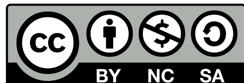
Methods in Computer Science education: Analysis

Teaching Computational Thinking

Andrea Sterbini – sterbini@di.uniroma1.it



This lesson's topics

What are we doing here?

How is it structured this course?

What is Computational Thinking?

Why one should learn how to program?

What false assumptions people have about Computers?

What new concepts are introduced because of Computers?

How to teach how to analyse and build a program?

How other fields can benefit from C. Science methods?

What Learning environments could be used?

How others are teaching C.T. around the world?

What are we doing here?

OUR MAIN GOAL: How do we teach Computational Thinking?

WHY? (today)

HOW? (rest of the course)

- Define the Computation Thinking concepts
- Analyse several Learning environments
- Define a methodology/checklist
- Analyse example learning units
- Build learning units

Course methodology

The course is very hands-on, we will

- Use **many** learning environments, visual or textual
- **Analyse** their strengths/weaknesses w.r.t. learning Computational Thinking
- **Analyse** learning units built by others
- **Design and Build** complete functioning learning units

We focus on interdisciplinary learning units

- To better motivate the application of the **Computational Thinking methodology**
- To show that programming helps **understanding the problem to be solved**

This is the first run of this course, thus your comments/suggestions/improvements/critiques are VITAL

Course prerequisites

You MUST be fluent in AT LEAST one programming language

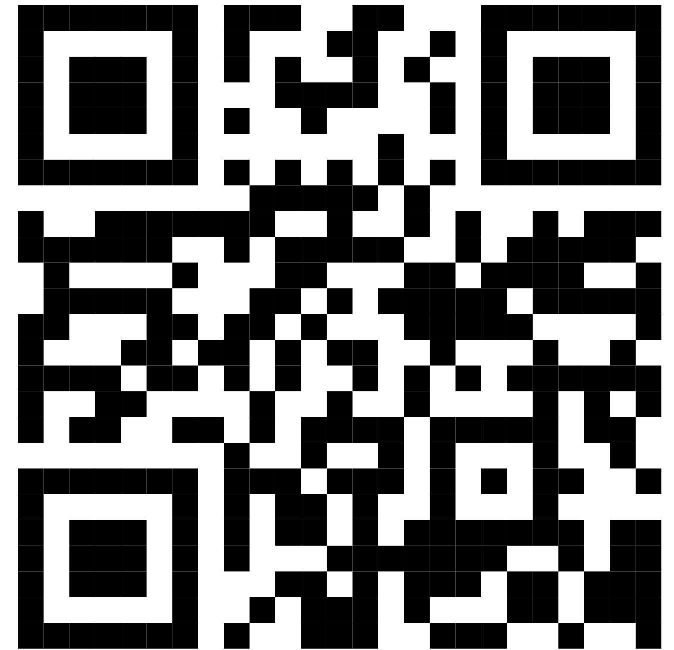
Python?	C/C++?	Java?	Pascal?	Ruby?	Lua?
Prolog?	Scala?	JavaScript?	Assembly?	Go?	???

You MUST be fluent in AT LEAST one programming paradigm/style

Procedural?	Object Oriented?
Declarative/logic?	Functional?
Data-flow?	???

Please fill the on-line questionnaire

http://bit.ly/CSedu-q1



Course assessment

You will build new interdisciplinary learning units in 3 different learning environments/systems of your choice (60% of the grade)

- You can work either alone or in small groups (max 3)
- Groups are expected to produce more complex learning units. The group work done should be clearly split among the participants (**“who did what?”**)

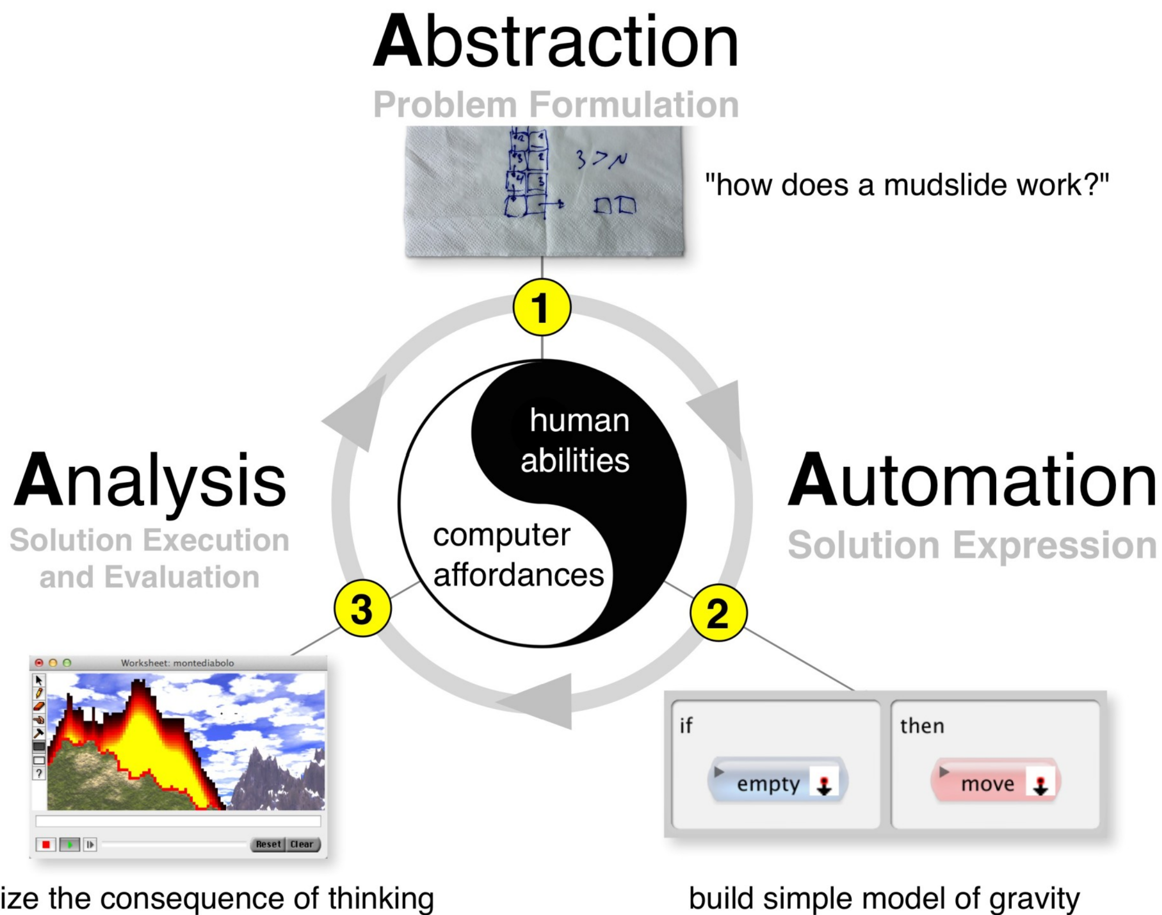
Learning unit presentation and discussion (40% of the grade)

- You will present and discuss with the rest of the class your learning units, describing motivations, methodologies, features, experienced problems, possible problems for application in class and proposed solutions

“Net-borrowed” learning units gets just up to 50% of the grade

- Depending on the quality of the presentation/discussion

What is Computational Thinking? [Papert ‘80]



Abstraction

Analysis, representation

Automation

Planning steps
Define sub-problems, and transformations

Analysis

Observation, consequences, evaluation

Image by KaptainFire - Own work A. Repenning, A. Basawapatna, and N. Escherle, "Computational Thinking Tools," to appear at the IEEE Symposium on Visual Languages and Human-Centric Computing, Cambridge, UK, 2016., CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=48453667

Computational Thinking: Abstraction

Abstraction of information

Data representation, variables and memory, objects and attributes, types

Abstraction of process

Sequential algorithms, **event-based programming**, parallel programming, data flow, declarative programming, object oriented programming

Abstraction of methodology

Top-down analysis, bottom-up analysis, declarative style, flow-based, pattern-matching rules, object orientation, functional, ...

Computational Thinking: Automation

Find a suitable representation for the information

Split the problem in small steps (or better said "smaller problems")

Order them in a sequence/algorithm

Describe the data flowing between steps

Find a "suitable" implementation of the steps

- Within the constrained resources available (time, memory)

But also: (motivation for literate/well documented programming)

- Prepare for the evolution/maintenance of your solution
- Keep track of the ideas guiding your thoughts
- Enable/empower others to use your solution

Computational Thinking: Analysis of the execution

Prepare for observation

Choose good visualizations, print intermediate data to expose inner details

Compare with expectations

Simulate the **algorithm in your head**, **predict the outcome** for simple cases, define test cases/examples

Diagnose discrepancies w.r.t. specification AND expectation

Find reasons for observed discrepancies, use assertions to early detect for anomalies, debug and observe the inner computation (variables **AND** flow)

Better understand BOTH the problem AND the computer

The **problem description/specification** could be challenging to fully grasp
The **programming** language, functions, libraries can be tricky to master

What about Social impact of C.T.?

C.T. could be seen as too much focused on the C.T. process

Abstraction / Automation / Analysis

A critique moved to C.T.:

little analysis of the impact on other fields

Reuse and modularity, analogy, social impact

For this reason (and others) we will design interdisciplinary units

And we will give some attention also to the program “life” and to the data required, managed, deduced

Why one should learn C.T.?

Pro:

Computer Science is the Science of **HOW (to represent, to compute, to solve)**

You will see other fields (Society, Music, Language, Art, Medicine ...) with a different analytic/creative eye

Society is more and more computer-based, therefore knowing how to write new programs makes you **less dependent** on others

You can **explore (virtually and physically) new ideas** at relatively low cost

Even if you WILL NOT program, you will **understand the possibilities** and you will be able to describe **what you want** to be programmed/created

Con:

Shabby/good-enough solutions trick you into **false understanding and lazy methodology**

The **social impact** of a program or of its data could be way bigger than you think

Motivation, in school, could be a huge problem

Teaching programming to university students is easier

- They chose it, and we (try to) go deep in many interesting ways

Some school students didn't choose the topic, but could be motivated by raising their interests with concrete problems

- Robotics, Embedded systems (see CS-edu:Design), Storytelling, Simulation, Social impact, Video games, Personal interests, Local issues, Phone apps

Role playing can make C.T. concepts very clear in a playful way to younger students

- They could either pose as the "programmed agent" or be the "programmer"

CS Unplugged activities can show C.T. methods without a PC

- Appealing for very very young students

What false assumptions people have about Computers?

You just need to know how to USE a computer (?!?)

Computers are FAST

- BUT DUMB!!! Limited instructions BUT bloody fast CPUs and intelligent algorithms

Computers are FLEXIBLE and MULTI-PURPOSE

- BUT RIGID and UNFORGIVING :-) There are soooooo many details to be aware of (declarations, initializations, scope, arguments, program termination, syntax, errors . . .)

Computers SAVE YOUR TIME, Programming is EASY (?)

- BUT programming is TIME-CONSUMING, you must be EFFICIENT and PERSISTENT:
  - When you code: (good IDEs, good documentation, easy programming languages, . . ., GOOD METHODOLOGY)
  - When you run (efficient algorithms, special data structures, . . .)
  - When you fix YOUR (or other's) mistakes (good documentation, good tests)

Computer can store HUGE amount of data

- BUT memory space is limited. Virtual Memory helps but SLOOOOWS DOWN EVERYTHING

What new concepts are introduced because of Computers? (methodology level)

Problem solution by <u>reduction to smaller problems</u>

Algorithm as a sequence of actions

(but see also declarative, parallel, data-flow, rule-based or … neural networks)

Data representation

Algorithms must manage some **meaningful representation** of information

Constrained execution (time, memory)

Simulation as tool to explore the impossible ("What if")

Explore multiple consequences in a virtual world with new rules

Empowerment and collaboration of the individual in the society

Open-data, Open-formats and Open-source development enable the single to **collaborate with others** and tackle global issues

Social issues of the information you receive/derive

Information as a good to be sold/exchanged. **Sensitive data**.

What new concepts are introduced because of Computers? (computer specific)

STATE changing through time (THE main difference w.r.t. Math)

Information representation, data types (analogy with Physics?)

Names vs memory (HUGE misunderstandings arise here)

Functions, arguments, return values

Side-effects (and bloody global variables)

Language syntax (bloody parentheses and semicolons)

Objects, attributes (and again, changing state)

Methods as object's actions/abilities, the office metaphor

Control structures (loops/repetition, conditions)

How to analyse and build a program?

Top-down analysis

- Define input/output data representation
- Write an high-level description of the problem, divided in steps
- Implement the algorithm by defining mock functions for each step, mimicking their I/O
- If needed:
  - define the additional intermediate data passed between steps
  - add the initial data definition and initialization
- Test if the logic is correct
- Repeat the analysis/implementation on each high-level step/function so defined
- When the steps are sufficiently detailed and similar to the programming language constructs, implement the details of the actual program

Be aware that

- Global variables → side-effects hidden from functions definition and usage
- Poor control structures and poor logic can produce inefficient/endless computations

Other analysis methodologies

Object-oriented

Define classes of objects responding to requests and interacting with each other. Try to reuse/standardize behaviours/definitions to simplify interoperability of objects and algorithms. Find common procedures but allow for exceptions.

Event-based (GUI, e.g. see AppInventor)

Describe how a collective set of objects should **react to external events**

Declarative/Logic-based

Describe **relations** among data and how more complex **properties can be derived** from simpler ones. **Let the system find a solution** plan.

Bottom-up

Start from small data manipulations and build more complex ones.

How other fields can benefit from Computer Science <u>methods</u>?

Exploration by simulation

Physics, Combinatorics, Chemistry, Geometry, …

Exploration by building computational models

Language generation and analysis, Music generation, …

Algorithmic description of problems/solutions or of rules

Math simplification, Language analysis

Learning a methodology to analyse problems

Data representation: a way to capture regularity and exceptions

Randomness: a tool to explore creativity (and mimic intelligence)

Simulation of Darwin's evolution

What approaches can make easier learning C.T.?

Syntax is considered one main problem

- We could completely remove the syntax by using visual programming
  - Joining **snap-on blocks** (Blockly, Scratch and similar)
  - Drawing **flow charts** (Flowgorithm)
  - Drawing **data-flows** (LabView and similar)
  - Editing **multiple agent properties/predefined behaviors** (GameMaker, Alice, ...)
- Or simplify the syntax to make the programs easier to read/write
  - Python, Ruby, Scala, (Prolog), Occam, …

Helping the student to build a mental model of what happens

- Visualizations of the **inner program status** (variables, execution, debug)
- Visualization of **external effects** (simulated agents moving around, robots)

What Learning environments could be used?

In the rest of the course we will:

- **Analyse** environments/languages built for learning how to program
  - Visual-based: Snap, Scratch, Blockly, OpenRoberta, AppInventor . . .
  - Logo-based: NetLogo, LibreLogo
  - Java-based: Alice
  - Scala-based: Kojo
  - Logic-based: Prolog
  - Flowchart-based: Flowgorithm
- We will **build an example** learning unit within the environment/language
- We will find and **analyse learning experiences** from around the world
- We will **suggest/discuss/plan new learning units**
- You will **build and present the learning units** designed

First course goal: discuss/define a checklist/methodology for

Analysis of the language/tool w.r.t.

- How to map C.T. concepts, constructs and **methodologies** to the tool's usage
- Missing concepts, tricks to mimic what's missing
- Technicalities (installation, configuration, interaction with tools/robots)

Analysis of the learning units w.r.t.

- **Computational Thinking:** Data representation, control structures, methods of analysis, programming style, ...
- **The problem solved:** shallow/deep understanding of the topic, possible extensions, didactic method applied, . . .
- **Application in class:** age, prerequisites, technical requirements, personnel required, ...

How others are teaching C.T. around the world?

Visual programming

Scratch Blockly Snap! AppInventor OpenRoberta

Programmareilfuturo.it code.org

Commercial

Microsoft Minecraft Education edition education.minecraft.net

Apple Swift Playgrounds (on iTune)

Wolfram computationalthinking.org

Less knowns approaches

Flowgorithm, LabView, NetLogo, Alice ...

Homework

Send me (sterbini@di.uniroma1.it)
your comments/suggestions on these slides

Fill the on-line questionnaire

http://bit.ly/CSedu-q1

(it takes just 2 minutes)