Tutorial 3 (invited)

Introduction to Computational Thinking
1 - Essential Concepts
2 - Pedagogical/Curricular Relevance

Ulrich Hoppe
University of Duisburg-Essen (Germany)
http://www.collide.info
Personal background

Secondary education teacher (math/physics)

PhD in Educational Technology (Tübingen, 1984)

Researcher in HCI / Adaptive UIs (1984-1995)

Professor @ Univ. of Duisburg-Essen / Germany „Collaborative and Learning Support Systems“ (Applied Computer Science)
The essence of Computational Thinking (CT) lies in the creation of “logical artifacts” that externalize and reify human ideas in a form that can be interpreted and “run” on computers.

The current discussion of CT can be traced back to Wing (2006). Wing characterized CT by stating that it “involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science”.

CT involves representing information in the form of data structures as well as techniques for the specification of algorithms for data processing and analysis. It is a problem of the current educationally oriented discussion of CT that the specific meaning of some of the underlying concepts (esp. "abstraction") differs substantially from the common sense understanding.
Conceptualizing, not programming. Computer science is not computer programming. Thinking like a computer scientist means more than being able to write programs. It includes thinking recursively, designing systems, and understanding human behavior by drawing on concepts fundamental to computer science.

It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.

Computational thinking is thinking recursively. It is parallel processing. It is interpreting code as data and data as code. It is type checking as the generalization of dimensional analysis. It is recognizing ants to describe a system’s behavior succinctly and declaratively. It is having the confidence we can safely use, modify, and influence a large complex system without understanding its every detail. It is

... thinking at multiple levels of abstraction

... using abstraction and decomposition

... thinking recursively

... designing for simplicity and elegance
Abstraction, the cognitive process of isolating, or “abstracting,” a common feature or relationship observed in a number of things, or the product of such a process. The property of electrical conductivity, for example, is abstracted from observations of bodies that allow electricity to flow through them; similarly, observations of pairs of lines in which one line is longer than the other can yield the relation of “being longer than.”

What is abstracted—i.e., the abstraction or abstractum—is sometimes taken to be a concept (or “abstract idea”) rather than a property or relation. Which view is taken on this issue depends in part on the view one holds on the general issue of universals (entities used to explain what it is for individual things to share a feature, attribute, or quality or to fall under the same type or natural kind).
Abstraction – from what?

mathematics

programming, data
organisation / representation

computer science

creating "abstractions"

externalisation, operationalisation

abstraction

"primary experience"

counting, arithmetic, concrete constructive geometry
... back to the roots:
Seymour Papert (1928-2016)

- Logo (*since ???*) 1967
  
a constructive medium to explore „mathland“
  
affording „ego-syntonic“ experience (esp. with „turtle geometry“)
  
low threshold / high ceiling
Initially, we have a square with one turtle in each of the four corners, viewing towards the next corner (clockwise) along the side of the square.

Each turtle will continue looking towards its successor as they start to move forward step-wise.

What will be the shape of the turtles’ trajectories, and how long will these be when they meet eventually?
TO pro4 :s

; bring four turtles to life:
SETTURTLE 1
SETTURTLE 2
SETTURTLE 3
SETTURTLE 4

; position the four turtles
; in the corners of a square:
ASK 1 [PENUP SETPOS [-125 -50]]
ASK 2 [PENUP SETPOS [-125 200]]
ASK 3 [PENUP SETPOS [125 200]]
ASK 4 [PENUP SETPOS [125 -50]]

; draw the surrounding square
; using turtle 0:
SETTURTLE 0
PENUP
SETPOS [-125 -50]
PENDOWN
square 250

; make the MOVES:
t4moves 1 :s 1

; hide turtles 1-4
REPEAT 4 [ASK REPCOUNT [HIDETURTLE]]
END
TO pro4 :s

; bring four turtles to life:
SETTURTLE 1
SETTURTLE 2
SETTURTLE 3
SETTURTLE 4

; position the four turtles in the corners of a square:
ASK 1 [PENUP SETPOS [-125 -50]]
ASK 2 [PENUP SETPOS [-125 200]]
ASK 3 [PENUP SETPOS [125 200]]
ASK 4 [PENUP SETPOS [125 -50]]

; draw the surrounding square using turtle 0:
SETTURTLE 0
PENUP
SETPOS [-125 -50]
PENDOWN
square 250

; make the MOVES:
t4moves 1 :s 1

; hide turtles 1-4
REPEAT 4 [ASK REPCOUNT [HIDETURTLE]]
END

TO t4moves :k :s :curT

; set the initial turtle for this round:
SETTURTLE :curT

; determine the next turtle:
LOCAL "nexT
MAKE "nexT 1 + REMAINDER :curT 4

; if close enough ... then STOP
IF ( dist ( ASK :curT [POS] )
     ( ASK :nexT [POS] ) ) < :s [STOP]

; (otherwise) move towards next turtle:
SETHEADING TOWARDS ( ASK :nexT [POS] )
PENDOWN
FORWARD :s

; ... and start the next round:
t4moves :k + 1 :s :nexT
END
Four turtles pursuit – learning goals

- Basic turtle geometry extended with direction finding
- Mapping behavior to agents (turtles)
- Modularisation
- Procedural abstraction
- Parameter passing
- Recursion
Four turtle pursuit in Scratch
Feedback from “Dr. Scratch”
[Moreno & Robles, 2015]

Score: 13/21

The level of your project is... DEVELOPING!
You’re doing a great job. Keep it up!!!

Best practice
- 4 sprite attributes.
- 1 sprite naming.

Project certificate
icce.sb2
Download
“Dr.Scratch” - assessment rubric  
[Moreno & Robles, 2015]

<table>
<thead>
<tr>
<th>CT Concept</th>
<th>Basic</th>
<th>Developing</th>
<th>Proficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstraction and problem decomposition</td>
<td>More than one script and more than one sprite</td>
<td>Definition of blocks</td>
<td>Use of clones</td>
</tr>
<tr>
<td>Parallelism</td>
<td>Two scripts on green flag</td>
<td>Two scripts on key pressed, two scripts on sprite clicked on the same sprite</td>
<td>Two scripts on when I receive message, create clone, two scripts when %s is &gt; %s, two scripts on when backdrop change to</td>
</tr>
<tr>
<td>Logical thinking</td>
<td>If</td>
<td>If else</td>
<td>Logic operations</td>
</tr>
<tr>
<td>Synchronization</td>
<td>Wait</td>
<td>Broadcast, when I receive message, stop all, stop program, stop programs sprite</td>
<td>Wait until, when backdrop change to, broadcast and wait</td>
</tr>
<tr>
<td>Flow control</td>
<td>Sequence of blocks</td>
<td>Repeat, forever</td>
<td>Repeat until</td>
</tr>
<tr>
<td>User Interactivity</td>
<td>Green flag</td>
<td>Key pressed, sprite clicked, ask and wait, mouse blocks</td>
<td>When %s is &gt;%s, video, audio</td>
</tr>
<tr>
<td>Data representation</td>
<td>Modifiers of sprites properties</td>
<td>Operations on variables</td>
<td>Operations on lists</td>
</tr>
</tbody>
</table>
(Bad) habits found with Scratch
[Moreno & Robles, 2014]

“Students repeat code in the same project, sometimes even in the same character programs.”

“Thus, abstraction and modularization, two key components of the development of computational thinking, are not trained.”

Fig. 2. Two scripts repeating code

Fig. 3. Definition of blocks to avoid code repetition.
SCRATCH - pro‘s & con‘s

- “Engaging”
- Nice metaphors (sprites, sprite communication)
- Not inviting learners to use powerful programming abstractions
- Message passing (“broadcast”) often used to replace procedural/functional calls
- Limited abstraction support (functions, recursion)
An „unplugged“ example

Formal exercise,
not using a programming language
The PLAN

PLAN x y z 1

write (x,z)
The PLAN

- PLAN x y z 1
- write (x,z)

- PLAN x y z n (nat. number n > 1)
  - switch 0
  - switch 1
  - PLAN x z y n-1 (switch inputs 2-3)
  - write (x,z)
  - PLAN y x z n-1 (switch inputs 1-2)
The PLAN

with $x=a$, $y=b$, $z=c$ and $n=3$
Learning goals ("Hanoi unplugged")

- **Recursion** as a principle of problem solving

- **Symbolic** descriptions or notations can be operational and guide/generate action

- Solutions can be **generalized** (n=3 to any)
HANOI program in Logo

```
TO hanoi :From :Aux :To :N
  IF :N = 1 [PRINT WORD :From :To STOP]
  hanoi :From :To :Aux :N-1
  PRINT WORD :From :To
  hanoi :Aux :From :To :N-1
END
```

```
show count hanoi.op "a "b "c 5 31
show hanoi.op "a "b "c 3
  [ab ac bc ab ca cb ab]
show hanoi.op "a "b "c 4
  [ac ab cb ac ba bc ac ab cb ca ba cb ac ab cb]
show count hanoi.op "a "b "c 4 15
show count hanoi.op "a "b "c 5 31
```

Functionality style

```
TO hanoi.op :From :Aux :To :N
  IF :N = 1 [OP WORD :From :To]
  OP (SENTENCE
    (hanoi.op :From :To :Aux :To :N-1)
    (WORD :From :To)
    (hanoi.op :Aux :From :To :N-1))
END
```

Procedural style

```
```

Functional style
HANOI program in Scratch
Recommendations

- Use **Scratch** to provide initial, introductory experience
  … move on to
- **Python**, “good old **Logo**” (FMS, NetLogo), …
- **Snap!** (Brian Harvey et al.)
  supporting functional programming and full recursion
The Role of Programming

Programming is a constructive (possibly also cooperative) activity

Programming languages induce programming styles (which, in turn, may induce thinking styles)

Students should not only learn to program, but they should understand/reflect the influence of programming languages (as tools of thought) on how to abstract and how to make abstractions work!
Beyond its direct relevance as an essential prerequisite for professions in information technology and computational systems, computational thinking (CT) has influenced scientific methods and models and thus our way of thinking and reasoning in general.

Wing (2008) asks: "If computational thinking is added to the repertoire of thinking abilities, then how and when should people learn this kind of thinking and how and when should we teach it?"

Adequate answers to this bundle of questions have to be contextualized in and harmonized with general pedagogical theories of curriculum and lines of argumentation to justify educational content.
Basic questions

Talking about a specific kind of knowledge and/or skills …

- why should everybody (?) acquire these as part of general education?
- how is it related to other parts of “canonical” knowledge / skills?
- how should it be taught and when?
Pedagogical / Curricular Relevance …

according to W. Klafki (1927-2016):

- **Exemplary quality** of the subject
  
  *(beyond the specific content – “genericity”)*

- **Meaningfulness** for present life situation and for mastering the future

- **Structural quality**

- **Accessibility**
  
  *(on the part of the learners in the given context)*

Supporting critical thinking and self-determination
Klafki (1959): „Categorial Education“

What to learn / teach?

Selection based on “material” value (content)

Selection based on “formal” or “functional” value (method)

<< Synthesis >>

exemplary quality

contextualization
Exemplary Quality / Genericity

A simple reason:

*Computational thinking / understanding is an important ingredient of modern technology and science*
Exemplary Quality / Genericity

A simple reason:

*Computational thinking / understanding is an important ingredient of modern technology and science*

… and beyond:

*a door opener to understanding the principles and limits of logical operations in brains and machines (computers)*

[see: Hofstadter, 1979]
Meaningfulness

CT is a key to professional skills
(now and in the foreseeable future)

CT brings awareness about the potential and limitations
of logical/computational operations
=> enables learners to critically reflect on
- what computers are able to do (and not)
- in how far human reason can be explained
  by “mechanics”
Meaningfulness – historical perspective:

**Big ideas** related to the “human condition”:
The human efforts to operationalize (or “mechanize”) reasoning and rational decision making.

Alfred North Whitehead & Bertrand Russell: “Principia Mathematica” (1910)

Gottfried Wilhelm Leibniz (1646-1716)

Kurt Gödel (1906-1978)

David Hilbert (1862-1943)
Structural Quality

CT reflects formal & logical structure of problem solving and the general principles and limitations of computational modeling.
Accessibility

Logo …
Squeak …
Scratch …
Snap! …

resolved?
„Wrap-up“

- CT is qualified as a subject element of a general curriculum (according to Klafki)

- CT has the “mission” to transmit *big ideas of computer science* (mathematics?)
  - taking up the relevant abstractions
  - bringing in the historical perspective
Changing Discourse

CT before 2006
- epistemology
- mathematics
- fundamental ideas

CT in a general or comp.sci context
- games
- modeling
- simulation
- skills

CT in a scientific inquiry learning context
- scientific ideas
Caveats

- New interactive devices and easily usable tools are not sufficient for bringing forth “good computational thinking”

- Trivialization of core concepts (such as abstraction, generalization) by stripping off computer science meaning and context

- Preparation of teachers often insufficient (especially related to abstractions and “big ideas”)
Implementation strategies

- Spread CT over different subject areas (revenant of pluralistic “ICT education”)
- Establish computer science (hosting CT) as part of the canonical structure of school subjects
- Establish CT in school curricula as an extension of the mathematical domain