ThinkTech

An outreach program to teach middle school girls computational thinking

Rebecca F. Alford, Tomit Huynh, and Anastassia Kornilova

Program Overview

- After-School, informal setting
- Students from various schools in the Pittsburgh area
- Meet once per week for 2 hours over 10 weeks
- Opportunity for mentoring by undergraduate and graduate student volunteers
- Focus on transferable understandings and dispositions

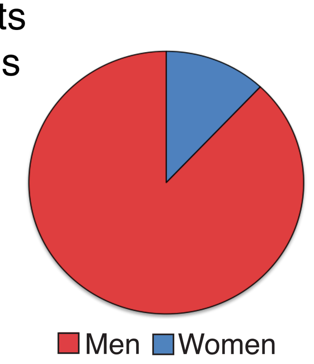
Goal: Teach middle school girls computational thinking skills, build confidence for solving challenging problems and provoke curiosity and interest in science, technology, engineering and math (STEM)

Motivation

Gender Gap
- Stereotypes about the computer science community discourage participation
- Women feel behind with skills compared to male counterparts
- Females lack confidence solving problems in unfamiliar fields

Educational Opportunities
- Lack of awareness of career opportunities
- Common misconceptions: "CS is just writing code"
- Limited computer science educational opportunities in K-12

A diverse community and equal opportunities further innovations in the field



Only 12% of computer science bachelors degrees are awarded to women (code.org)

Learners in Context

Learner Profile
- Female students, ages 11-14
- Limited previous experience
- Motivated to learn: personal interest or encouragement from parents or teachers

Prerequisite Knowledge
Basic algebra
Scientific reasoning
Pattern recognition
Data interpretation

Single-Gender Learning Environment
Research demonstrates a *confidence gap* between women and men. They lack a key growth mindeet - a belief that their abilities will improve. To accommodate, we specifically focus on **affirmation of choice, encouraging appropriate aggresiveness, discouraging total perfectionism,** and **building confidence.**

Research

Objective
- Evaluate serval new educational variables introduced in ThinkTech instruction including context for learning, interactive activities and feedback

Our key research questions are:
- What is more useful for instruction: **blocked** environemnts (like Scratch) or **code**?
- Do **interactive activities** outside of coding context improve transfer?
- **When and how do you provide feedback** to teach technical skills and dispositions?

Subjects
- Match our learner context - females, age 11-14
- Split into control and experimental groups by randomnly splitting the class by seating or section.

Setting
- Both classroom and laboratory

Possible Confounding Variables
- Motivation to learn
- Preference for visual or written tasks
- General receptiveness to feedback
- Confidence

Current Challenges

We encountered various challenges in the design of our curriculum. Some may be addressed by additional research, implenetation and iterations. We note the following key challenges:
- **Anticipating learner differences:** Few resources on how to teach and evalute a group of diverse learners
- **Multiple variables:** Each lesson introduces multiple new activities
- **Gender specific dispositions:** Few outreach programs address dispositions
- **Too many goals:** Scope of course covers many technical details

Goals and instruction aligned to key abilities needed tosSolve problems in computer science

Seven key **transfer goals** were designed based on the ACM Computer Science Teacher's Association (CSTA) standards, Explore CS curriculum, and TechNights curriculum.

1. **General computer science knowledge (G)** Be able to write simple programs and use field-appropriate terminology (data, algorithm) to communicate new knowledge
2. **Abstraction (AB)** Understand that behind complex processes are general patterns and strategies for problem solving
3. **Data Representation (DR)** Be able to identify features of simple datasets and implement ways to organize information (data structures: list, dict, graph)
4. **Algorithmic Thinking (AT)** Communicate a process to a computer, through code or to a friend, incorporating a procedure and required data
5. **Debugging and Troubleshooting (DT)** Learn strategies needed to solve computer bugs that transfer to general problem solving techniques
6. **Curiosity (CU)** Demonstrate excitement about various applications in computing, exhibit persistence while problem solving
7. **Confidence (CO)** Know that you are well-equipped to solve a hard problem, with limited resources, even if a solution is not immediately visible

Instruction directly derives from key transfer goals for an aligned educational module based on the **desired learning result**

1. **Session 1: What is computer science?** Panel of experts to discuss CS applications, (tell a story), algorithm for introductions (out-of-code context)
2. **Session 2: Abstraction and Art** Learn to generate explicit procedures and distiuish between general and specific through various art activities
3. **Session 3: Abstraction and the Internet** Continue learning about abstraction through interface and implementation on the internet.
4. **Session 4: Data structures and the human genome** Learn how to organize important datasets, such as information from the human geonome project, into data structures such as lists and dicts.
5. **Session 5: Graphs and our social networks** Continue exploring data organization by learning how social networking platforms like twitter, facebook, etc use graphs to organize networks
6. **Session 6: Algorithms for board games** Use board games to learn about algorithms in computer science and strategies for making them more efficient
7. **Session 7: Algorithmic Thinking** Apply algorithmic thinking skills from the previous session to programming a real chip using arduinos
8. **Session 8: How to catch your bugs - troubleshooting** Learn to debug using error-filled versions of code from your favorite sessions - generating art with code and organizing data into networks
9. **Session 9: Project - Exploring CS** Research an application in computer science you are excited about and show how it connects to the 5 technical topics from the program
10. **Session 10: Project - Exploring CS and wrap up** Continue researching and present your findings to the class as a wrap up for the sessions

Desired learning results and instruction designed to equip students to solve problems in CS. Example, writing code to print Fibonacci numbers:

Appropriate dispositions:

"I know I can solve this problem!" — Confidence to solve new problems (CO)

"I'm excited to learn more about the solution" — General curiosity for new ideas (CU)

Required Skills

Terminology to interpret requirements (G)
Skill to write a basic program (G)
Define a general method (AB)

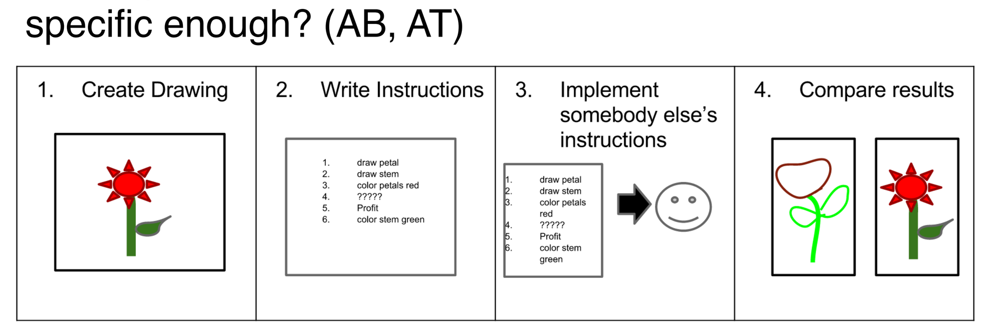
```
# Define a function that computes
# the nth fibonacci number
def fib( n ):
   if n < 2:
      return n
   return fib( n-2 ) + fib (n-1 )

# Call function to get a specific
# solution
>>> print fib( 4 )
3
```

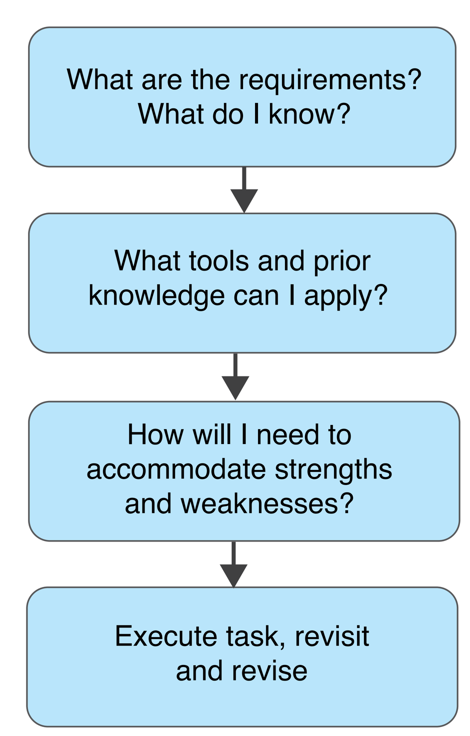
Gives data that may need to be organized for interpretation (DR)
Write detailed instructions to the computer (AT)

Metacognitive goals encourage active problem solving

Goals aim to teach transferable metacognitive strategies for solving problems in multiple contexts. Apply general processes accross lessons. Examples
- Use the internet to find example code
- Backtracking to last working point

Example Lesson: Drawing a flower using procedural thinking
Work in pairs - write instructions on how to draw a flower have partner execute drawing using instructions
- Guiding assessment question: Are my instructions specific enough? (AB, AT)



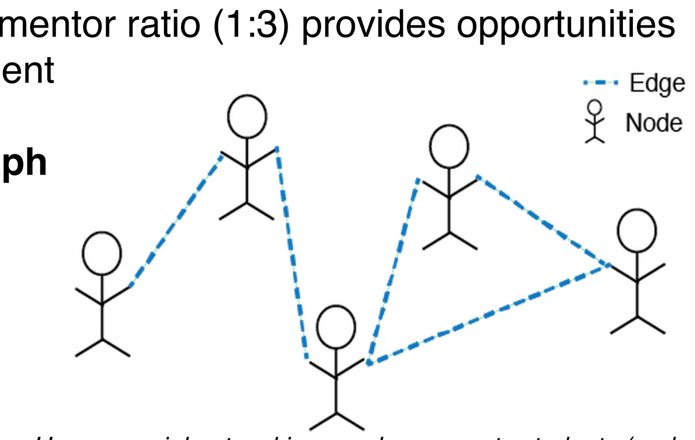


General metacognitive stragegy for approaching new problems

Instruction accomodates for individual and gender-specific differences

Single-gender learning enviornment enables us to design modules that accommodate for individual and gender-specific differences
- **Provide narrative:** Girls learn better given a story or context. For this reason, we couple content with applications in each lesson
- **Teach beyond the code:** Accomodate learner strengths and weaknesses by teaching concepts with code and interactive activities
- **Volunteer student mentors:** Small student-mentor ratio (1:3) provides opportunities for individualized feedback and encouragement

Example Lesson: Human social network graph
Work in a group - given a list of connections between students (edges), form a graph describing a network connecting the group
- Connect social networking 'story' to understanding the graph data structure (DR)



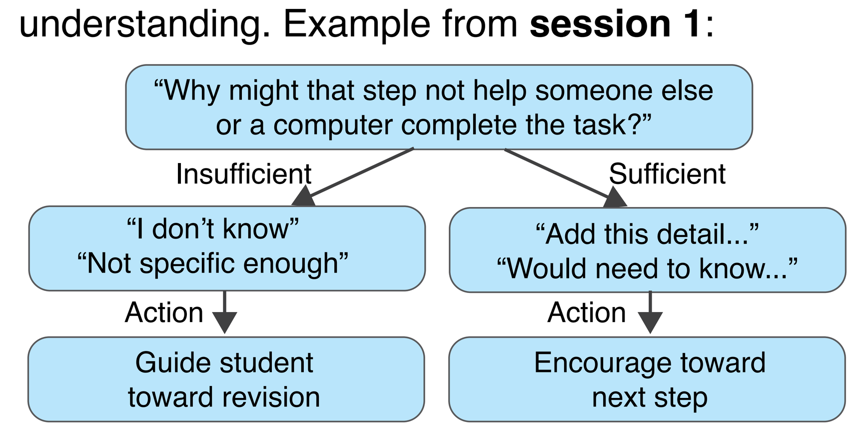
Human social networking graphs connects students (nodes) with string (edges) to present the graph data structure out of coding context

Informal Assessments enable real-time feedback

General Philosophy
We chose to conduct informal assesments (no tests or quizzes) to make the program fun and eliminate pressure in the learning environment. The informal assessments serve two purposes:
(1) Correct students misunderstandings in real-time
(2) Evaluate the effectivness of the instruction in real-time

Accomodate neurodevelopmental level
During adolescence, negative feedback or stressful scenarios easily discourage females from the field. To accommodate:
- Volunteers trained to use positive language during correction
- Informal assessments create low stress environment

Formative Assessment: Guiding Quesitons
For each activity, volunteers are provided sample quesitons and answers to evalutate student understanding. Example from **session 1**:



Summative Assessment: Final Project
Students will conduct a research project on some area of CS and how it connects to of the five big ideas discussed during sessions:

Steps of the activity pull from all seven transfer goals
1. Pick topic: CU
2. Research how the application connects to one of 5 technical transfer topics: AB, AT, DR, DT, and/or G
3. Present findings to the class: CO