2%
- Learn by doing: 2%
- Different time to completion: 2%
- Long duration: 2%
- More readable exercises: 2%
- Missing time: 2%
- Speed varies: 2%
- Good with async: 2%
- Increases interest: 2%
- Difficult to understand: 2%
- More readable exercises: 2%
- Confrontation: 32%
- Fast testing: 5%
- Gives breaks: 13%
- Prefer exercises: 4%
- Useful: 2%
- Learn by doing: 2%
- Different time to completion: 2%
- Long duration: 2%
- More readable exercises: 2%
- Missing time: 2%
- Speed varies: 2%
- Good with async: 2%
- Increases interest: 2%
- Difficult to understand: 2%
* Anything about the teaching you find particular good (please expand)? (N=79)
Sometimes yes, however, other times I feel like the teaching is going too fast.

Yes!! – There is relatively many successes. The exercises during teaching helps a lot with understanding the concepts. It is super nice that there is so much focus on letting the students getting "hands-on". A little like learning a music instrument, it makes sense that you do not have a teacher who runs theoretical one-way communication to their students but that the teacher allows their students to try things in practice before moving on with more theory.
Overall, I benefited from the course

5.61

The course was organized in a way that helped me learn

5.57
Identifying variation within the student cohort

Student experience impacts:
- Performance
- Self-efficacy (also of others)
- Retention

And potentially course design.
I have made programs of 50-250 lines of self-written code in an all-purpose programming language, e.g., Java, Python, Javascript, C#, C, Pascal, Basic; 55

I have made programs with 10-50 lines of self-written code in "real" languages, scripting languages, Scratch, AppMaker, or the likes; 23

I had no programming experience (Use of Excel or HTML on websites do not contribute towards this); 53

I have an AP degree in Computer Science (or similar); 3

I have made programs with more than 250 lines of self-written code or more than 100 lines in multiple languages; 32
Affecting CS1 courses…

What is the focus of CS1?


Extracurricular or mandatory?
Implementing programming exercises to foster theory

Systematically implement programming exercises.
Focusing on Tasks, Techniques, Technology, and Theory
Implementing programming exercises to foster theory
Implementing programming exercises to foster theory

1) Introduce (part of) technique
2) Task to learn-by-doing
3) Attempt technology generation through epistemological obstacles.
4) Generate theory by institutionalising differences between techniques.

Repeat until technique is covered for all related techniques.
Implementing programming exercises to foster theory (in practice)

1) Introduce Sets
2) Task in using Sets

3) Set and Map "choice-and-implement" exercises.
4) Outline benefits and downsides of structures, and use cases.
Implementing programming exercises to foster theory

Current (anecdotal) observations from this exercise design:

1) Students are more capable of keyword identification
   [Nielsen, S. K., *Obstacles and strategies of Novice programmers*]

2) Students still lack strategies for translating from problem to code.
   [Nielsen, S. K., *Obstacles and strategies of Novice programmers*]
Lack of strategies
Lack of strategies
Lack of strategies
Lack of strategies

Problem formulation
Which constructs should I use?

How do I put things together?
Patterns I have seen before, I need to combine …

A problem defined far from programming instructions

The word 'if' is used, I need to iterate …

P₀ → P₁ → P₂ → S*
Increasing focus on imperative programming

<table>
<thead>
<tr>
<th>UGE</th>
<th>#</th>
<th>DATO</th>
<th>EMNE</th>
<th>AFLEVERINGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>01</td>
<td>Aug 30</td>
<td>Introduktion &amp; Sekventielle konstruktioner</td>
<td>(Høsten begynder)</td>
</tr>
<tr>
<td>02</td>
<td>02</td>
<td>Sep 01</td>
<td>Kontrol Strukturer: forgrening &amp; Iteration</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>03</td>
<td>Sep 06</td>
<td>Objekt-Orienteret programmering (Introduktion)</td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>04</td>
<td>Sep 08</td>
<td>Funktioner</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>05</td>
<td>Sep 13</td>
<td>Datastrukturer I</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>06</td>
<td>Sep 15</td>
<td>Datastrukturer II</td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>07</td>
<td>Sep 20</td>
<td>Datastrukturer III &amp; Standardbibliotek</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>08</td>
<td>Sep 22</td>
<td>Arrays</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>09</td>
<td>Sep 27</td>
<td>Objekt-Orienteret programmering I</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>Sep 29</td>
<td>Objekt-Orienteret programmering II</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>--</td>
<td>Okt 04</td>
<td>Obl. programmeringsprøve 1 kl 14:00 @ AUD1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>Okt 06</td>
<td>Input/Output &amp; Regulære udtryk</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>12</td>
<td>Okt 11</td>
<td>Nedavnning I</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>Okt 13</td>
<td>Nedavnning II</td>
<td>(Høsten slutter)</td>
</tr>
<tr>
<td>42</td>
<td></td>
<td></td>
<td>Efterårsserie</td>
<td></td>
</tr>
</tbody>
</table>
Increasing focus on imperative programming

Current (anecdotal) observations from this change:
(And based on results of mandatory tests and exams)

1) Students are *better* at understanding program flow

2) Some still lack basic imperative (algorithmic) understanding
Are the students learning to code?

*Do you feel like you are learning to code? (N=98)

- Yes: 95%
- No: 3%
- Maybe: 2%

06/12/2022
Changing Introductory Programming – Sebastian Nicolajsen @ IT University of Copenhagen
Are the students learning to code?

Grade distribution comparison (%)

(N₀=125, N₁=152, N₂=120)
Takeaways & Future work

- The mixed format encourages confrontation and training
- Obtaining data on student experience is essential for design
- We need to train strategies for programming more.
Questions?
Not approved for exam

<table>
<thead>
<tr>
<th>N</th>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>
Recording use (hours spent)