Computational Thinking in the Classroom

Dr. Martine Ceberio
Computer Science Department, UTEP
August 16, 2018 - GRIT, 2nd Annual Canutillo ISD Prof. Dev. Conference
Today’s Plan

- Some of *my experience* and some of *yours*
- *Make it a discussion as much as needed*
Computational Thinking

What?
Why?
How?
What is Computational Thinking?

- Meet your two neighbors and [10 minutes]:

1/ Each of you shares to his/her group what they think CT is

2/ Discuss differences if any

3/ Do you use it in your classroom?
Computational Thinking?

- **A way of:**
  - Solving problems,
  - Designing systems, and
  - Understanding human behavior that draws on concepts fundamental to computer science.

- **Not limited to** computing or computer science
Computational Thinking [cont’d]

- A problem solving process that includes a number of characteristics, such as
  - logically ordering and analyzing data and
  - creating solutions using a series of ordered steps (or algorithms), and dispositions,
  - such as the ability to confidently deal with complexity and open-ended problems.

- Essential to the development of computer applications, but it can also be used to support problem solving
  - across all disciplines, including math, science, and the humanities. Students who learn CT across the curriculum can begin to see a relationship between subjects as well as between school and life outside of the classroom.

[from Google for Education: https://edu.google.com/resources/programs/exploring-computational-thinking/]
To flourish in today's world, **CT** has to be a **fundamental part** of the way people think and understand the world.

[from Carnegie Mellon University]
Computational thinking [cont’d]

- Algorithmically solving problems
  - Solving problems applies to any discipline

- Formulating problems such that computers can assist
  - In our digital age, knowing what can, cannot, should, etc. be done will be extremely valuable

- Analyzing and logically processing data

- Generalizing and applying this process to other problems
  - Abstraction, reusability, versatility
Computational Thinking... Why?

- Being able to solve problems is relevant to many disciplines
  - Law, medicine, engineering, etc.

- Problem-based learning has proven to be very successful

- Exposing students to problem-solving and possibly computer science will give them more options for careers
Computational Thinking... How?

- Obviously, this is central to **Computer Science 😊**

- **Mathematics**: posing problems and using the right tools to solve them

- But not only... What else?

- **QUESTION**: What do you do in your own classes? (or would like to do)
  - Take 5 minutes in your groups
  - Then share with the whole group
Simple examples

- **Computer science:**
  - Emphasize problem solving rather than putting sole focus on coding
  - More and more focus on this
  - UTEP collaboration with Google
  - CS unplugged
  - Kodu or similar

- **Mathematics:**
  - Posing problems rather than executing operations, repeating
    - Putting activities in context yield higher engagement and content retention
  - Show that many ways exist to solve a given problem, so that students have to think, pick, discuss
  - Use simple robots (e.g., for geometry)
Examples outside CS or Math

- **General activity (e.g., elementary school):**
  - The unplugged robot
  - Makes students think sequentially
  - Depending on the students’ level, discussion about:
    - The elements of their solution
    - The risks of their solution
    - How to make it better
    - What they would need to actually “plug” it
  - Etc.
Examples outside CS or Math

Social Studies:
- Pose (somehow) **open-ended problems** and have the students work on a **systematic approach** to solving them → e.g., the IDEAL framework
- Ask students to **design a video, create a video game, design an app**, etc. that addresses a problem presented in social studies
  - You can use programs like Scratch
  - More advanced (more time): robots, lego mindstorm?
  - E.g., identifying a problem, designing and building a solution
Examples outside CS or Math

- **Music:**
  - Plug it in an animated video
  - Have students design music with computers: creation of scales, etc.

- **Languages:**
  - Same as with music but with text for practice
  - English as a Second Language: using a simple programming language (like scratch or logo or even python turtle) can help students manipulate English at different levels (programming language, but also their project)
My Own Experience
+ Activities
My own experience

In CS:

- I teach CS1: intro to CS & I designed and taught a new Problem Solving course (along with Google)
- In **CS1**: problem-solving and programming (because we solve pbs on computers 😊)
- In **Problem-Solving**: pure strategy, no coding, no implementation

Outside CS:

- Worked with a French teacher (using Scratch)
- Worked with ESL teacher
Main Goals

- Keeping the interest of the students up:
  - **Motivation**: purpose and relating topics to their everyday lives
  - **Acknowledgment**: they know a lot already. I am just there to help them make sense of their skills → asset-based teaching

- Providing valuable training to my students
  - Equipping them with skills of **value across disciplines**
How can we do this?

- **Purpose:**
  - Use *videos* to show students what Computer Science is: code.org is a great resource
  - Show what is done in other fields as well that relates to CT
  - Give students *projects* that are **relevant** (they could pick them)

- **Relevance:** Share with them the *accomplishments* of people in CS -- or other fields (make sure to *include diversity*: women, other minorities, and *culturally-relevant* environment)

- **Acknowledgment of their prior skills:** relate the topics to “real-life” common tasks and activities + be casual (show trust)
  - *E.g.*, algorithms: unplugged activities, robots
  - Recursion, repetitions: CS unplugged
  - Arrays and Linked-lists: rows of houses vs Treasure Hunt, Monkeys in a barrel
  - Etc.
Examples of Activities

- Activities to do together in the classroom:
  - 1. **Robot activity:** it makes them stretch and work in teams
  - 2. **Recursion:** counting together, checking a condition
  - 3. **Looking for an element in an array** (logic & storage): looking for an image on a computer screen for instance
  - 4. **Linked-lists manipulations:** monkeys in a barrel, balloons, linked-list of students (like a network of friends), etc.
  - 5. **Sorting** people, papers, etc.

- And you can come up with many more!

*Computer Science rests on computational thinking (algorithms, problem-solving). So you can teach it mostly without computers!*

*You can also use these in non-CS classes, even in PE*

*The trick is: “do it and tell later”*
Examples of Activities

Let’s go over:

- **Recursion:** counting together
- **Linked-lists manipulations:** adding, removing elements in a chain.

**QUESTION:** What else could you do? Share with your team:

- What you teach, what you have done in CT, how you did it, what you’d like to do
- What you wish you could do but do not know exactly how to do
Getting Started!
How to get you started?

- Visit the resources mentioned in this presentation

- But I am also happy to:
  - Help you design activities to get you started on your individual courses
  - Provide tutorials on tools and frameworks you can use
  - Provide workshops on how to design class activities around computational thinking
  - Build an interest group of teachers
  - Let me know: contact me (mceberio@utep.edu)
Existing Opportunities

- **School Districts can partner** with Code.org
- **Exploring CS**: summer professional development
- **EngageCSEdu**: a Google and NCWIT initiative → resources for the classroom
- The **Hour of Code**: First week of December
- **After-school** programs
  - E.g., with Little Bits: [http://littlebits.cc/education](http://littlebits.cc/education)
  - NCWIT AspireIT
- **Code.org**: curricula and ad-hoc activities available
Why is all of this important?

- We need to **inform young students about what CS is**: so they can make informed decisions
- We **need more people in CS**: many jobs (and even more going forward) will require knowledge of CS, or at the very least strong computational thinking
- We **need diversity in CS** (currently not diverse)

But mostly because:
- Technology is all around
- We need people with a general understanding of CT to discuss and design the tools of the next generation
- We **need skilled people**
- We need **SKILLED THINKERS**
Thank you!

Questions?

Martine Ceberio
Associate Professor of Computer Science
The University of Texas at El Paso
mceberio@utep.edu

Presentation available at: http://martineceberio.fr under Outreach
References

- Google for Education: https://edu.google.com/resources/programs/exploring-computational-thinking/
- Problem Solving @ UTEP: http://martineceberio.fr/blog/problem-solving-computer-scientists
- Code.org (http://code.org)
- Exploring CS: http://www.exploringcs.org/for-teachers-districts
- EngageCSEdu: https://www.engage-csedu.org
- The Hour of Code: https://hourofcode.com/us
- Little Bits After-school program: http://littlebits.cc/education
- NCWIT AiC: http://aspirations.org
- NCWIT AspireIT: https://www.ncwit.org/project/aspireit-k-12-outreach-program