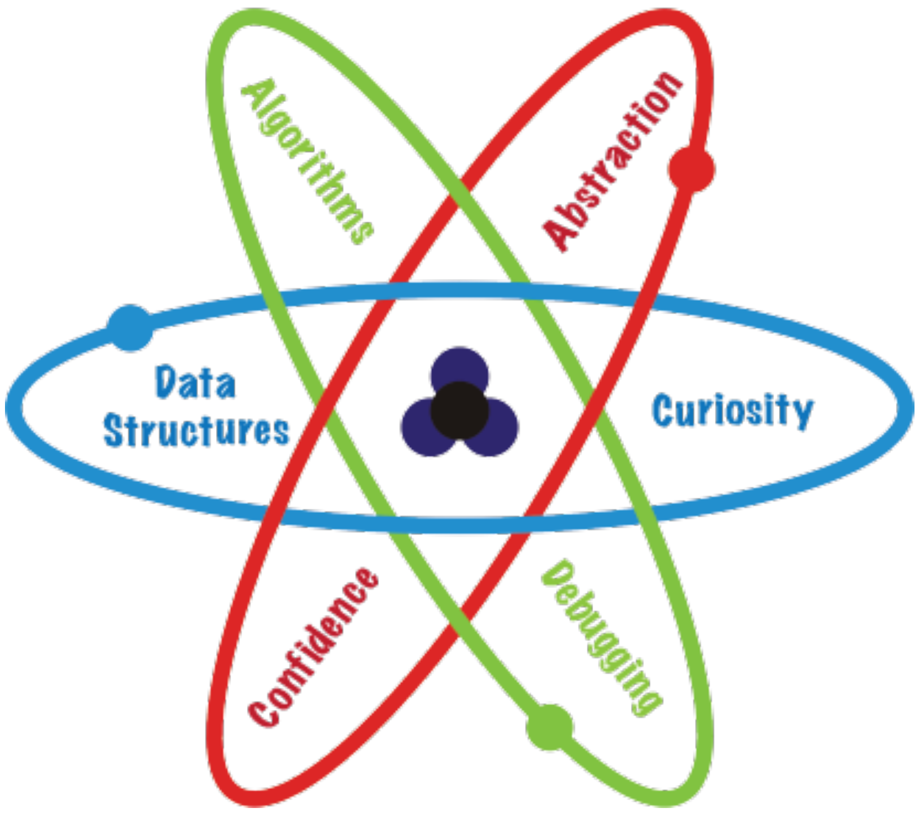
ThinkTech

An outreach program to expose Middle School Girls to computer science and teach computational thinking skills

Rebecca Alford, Tom Huynh, and Anastassia Kornilova

Final Draft Submitted: 12/13/14



"Everybody in this country should learn how to program a computer because it teaches you how to think" -- Steve Jobs

Author Designs

Several elements of this curriculum were individually designed and co-designed by the authors. The following color key describes who designed each component:

Red - Anastassia Kornilova
Blue - Rebecca Alford
Green - Tom Huynh
Purple - Co-Designed by all team members

Table of Contents

Program Overview

While 50% of women comprise the workforce, only 12% participate in science, technology, engineering or mathematics (STEM) related fields. In stark contrast, STEM fields, computing in particular, are emerging as critical skillsets in the 21st century. In effort to close this gap, we designed ThinkTech, an extracurricular outreach program for girls ages 11-14 to teach computational skills and achieve early exposure to computer science. Our curriculum guides students through key skill-based elements of computational thinking: abstraction, organization of data, design of algorithms, and debugging (problem solving). Each lesson achieves exposure by coupling with a current application in computer science. We also place special emphasis on dispositional goals of confidence and curiosity. Ideally, this curriculum will better prepare students as computational thinkers and open the door to interests and careers in STEM fields.

Learners in Context

Who will you teach? Age Range? Experience Level? Relationship to you?
Our learners are female, 11-14 year old middle school students from the greater Pittsburgh area. We anticipate our students will have age-appropriate math and science abilities. Because computer science instruction is sparse at the middle school level, our students will likely have little formal or informal exposure to computer science. Some girls might attend our program from personal interest in science, technology, education, and mathematics (STEM), whereas others might enroll by encouragement from their parents or teachers. Because this program is extracurricular and independent of formal instruction, we will have no previous relationship to the students. For this reason, we implement additional mentoring support by student volunteers discussed in the **Instruction Overview**.

In what domain have you or other educators noted learning challenges, and which of them will your project target?
Computer science has emerged as a critical field of study in the 21 century. However, the subject matter remains absent in formal education curricula. Most schools do not offer computer science courses at any level of study. Some high schools offer elective instruction in programming languages. Still, many of these do not emphasize the critical thinking and problem solving skills required for a successful career in computing.

Our project aims to fill two gaps in learning: exposure and skill development. First, we expose students to fundamental themes in computer science such as data structures, algorithms, debugging, and optimization. We also demonstrate application by presenting each theme with its use in some area of computing. For example, algorithms are presented with the Mars rover project, explaining how navigation software connects to procedural thinking. Second, we provide students with opportunities to develop computational thinking skills. Students will get hands-on experience designing algorithms and organizing data to strengthen their computational thinking skills.

Why is it important for them to learn what you plan to teach?
Computer science helps students to 'think better.' The rigor needed to approach new problems encourages students to think critically. Debugging enables learners to develop persistence and independence in problem solving. Algorithm design provokes unforeseen creativity. Furthermore, optimization prompts students to consider solutions beyond the immediate and obvious. These skills, while important for computing, transfer to nearly any career path: organized, procedural thinking can help anyone from doctors to managers. Therefore, we strongly believe developing these skills at an early age will facilitate future success in any career path in addition to equipping students to pursue computing.

Our project also accomplishes important social goals. Computer science is commonly mistaken as male-dominated field devoted to writing computer code. In reality, computer science is about using code to solve exciting real world problems. Still, most girls never correct these misconceptions until later in life and have few opportunities to pursue careers in computing. Our project provides a safe, welcoming environment for girls to acquire new skills, develop confidence, be creative and inquisitive, and dive deeper into, the potentially unknown, world of computing.

Where will you teach them? At what point does this instruction occur relative to the entire trajectory of their learning?

Our program is an after-school activity hosted at a local Pittsburgh school or at Carnegie Mellon University. While this program is extracurricular, its importance to student learning should not be underestimated. Our program falls 1-2 years before students enter high school. This allows girls time to pursue their interests in a low-risk environment. We specifically design our program with continuation in mind, highlighting resources for students to continue programming and learning once our sessions end. This alignment also provides ample time for girls to explore the computing field before choosing electives or extracurricular involvement prior to college. By designing our instructional program at this time, we believe girls from our program will gain confidence and perform on par with their male counterparts.

What are your expectations for learner's baseline relative to the domain?

Developmental Level

Our learners will be female middle school students from the greater Pittsburgh area ranging in age from 11 (youngest) to 14 (oldest). According to the American Psychological Association Reference for Developing Adolescents, females in this age-range exhibit a common set of developmental features. Specifically, adolescents often begin developing a sense of accomplishment: largely focused on academic and interpersonal activities. Adolescents also form their own set of values and beliefs, often contrary to parental directives. Children at this age begin improving their ability to communicate and express ideas clearly. In addition, students can engage in higher-level thinking processes.

We also recognize an important set of behaviors applicable to classroom learning. It is normal for adolescents to argue for the purpose of an argument and increase dramatics in a situation. Adolescents might also be self-centered and jump to conclusions in a situation before evaluating the facts. Students might also become competitive with one another; the instructional team will work to discourage this behavior, because our goal is to encourage confidence.

Knowledge Base

Prior Knowledge and Skills

Girls in our program will enter with wide-ranging exposure to computer science. We anticipate most students have little understanding of the science behind computing. However, some will have prior knowledge because of relatives in STEM fields or participation in prior outreach activities. In the design of our sessions, we accommodate for each end of this range by assuming little prior knowledge to ensure activities are accessible to all participants.

First, we assume little prior knowledge to ensure activities are accessible to all participants.

Second, we provide a set of advanced activities for more experienced students.

Our program targets 11-14 year old students. Accordingly, we expect our students to be knowledgeable of or learning the following subjects:

- **Algebra:** Variables, functions, solving single-variable equations, solving systems of equations
- **Scientific Reasoning:** Understanding, and possibly designing, procedures to answer questions, defining independent and dependent variables,
- **Pattern Recognition:** Linear processes, iterative ("looping") processes
- **Data Interpretation and Manipulation:** Plotting data on XY axes, organizing into tables and lists, using Excel

Technical Background

We expect our students to arrive with age-appropriate proficiency working with computers. We expect our students to have elementary or intermediate typing skills and can perform Internet searches. Students will have a basic knowledge of computer terminology such as program and operating system. Our students should also understand that computers use a specific set of instructions from people to perform tasks (conversely, vague instructions lead to lack of productivity).

Students enrolling in our program will travel from schools with different curriculums. We therefore expect diversity in the math and science backgrounds of our learners. Despite these differences, we anticipate our students will be proficient in arithmetic, elementary algebra, geometry, and graphing data. Knowledge in these areas will contribute to student success in our program. However, we do not establish official prerequisites because as an outreach program, we aim to broaden our influence and believe this can be best accomplished by setting few constraints.

Cultural Differences

According to the National Science Foundation, there remains a substantial gender gap in science, computer science and engineering fields. This bias may originate at home as some cultures do not believe computing is a suitable career for women. Alternatively, male dominance in technology might increase discomfort and discourage women from entering the field. Computing also requires confidence, which adolescents are beginning to develop and struggle with. We anticipate our students will enter our program with wide-ranging biases toward women in computing. It is our goal to eliminate these dispositions.

Individual Differences

Neurodevelopmental Level

We anticipate our learners to exhibit a wide range of neurodevelopmental profiles. Each student will vary in his or her ability to attend to details, remember concepts, express thinking in language, organize spatial and sequential information, control the motor system and work collaboratively. As described earlier, adolescents also experience several physical, social and emotional changes that will impact their ability to learn in our outreach program.

To accommodate for the varied neurodevelopmental level of our learners, we aim to vary instructional modes and provide support to students in the learning environment.

- **Social interaction:** Students will have wide-ranging communication skills. Some will be able to clearly identify problems and ask for help. Others might need guidance to express frustrations. We will use positive language and provide mentorship opportunities to bridge this gap.
- **Teamwork and Collaboration**: Our program will encourage a mixture of collaborative and individual activities. By providing a mixture of activities, students who are more comfortable in either setting can thrive in varied activities
- **Personality Differences**: During adolescence, girls begin to struggle with self-esteem. For this reason, we will train instructors and volunteers to identify when confidence is lacking.
- **Patience/Attention Span:** Students will also exhibit differences in patience. For this reason, we aim to keep tasks relatively short and engaging.
- **Temperament:** Some students will be more outgoing whereas others are more introverted. We will ensure instructors and volunteers reach out to all students to ensure proper attention and mentoring occurs.

Interest

Our students will exhibit different levels of interest. This is especially dependent on students’ prior exposure to the computing field. We hope to maintain engagement by adjusting applications and topics discussed based on individual interests. We also hope to vary topics of applications to serve a broader range of possible interest.

Goals

Broad Learner Context

ThinkTech is supplementary to a student's formal education. For this reason, learning goals aim to enhance the middle school experience by strengthening computational thinking skills and exposing students to the computer science realm. The course centers around a set of seven transfer goals we identified to be key for future success in pursuing computing or STEM related interests.

Developing Relevant, Ecologically Sound Goals

Our detailed set of goals is based on two sets of computer science learning standards and our own outreach experience. Both sets capture a wide age range and timescale (several years compared to 10 weeks). However, the key ideas remain the same. We specifically look at two curriculums:

(1) Association for Computing Machinery (ACM) - Computer Science Teacher Association (CSTA) Computer Science principles curriculum spans grades K-12. Here, we focus on grade 6-8 standards (Link: http://csta.acm.org/Curriculum/sub/CurrFiles/CSTA_K-12_CSS.pdf):

- **Computing Practice & Computing:** Understanding algorithms, dispositions for problem-solving and computer science applications.
- **Collaboration:** Teamwork skills and ability to integrate feedback
- **Computers & Communication Devices:** Differentiate between human and artificial intelligence, fundamental understanding of computer hardware.
- **Community, Global, and Ethical Impacts:** Understand global impact of technology and ethics of online behavior.
- **Computational Thinking:** Abstraction, visualization and general problem-solving techniques.

(2) United Kingdom Computing at Schools (CAS) curriculum divides standards into levels. We focus on level 4 and 5, which capture our target age range of 11-14. This curriculum does not explicitly define standards, but centers on the following key concepts Link: http://csta.acm.org/Curriculum/sub/CurrFiles/CASUKComputingCurric.pdf):

- Languages, machines, and computation
- Data and representation
- Communication and coordination
- Abstraction and design
- Wider context of computers

Transfer Goals

The aim of ThinkTech is to equip middle school girls with needed computational thinking skills and provide exposure to the exciting field of computer science. This desired result could be described by seven core transfer goals detailed below:

The 7 transfer goals were co-designed by all team members.

Goal	Criteria	Transfer	Connection to Other Curriculums
General Computer Science and Coding (G)	-General understanding of computer science and related fields. -Be able to run and write programs in python	-Use vocabulary and skills from course to acquire new computer science knowledge and skills	**ACM:** Computers & Communication Devices **CAS:** Languages, machines, and computation, The wider context of computers
Algorithmic Thinking (AT)	-Identify a sequential set of steps that solve a well-defined computational problem -Determine whether a solution is specific or general -Understand iterative or parallel procedures as alternative solutions	- Find systematic solutions for challenging problems - Evaluate whether a procedure appropriately fits the problem domain - Evaluate multiple solutions to a given problem	**ACM:** Computing Practice & Computing **CAS:** Languages, machines, and computation
Debugging and Troubleshooting (DT)	-Identify bugs vs. compiler errors (where applicable) -Understand the difference between a bug and weakness of a chosen algorithm -Acquire a set of strategies for debugging. Ex: Limiting scope,	- Patience and persistence when solving problems - Active and informed decision making during troubleshooting - Meta strategies for problem solving: organizing information	**ACM:** Computing Practice & Computing, Computational Thinking **CAS:** N/A

	internet search, finding specific conditions -Ask for help when students do not understand the nature of the bug or how to approach a solution		
Abstraction (AB)	-Understand the difference between a general and specific solution -Understand the difference between interface and implementation	-Relate familiar solutions to new problems -Explain situations with varying levels of depth. -Identify different solutions to the same problem.	**ACM:** Computational Thinking **CAS:** Abstraction and design
Data Representation (DR)	-Understand different formats for storing data -Understand advantages of certain data formats in certain situations	-Find effective ways to organize data in novel situations.	**ACM:** Computational Thinking **CAS:** N/A
Curiosity (CU)	-Ask general and specific questions about the nature and scope of tasks -Feel motivated to explore new challenges	- Encourage creative and novel thinking	
Confidence (CO)	-Readily approach tasks with unfamiliar content -Exhibit increased self-esteem	-Circumventing gender stereotypes in later career paths -Encourage creative thinking and collaboration with others	

Complete Goal Specification

In order to specify the goals for this course, we organize goals by main transfer goal as well as knowledge type. The following key is used to denote type of goal:

Type of Goal	Cognitive	Metacognitive
Knowledge	1	4
Skills	2	5
Disposition	3	6

Each goal is associated with a transfer goal, type of goal, and a number in the format TransferGoal.GoalType.Number (e.g. AB.2.1 denotes a cognitive skill goal under abstraction). Not all transfer goals will have items corresponding to each field. Specifically, the transfer goals curiosity and confidence focus on dispositional goals, whereas the remaining transfer goals focus on knowledge and skill building.

General Computer Science and Coding (G)

Code	Type	Full Specification
G.1.1	Cognitive Knowledge	**Define** the term 'computer science:' field of study relating to principles of development and use of computers *Students will understand that computer science is a field that extends beyond writing code. The field is centered on using computers to solve problems in the real world.*
G.1.2	Cognitive Knowledge	**Form** a model for a computer scientist's role *Students will understand that a computer scientist uses several different frameworks for thinking to solve problems. These frameworks or approaches include organizing data, reasoning how to pass information, writing sets of specific procedures (algorithms), and manipulating information. These computational thought processes are key tools in a computer scientist's toolbox*
G.1.3	Cognitive Knowledge	**Establish** a common vocabulary for communicating *Students will be come familiar with proper terminology associated with computational thought processes: data, data structures, algorithms and debugging. Using these common*

		terms will enable students to communicate their computational ideas more concisely
G.1.4	Cognitive knowledge	**Familiarize** applications of computer science: *Students will understand that computer science, while a developing field, is being applied to several important areas including medicine, biology, chemistry, physics, social media, search, and economics*
G.1.5	Cognitive Knowledge	**Understand** that computers use code and procedures as instructions for a computer *Students will understand that computers use a set of specific instructions, written in various languages, to interpret instructions from humans to perform new tasks.*
G.1.6	Cognitive Knowledge	**Understand** that computer code has to be specific and explicit. *Students will understand that instructions that make sense to humans often don't make sense to computers - computers only understand very specific, quantitative commands. Students will understand that they need to think through how something is done in a lot of details in order to get a computer to do it.*
G.1.7	Cognitive Knowledge	Gain basic **understanding of a function**: code that does something according to specification, potentially on a variety of inputs.
G.2.1	Cognitive Skill	**Comfortably write small programs in Python (built up to throughout course)** *This goal will be referred to throughout the course, but at different stages it may mean different things. For example, in initial lessons, it may mean copying existing code and changing a few values, while in later lessons it refers to constructing loops and data structures. Because coding is not a primary goal of the course, we do not breakdown potential learning goals.*
G.3.1	Cognitive Disposition	**Feel comfortable** opening and running programs. *When given a task in python, students will be able to readily fill in statement, resolve syntax errors and bugs, and identify the appropriate point to ask for help. They will be able to run scripts from either a graphical user interface or using execute commands in the terminal on the computers provided for the course.*

G.3.2	Cognitive Disposition	**Understand** that computer science extends beyond just writing code *Many students may enter our outreach program with the predisposition that computer scientists just write code. We aim to reconstruct this mental model to include what this code actually does to solve problems*
G.3.3	Cognitive Disposition	**Build** patience through tracing for a bug *Student will understand finding a problem takes patience as it requires a correct identification of the problem. The process of identifying the problem takes time and sometime many trials.*
G.3.4	Cognitive Disposition	**Build** perseverance through trial and error *Student will understand that fixing a problem requires a strong mindset to try many times. Student will also understand that from the failure they can learn, and with enough trials they will succeed.*
G.4.1	Metacognitive Knowledge	**Know** a general approach to breaking down problems in computer science *Students should understand how to break down a problem into available data needed processes (algorithm design)*
G.4.2	Metacognitive knowledge	**Know** strategies for identifying strengths and weaknesses (bugs) that make a problem easy or difficult *When given a problem in English (not in code), students should be able to identify what elements of that problem might make it more challenging to solve by a computer scientist*
G.5.1	Metacognitive skills	**Know** strategies for dealing with unfamiliar coding tasks *Students will know how to identify whether their problems are conceptual or syntactic. For conceptual problems, students will use general problem solving techniques; for syntactic problems, students will interpret error messages of look online. Students will develop personal techniques for dealing with these kinds of problems.*
G.5.2	Metacognitive Skills	**Analyze** what is required to solve a problem *Given a problem to solve, students will analyze what result is needed and what information is given. Students will realize if they are missing information necessary to solve the problem.*

G.5.3	Metacognitive Skill	**Adjust** student approach based on evaluation of alternative ways of solving a problem *Student will be able to consider alternative solutions and choose the one that best fit their updated knowledge.*
G.6.1	Metacognitive Disposition	**Feel confident** that computer science is an accessible field *Many students may enter with a predisposition that computer science is too challenging. Students' general knowledge of computer science should assert that this is a reasonable and accessible field to them.*

Algorithmic Thinking (AT)

Code	Type	Full Specification
AT.1.1	Cognitive Knowledge	**Define** algorithm as a sequence of instructions to accomplish a goal *Student will understand that there are smaller building blocks of completing a larger task. They will recognize what they consider an easy task could in fact be harder when viewing at a smaller scale*
AT.1.2	Cognitive Knowledge	**Understand** algorithm efficiency as how much "work" (instruction step) involved in each algorithm *Student will understand that there are multiple ways of doing a problem and that each way has a different requirement and needed effort.*
AT.1.3	Cognitive Knowledge	**Recognize** the need of algorithm due to limit in computational processing power and the massive amount of data (large problem) *Student will understand that computer processing is constrained by time and the amount of data it has to work through. Then, they will understand the significance of algorithm that has been worked and developed before by other computer scientist*
AT.1.4	Cognitive Knowledge	**Understand** the use of algorithm to search/modify data *Student will understand that algorithm creation was motivated by data, and that without data algorithm does not have a purpose.*

AT.1.5	Cognitive Knowledge	**Recognize** that algorithm is a tool that may be used on different data *Student will recognize algorithm as an abstract tool that may be reused with the correct requirement/constrains*
AT.1.6	Cognitive Knowledge	**Understand** bubble sort and selection sort as examples of commonly used sorting algorithm
AT.1.7	Cognitive Knowledge	**Identify** the efficiency of selection sort in **comparison** with bubble sort
AT.1.8	Cognitive Knowledge	**Contrast** a specific algorithm (like searching for a word in dictionary) with a general algorithm (like binary search algorithm) *Student will understand the link of using an algorithm as an abstract tool in solving specific problem. This is helpful in creating value of the abstractness of the algorithm for student.*
AT.2.1	Cognitive Skill	**Identify** example of algorithms in everyday life (e.g., searching in dictionary for a word, searching through clothes for outfit, etc.) by finding a sequence of instruction to accomplish a goal
AT.2.2	Cognitive Skill	Give clear and specific instructions both orally and for computers
AT.2.3	Cognitive Skill	**Create** a simple algorithm using natural language *Student will understand the main idea of algorithm as creating instruction and the creation of these instruction is independent of language, be it natural language or computer programming language*
AT.2.4	Cognitive Skill	**Apply** general algorithm to specific data sets
AT.2.5	Cognitive Skill	**Read** algorithm in form of python code
AT.2.6	Cognitive Skill	**Recognize** where algorithms are used in real-world contexts
AT.2.7	Cognitive Skill	**Apply** known algorithms in new problems.
AT.2.8	Cognitive Skill	**Design** a new algorithm or procedure outside of code *Students will be able to use spoken language or computer code to design a set of steps to perform a task. Language will include detailed steps (the how, what, and where) and pseudo code will resemble control flow (if/when), the what (data or variable elements) and key descriptions of sequence or iteration (linear vs. repeating)*

AT.3.1	Cognitive Disposition	**Know** that there is some set of steps that I can use to fulfill the requirements *Given requirements for solving a problem, students will know that there is some set of possible steps to come up with (post-instruction) instead of providing a direct answer (pre-instruction)*
AT.5.1	Metacognitive Skill	**Plan** on using the most effective algorithm known to solve problem
AT.5.2	Metacognitive Skill	**Monitor** the effectiveness of their instruction by asking the question “Is there another way of doing this?”

Debugging and Troubleshooting (DT)

Code	Type	Full Specification
DT.1.1	Cognitive Knowledge	**Understand** the historic naming of “bug” *Student will understand why the term bug is used and appreciate the work of Grace Hopper. They will understand that deep inside the culture of this field exists a term contributed by a female computer scientist.*
DT.1.2	Cognitive Knowledge	**Define** bug as logic error, syntax error, misuse of predefine code or anything that cause students’ instruction to not work.
DT.1.3	Cognitive Knowledge	**Understand** debugging as a process of finding problems in student’s code
DT.2.1	Cognitive Skill	**Find** a bug in a code example
DT.2.2	Cognitive Skill	**Trace** a function call by identify where it was “defined” and where it was “called” in Python
DT.2.3	Cognitive Skill	**Test** code to ensure it perform to specification by asking “what is the code designed to do?” and “what is the result of executing the code?”
DT.6.1	Metacognitive Disposition	**Believe** that with enough trials one can fix any problem. *Student will build confidence and personal belief in attempting to solve problem. From such belief, such then can build patience and perseverance.*

Abstraction (AB)

Code	Type	Full Specification
AB.1.1	Cognitive Knowledge	**Understand** the terms specific and general in problem solving. *Specific means a specific answer to a specific question, general refers to an approach that can be used for many problems. Students will be able to distinguish between a specific answer and a general answer that goes with it. They will know that general solutions can be modified to fit specific problems.*
AB.1.2	Cognitive Knowledge	**Understand** the difference between interface and implementation *Interface refers to functions/code for which one only knows the mapping of inputs to outputs; implementation refers to how the code is actually written. Students will understand that when they call a function they did not write, they are using an interface; code they did write is an implementation.*
AB.1.3	Cognitive Knowledge	**Understand** that on a certain level instructions they write are already implemented. *In the course, students will have to write their own code as well as call implemented functions; however, all the commands they write have been implemented on a lower level that the machine can understand. Students will understand that even the basic commands they write are an abstraction for more complex commands that the computer can follow.*
AB.1.4	Cognitive Knowledge	**Explain** the importance of abstraction. *Students will understand that abstraction is important because it can make code simpler. It allows programmers to focus on specific aspects of a program and not be concerned with implementing other parts. Furthermore, general solutions can help one solve multiple problems at once.*
AB.1.5	Cognitive Knowledge	**Understand** that certain parts of procedures can be abstracted/hidden. *If a certain implementation is unknown or irrelevant, one may only need to know what output it produces. Students will understand that abstraction is used to let different systems communicate, without all the programmers needing to know*

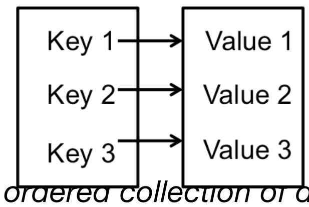
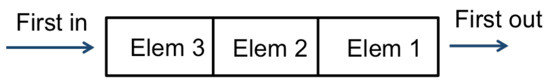
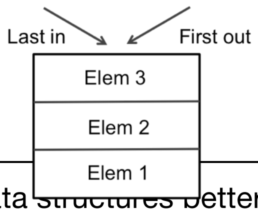
		how everything is implemented.
AB.1.6	Cognitive Knowledge	**Understand** why functions are reusable. *Contrast with procedures, where one explicitly specifies each step. Students will understand that functions are sets of procedures that can be used multiple times without explicitly "copy-pasting" them.*
AB.1.7	Cognitive Knowledge	**Understand** that specific instructions or solutions can connect to more general ones. *During learning events in the course, students will come up with solutions for specific problems. While those solutions will be relevant for the problems they are trying to solve, students will realize that these solutions are related to more general problem-solving approaches. Knowing the generalizations will help students approach new problems in the future that share common elements.*
AB.2.1	Cognitive Skill	**Identify connections** between general procedures and specific ones. *Students will be able to match general solutions to matching specific ones. The generalization skill can be used in many contexts.*
AB.2.2	Cognitive Skill	**Distinguish** when one would care about implementation versus the functionality *When you are creating a program or trying to solve a problem, all parts of it need to be implemented or the code/solution won't work. However, when trying to solve a new problem, everything does not need to be implemented from "scratch". In the context of the program, this means that students are getting starter code, while in the real world this means using coding libraries that have already been developed. While it may be interesting to learn about how this code is implemented, one only needs to know the functionality (high-level descriptions of functions) to use it. One only needs to know how to implement the new code that one is writing. While we don't want to discourage curiosity, we want students to learn to distinguish what aspects of a problem have been solved (knowing functionality) and what they need to solve (knowing implementation). Being able to prioritize in this manner is a very relevant skill in real-life contexts.*
AB.2.3	Cognitive Skill	**Explain** a situation with multiple levels of detail. *After being introduced to a process, students will explain it with*

		multiple levels of detail. Asking students to use varying numbers of steps to explain the process or asking them to "hide" specific parts of the implementation can test this.
AB.2.4	Cognitive Skill	**Identify** the same procedure discussed on different levels. *Students will recognize the same process being described with varying levels of detail. Students will identify the similarities through steps being discussed in various levels of details. Students will identify similar procedures based on the outputs they produce.*
AB.2.5	Cognitive Skill	**Define** a general procedure based on several specific ones. *Students will take several examples of specific procedures and create a general procedure based on them. Students will find identify the common elements of specific procedures and use them for the general solution.*
AB.2.6	Cognitive Skill	**Identify** the use of abstraction in an **unfamiliar** context. *Students will identify when interfaces are being used in new contexts. They will identify that a process can be split up into several parts and certain parts may be hidden. Students will also recognize when a general process can be implemented in different specific ways.*
AB.3.1	Cognitive Disposition	**Feel comfortable** approaching new ways of thinking *Students may have never dealt with abstractions before, but they will learn to be comfortable with the new method of thinking. Students will learn to be comfortable learning new approaches to thinking even if they don't pick up on the method right away.*
AB.3.2	Cognitive Disposition	**Learn** to see the big picture. *By learning about abstraction, students will learn how specific implementations connect with general ideas; this skill can transfer to their overall learning in the course: students will see that the skills that they use for specific applications are general problem solving skills.*
AB.4.1	Metacognitive Knowledge	**Know** what strategies one can use when solving abstractions problems. *Students will know what specific tools they have for solving problems that deal with abstraction (connecting general and specific problems). For example, they will know specific techniques for finding a general solution for a set of specific problems.*

AB.4.2	Metacognitive Knowledge	**Know if a solution is general enough** *Students will be required to come up with a general solution for a specific set of tasks and evaluate if the solution is appropriately general. Students will recognize if the solution they came up with only works for the specific examples given or if it can be used with other kinds of examples. Students will distinguish what aspects of a problem they can be specific about and what aspects of their solutions should be more general.*
AB.5.1	Metacognitive Skill	**Plan** to use **abstraction** in problem solving *Students will identify if making connections to more general problems is helpful for the problem at hand. Students will plan to use interfaces to hide parts of problems they don't need to consider.*
AB.5.2	Metacognitive Skill	**Evaluate** if abstraction is helping the problem get solved *Students will identify if using abstraction is getting them closer to solving the problem: certain problems may rely on specific circumstances and trying to make general connections will not help solve them. Students will identify if a part of the problem that they "abstracted away" actually needs to be specified.*
AB.5.3	Metacognitive Skill	**Evaluate** if all kinds of connections between general and specific problems were considered *Students will learn to analyze all aspects of a problem. This is a primarily a "debugging" strategy for problem solving: if students cannot recognize a connection between a specific problem and a general one, they will use this skill to figure out what aspects they have not considered.*
AB.5.4	Metacognitive Skill	**Reflect** on the generalizability of their strategy (their code) to solve the problem
AB.6.1	Metacognitive Disposition	**Know** when an appropriate level of abstraction has been reached *Students will learn when to stop developing a general approach (satisficing), even though a more general solution may be possible. In programming, it may often be possible to come up with a more elegant solution, but at times it is appropriate to stop with a solution that works for the given tasks and move on to a new problem.*

Data Representation (DR)

Code	Type	Full Specification
DR.1.1	Cognitive Knowledge	**Define** data *Students will be able to define data using appropriate computer science terminology – being able to distinguish that data is information. It can be presented as numbers (most common in computer science), or trends extrapolated from groups of numbers.*
DR.1.2	Cognitive Knowledge	**Understand** the importance of appropriate data organization *Students will know that in real-world problems, data can increase quickly in complexity. This makes it challenging for a scientist to interpret the data. Therefore, it is key to find new, appropriate ways to organize data to enable the analyst to extrapolate new trends. Students can demonstrate sufficient understanding of this using an example (application) or in general terms.*
DR.1.3	Cognitive Knowledge	**Identify** examples of data organization in everyday life *Students will be able to identify examples of data organization in real-world contexts. For instance, in a grocery store, different fruits (the data) are organized into bins (organization). The skill of identifying instances of organization will help students connect to prior context when organizing new, more complex data structures into new formats.*
DR.1.4	Cognitive Knowledge	**Learn** common basic data structures: Lists (stacks & queues) and dictionaries *Students will be able to describe various simple formats for organizing information, largely surrounding common data structures in computer science, including a list, stack, queue, dictionary, and/or graph. These simile formats will define 1-2 features of the data at most such as ordering (sequential) and pairing (dictionary).* *Students will be able to describe the properties of these data structures in words and draw a simple example:* • ***List:*** *ordered collection of data.* *[Elem1, Elem2, Elem3]* • ***Stack:*** *ordered collection of data where the last element in is the first element out.*

		 • ***Queue:*** *ordered collection of data where the first element in is the first element out.*  • ***Dictionary:*** *Pairs of data elements* 
DR.1.5	Cognitive Knowledge	**Know** that some data structures better organize datasets than others (data-structure fit) *Students will know that some data structures are more appropriate for datasets than others based on the number of features, pairing, sequential or non-sequential ordering, and general complexity*
DR.1.6	Cognitive Knowledge	**Know** that a graph data structure relates data connected in multiple ways (analogous to a network) *Students will be able to identify the graph data structure as a format of organization that relates pairs of elements where elements can be part of multiple pairs. This is similar to a network or 'group of connections'. Students will also understand that graphs are appropriate for organizing data that might not be sequentially organized in a straightforward manner.*
DR.1.7	Cognitive Knowledge	**Define** vocabulary for communicating features of the graph data structure: edge, vertex, connection, cycle *Students should be able to describe the following elements of a graph:* • *Edge: connection between two data points* • *Vertex: A single data point*

		• *Connection: an edge describing the relationship between two data points* • *Cycle: A pattern in a graph where you can reach a single vertex by following a circular path of vertices*
DR.1.8	Cognitive Knowledge	**Understand** that graphs have various applications including social media *Students will understand how graphs enable data scientists to represent large social networks in popular platforms such as twitter and Facebook. Students will demonstrate sufficient understanding by identifying the 'people' as nodes, relationships as 'edges', and groups of friends as 'cycles.' Students will use language like 'My friend is connected to me by an edge in a graph on twitter'*
DR.2.1	Cognitive Skill	**Represent** the same information using different formats. *Students will be able to organize data with one ore more features into different representations. Students are only expected to know the data structures described earlier, but are encouraged to also be creative with their organizations (i.e. try new organizations).*
DR.2.2	Cognitive Skill	Given a set of simple data, **identify** an appropriate data structure *When provided a set of data, students should be able to identify a data structure that allows an analyst to extract out appropriate features. For example, if data is organized into a dictionary, one can extract a relationship between the keys and values provided. Students are only expected to know the five data structures taught in these lessons, but creativity is encouraged.*
DR.2.3	Cognitive Skill	**Justify** choice of data structure for a set of data *Students should be able to defend their choice of data structure based on the features of the data. For example, a list was selected because the order of the data only matters in 1 dimension.*
DR.2.4	Cognitive Skill	**Write a simple program** using the data construct *Given a python script with most of the data structure implemented, students should be able to complete the implementation (where appropriate), add, and remove data elements from the data structure*

DR.2.5	Cognitive Skill	**Identify** data that can be organized into a graph *Find another example of data from everyday life that can be organized as a graph. For older students, forces between atoms in a solid are a good example.*
DR.2.6	Cognitive Skill	**Sketch** a graph from a list of data points and connections *Given a list of data points and connections, students should be able to draw the data points and the right connections between each vertex.*
DR.2.7	Cognitive Skill	**Identify** connections and nodes in a graph *Given a drawing of a graph, students should be able to identify edges as elements that connect data points. Students should also be able to identify nodes as data points.*
DR.2.8	Cognitive Skill	**Make connections** with past data structure usage *Students should be able to connect* *Students should be able to connect data structures to their prior knowledge of organization in everyday life. Connection to prior knowledge will strengthen understandings. Examples below:* • *List: List of groceries to buy at the store.* • *Queue: People waiting in line at a counter* • *Stack: 'Stack' of pancakes* • *Dictionary: Pairs of words with their definition in a real dictionary*
DR. 3.1	Cognitive Disposition	**Know** that raw output from a program is not the best way to organize data *Students will understand that data is more than just numerical values output from a computer program. They will know that manipulation and recombination of this data is useful in understanding more about the dataset.*
DR. 3.2	Cognitive Disposition	**Include** organization of data as a step in the overall set of cognitive tasks associated with generating data when solving a problem.
DR.4.1	Metacognitive Knowledge	**Understand** that a conscious choice of data structure will be required for the best data organization

		Students will intuitively know that some data structures are better than others, but overall a conscious choice of data structure given the features of the dataset and requirements for the problem will enable extrapolation of valuable information.
DR.4.2	Metacognitive Knowledge	**Know** that the data structures a student currently understands about may still not be the best way to organize data *Know that optimal data organization extends beyond simple data structures. Students know they can select a data structure they know about, perform searches for new structures, or creatively invent their own.*
DR.5.1	Metacognitive Skill	**Suggest** formats for new data *Students should be able to identify features of a data structure to satisfy data in a more complex problem. For example, 3 dimensional data might require a data structure that has 3 columns (inferring a matrix)*
DR.5.2	Metacognitive Skill	**Plan** organization of data *Given a set of data, students should know to first identify common features and reason how this data will be reassembled in a data structure (prior to performing the organization)*
DR.5.3	Metacognitive Skill	**Evaluate** poorly organized data *Students should be able evaluate when their data does not fit into a correct data structure and iterate toward an alternate solution.*
DR.6.1	Metacognitive Disposition	**Confidently** iterate through a challenging organizational problem *When data does not fit into the originally chosen data structure, students should be confident that an alternate organization would still lead to a reasonable solution.*

Confidence (CO)

Code	Type	Full Specification
CO.3.1	Cognitive Disposition	**Volunteer** suggestions on unfamiliar topics

		Students will apply what they know to make suggestions about questions on unfamiliar topics. Students will acknowledge that while their suggestions may be wrong, educated guesses can be valuable in a conversation.
CO.3.2	Cognitive Disposition	**Develop** confidence in creating algorithm for daily life activities
CO.3.3	Cognitive Disposition	**Feel comfortable** presenting to a group *Students will be able to present their work and their ideas both in a small group setting and in front of the class. Students will feel ready to contribute to small group discussions. For class presentations, students will feel comfortable about presenting after practicing ahead of time.*
CO.3.4	Cognitive Disposition	**Feel comfortable** writing new code. *Students will feel ready to approach new coding tasks when they are presented in class. Students will know that they have been taught what they need to know to implement the assignments. If students do not know how to implement something, they will know how to get help.*
CO.6.1	Metacognitive Disposition	**Know** you are well equipped to solve a problem *Students will know that they have learned the problem-solving skills necessary to approach a variety of challenges. If students run in to problems, they will know how to debug/troubleshoot or get more help.*
CO.6.2	Metacognitive Disposition	**Be unafraid** to solve problems despite lack of prior knowledge. *Students will know that they can solve problems that appear unfamiliar. Students will develop techniques to help them tackle problems they do not know a lot about.*
CO.6.3	Metacognitive Disposition	**Feel comfortable** evaluating your approach to a problem and adjusting it if necessary. *Students will be comfortable reflecting on whether the approach they have chosen for a problem is working. If an approach is not working, students will be ready to adjust it. Students will learn to focus on trying new things instead of focusing on failed attempts.*

Curiosity (CU)

Code	Type	Full Specification
CU.1.1	Cognitive Knowledge	**Know** that being curious is a fundamental element of problem solving *Students will know that asking new questions and thinking in alternate directions is key to problem solving. They will know it is ok to question concepts that they do not understand or topics they want to learn more about.*
CU.2.1	Cognitive Skill	**Find** alternative application of principles/definitions *Students will be able to expand main course concepts by searching for additional technical details and connecting to new real-life applications. They might find these by asking questions or demonstrating initiative in their own internet searches and thinking.*
CU.2.2	Cognitive Skill	**Search** for a solution or implementation Without teacher or volunteer initiative, students will search the internet or ask questions about an implementation they are unsure about. They will apply appropriate technical terminology in their search, demonstrating curiosity specific to the technical content in a correct manner.
CU.4.1	Metacognitive Knowledge	**Know** that you can ask 'why' and consider improved ways to approach problems *Our classes will be made up of activities and group discussions. Initially, we will ask our students 'why' type exploratory questions about activities. As the course progresses, students will ask similar questions without prompt. Students will*

		understand that activities can be expanded and connect to real-world applications.
CU.4.2	Metacognitive Knowledge	**Know** that you can do something differently *Moving beyond goal CU.4.1 students will not only be curious about how to expand on class activities, they will also know that they can perform a task using an alternate strategy. This metacognitive knowledge goal encourages students to apply confidence to the context of solving new problems. By knowing there are alternate approaches, students will be able to take on a broader range of challenges. This goal is key because it will encourage students to move beyond lessons and try new things.*
CU.5.1	Metacognitive Skill	**Monitor** approach to a problem and adjust approach in creative ways. *Students will learn to troubleshoot their problem-solving approach in creative ways. Students will ask themselves "What have I not tried?" and "Can I combine solutions I've seen before in creative ways?"*
CU.6.2	Metacognitive Disposition	**Be** curious about a new problem *When students approach a new task, they will want to know the solution.*

Assessment

Overview

Our outreach program aims to create an informal learning environment. In alignment with this, we will implement no formal assessments (exams, quizzes, etc.). Instead, each lesson is designed with increasing rigor as students progress. This will allow students to gain independence and confidence with what they are learning. The assessments will be embedded into the learning events, because completion of the in-class activities will correspond with mastery of goals. Small-group and whole class discussions will also be an informal assessment of understanding. These are general examples of formative assessments used in the class. The full breakdown is specified in the next section.

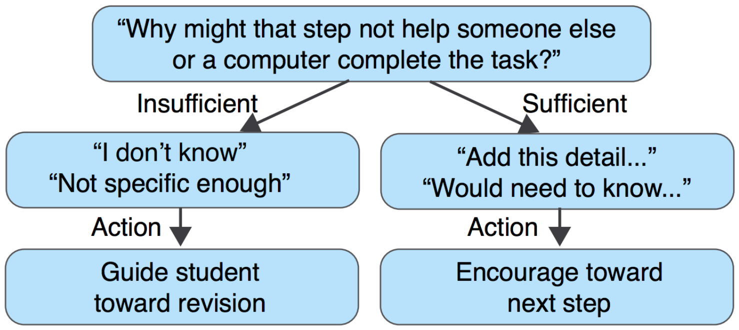
- Instructors will ask questions during lessons to gauge understanding and promote further thought
- Students will demonstrate mastery in algorithm design by individually and collaboratively creating algorithms in both code and on whiteboards
- Students will demonstrate understanding of abstraction by decomposing different problems into algorithms and new methods of data organization
- Students will demonstrate debugging skills by taking a systematic approach to problems they encounter, and asking questions when appropriate.
- Students will have opportunities to exercise creativity and curiosity during group brainstorming sessions (asking questions is a metric for curiosity)
- Students will demonstrate confidence, persistence and independence when working on coding activities and problem solving challenges.
- Students will demonstrate ability to work in groups during the variety of group activities during class.

The class will have one summative assessment in form of a group project completed in the last two sessions. These projects will be open-ended and encourage students to use skills learned throughout the course. The purpose of this assessment is to test for transfer, because students will be applying skills to new problems. At the same time, this assessment is personal, because it allows students to apply computer science skills to a topic that they are personally interested in.

In general, these assessments reflect both on the program and the students. The students' understanding and completion of in-class activities will reflect how they are meeting the goals. At the same time, because students will work closely with volunteers, volunteers can judge how well the instruction is helping the students meet the goals. For each lesson, volunteers will be presented with a set of guiding questions to discuss with their students; the volunteers will be taught to use these questions to assess student understanding.

Guiding Questions

Each lesson incorporates Guiding questions, which are introduced during group discussions. These questions help instructors and volunteers to evaluate students understanding of content while encouraging continuous exploration of a topic. Volunteers will be provided examples responses and criteria for assessment to gauge student understanding from these questions. Below is an example sequence used during the first session:


"Why might that step not help someone else or a computer complete the task?"
Insufficient
Sufficient
"I don't know" "Not specific enough"
"Add this detail..." "Would need to know..."
Action
Action
Guide student toward revision
Encourage toward next step

Justification

Our assessment philosophy is based on a number of principles from our Big Ideas:

- **Low-Pressure Environment:** All assessments are informal and volunteers assist students during assessments. As a result, students will not feel pressured to "perform perfectly." Our goal is to encourage students to try new things and do their best (**Aligned with Goals: CO, CU**)
- **Metacognitive Skills:** As the class progresses, students will be required to apply metacognitive strategies. For example, retest and revise during debugging or providing follow up questions during discussion sessions.
- **Understanding Learners in Context:** Because assessments are informal, volunteers will understand student differences on a deeper level. The assessments will focus on the individual students so we can better accommodate their learning.
- **Feedback Opportunities:** Because students work closely with volunteers, they can receive frequent feedback on their progress. Students will have many opportunities for discussion where volunteers can correct and make suggestions, enabling students to improve.
- **Focus on the Essentials**: While many activities involve coding, assessments are focused on key concepts. Students will not be penalized for problems related to syntax and volunteers will help students complete all activities.

Instruction

Overview

When

This will be an after-school program run during a semester of school (fall or spring) for 10 weeks. Students will meet once a week for two hours with a snack break in the middle of the session. The program will sum up to about 20 hours of instructional time.

Learning Environment

The program will accommodate 25-30 girls; it will have one or two instructors, who run the sessions, as well as 8-9 volunteers. We will aim to have a 1:3 ratio between the volunteers and the girls to maximize individual attention. We will assign a volunteer to track the same small group of students throughout the course, both to help with the assessment and promote a stronger mentor-mentee bond. If the same volunteer tracks the same students throughout the course, they will be better able to assess growth.

The program will be run in a classroom set-up with tables that allow for small group work. Depending on the program for the day, the tables may be moved out of the way for more interactive activities or laptops will be set-up for coding activities. It will be important for the classroom to be set-up in such a way that the students can both pay attention to a presenter and switch to working with a small group.

Mentors and Instructors

We note the importance of forming strong positive relationships with students early in sessions. We envision a learning environment where students, instructors, and volunteers foster a mentoring relationship. Our instructors and volunteers are female university students majoring in science, engineering or computer-science, and are therefore well-positioned to act as role models to younger learners. Basic training may be given to the volunteers to help them work with the students effectively. Instructors and volunteers will create this environment by:

- Sharing their knowledge and experiences with students
- Exhibiting enthusiasm for their respective field
- Demonstrating a positive attitude

Volunteers will be present at each session to assist with activities, guide learning and serve as experienced mentors to students. These volunteers will likely be undergraduate or graduate university students interested in STEM outreach both with and without previous outreach or teaching experience. Prior to each session, volunteers will attend a 30-minute training in which the guiding questions, themes, and activities are explained for this session. Volunteers are directed on how to ask guiding questions to maximize constructive learning, facilitate interactions between students in group activities, encourage participation and make assessments.

Single-gender learning environment and mentorship

We chose to create a single-gender learning environment based on two key factors. First, research suggests that women learn better when information is presented in a narrative or contextualized form. Second, many women in STEM often feel less confident and under prepared based on technical skills when entering the field. By creating a single-gender learning environment, we can accommodate for both of these factors by presenting lessons with narrative and providing women a safe environment to grow technical skills and confidence at an early stage. Programs that model this learning environment include STEM programs at the Ellis School, TechNights and Carnegie Mellon and TechBridge.

While we were unable to identify specific research justifying the introduction of female mentors, we have found that there is a substantial precedent for a single-gender mentorship model. Less then 50% of high school girls know a woman in a STEM career. Therefore, we believe female mentors will help to close this gap. Other STEM outreach programs also follow this paradigm. For example, 93% of girls who complete TechBridge, a computer science outreach program in California, note that female role models have increased their interest in pursuing STEM careers. We also believe that inclusion of female mentors will counteract negative stereotypes about women and STEM and model positive, confident behaviors for younger students.

While we recognize these techniques may extend to a co-gender learning environment, we believe maintaining a single-gender learning environment is the best way to address gender-specific dispositions in STEM.

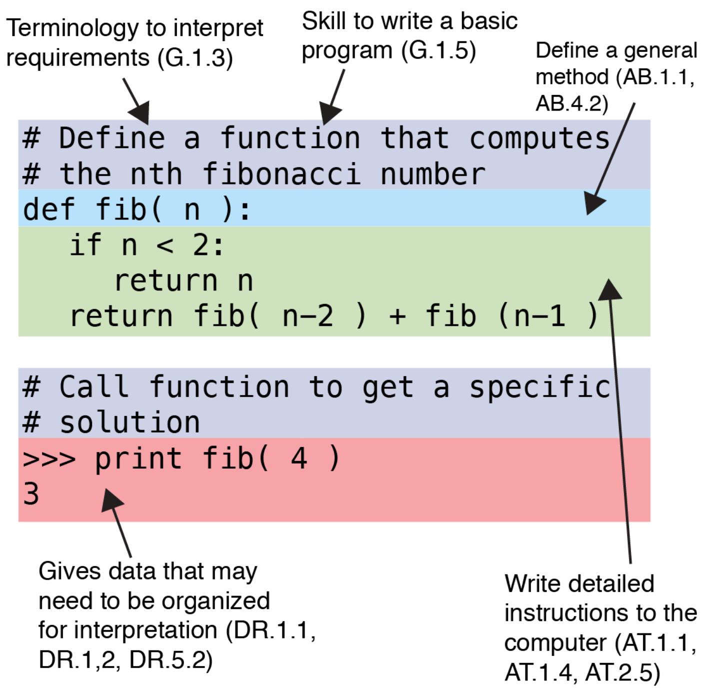
Overall Organization

The lessons will teach both big ideas in computer science and interesting applications. The lessons are centered around our transfer goals. We will focus on teaching each concept for two weeks in context of various applications, but we hope the general ideas from each lesson will apply throughout the course. The class will culminate with a final project that will allow students to explore their interests and show off what they have learned.

Sequence

We determined the order of lessons over the course of the session by breaking down a common computer science task, writing a function, into its most basic components. Below is an example coding task for computing numbers in the Fibonacci sequence:

To execute this task, the individual needs to (1) have the vocabulary to interpret the requirements (purple), (2) know how to write a specific procedure with an abstract definition (blue), (3) understand the inputs and outputs (data), (4) write a specific procedure that manipulates the data, and (5) troubleshoot.
The ordering of ThinkTech lessons directly mirrors the metacognitive process described above for approaching problems in computer science. This not only allows leaners to build knowledge in an appropriate and logical sequence, but do so in a way that is ecologically valid (realistic).


Terminology to interpret requirements (G.1.3)
Skill to write a basic program (G.1.5)
Define a general method (AB.1.1, AB.4.2)
Gives data that may need to be organized for interpretation (DR.1.1, DR.1,2, DR.5.2)
Write detailed instructions to the computer (AT.1.1, AT.1.4, AT.2.5)

```
# 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
```

Lesson Structure

Each lesson will consist of presentations, discussions and interactive activities. The lessons will typically begin with short presentations to introduce the new content. These will be needed to introduce new vocabulary and concepts, but may also include discussions. The majority of each session will be devoted to an interactive activity. The interactive activities may include coding, physical activities and games. The specific activity will depend on the theme of the week and the bigger lessons it focuses on. At the end of each session, we will have a discussion to review the key concepts covered that day.

Multiple Contexts for Instruction

While the ability to write computer code is fundamental to a career in computer science, much more fundamental is the ability to communicate computational ideas in multiple contexts. Many technology interviews require students write out steps of algorithms on whiteboards or speak aloud the steps. Group problem solving sessions require students to present and be criticized on their ideas. For this reason, we aim to build instruction that both teaches students how to write code, but more so how to think in multiple contexts and scenarios. This is supported in lessons through the use of group discussions, presentations, interactive drawing and construction activities, and games to supplement coding tasks.

Overall Justification

Our design relies on a number of our Big Ideas from our individual big ideas sections:

- **Engaging Instruction:** Because ThinkTech is an outreach program; we aim to make all our instruction fun. We want girls to be constantly engaged in activities that they can learn from.
- **Constructivism:** Many of learning events are focused on problem-solving challenges and follow-up discussions. This model allows us to have the students' think of many of the important concepts on their own.
- **Connecting Abstract and Concrete:** In each lesson, we combine computational thinking skills with computer science applications. This model enables students to envision the importance of these concepts in solving real world problems.
- **Variety of Representations:** Each lesson combines a number of interactive and coding activities. This provides students with multiple opportunities to learn the material, specifically if they are struggling with coding activities, they have the opportunity to learn the concept from interactive activities.
- **Personalized Feedback and Motivation:** Because students will work closely with volunteers, they will receive ample help with activities. Volunteers will also learn about individual student backgrounds to appropriately motivate them.

=

Syllabus Level Instruction Outline

Session 1 - **Introduction**
Learn about the field of computer science and the typical role of a computer scientist. Learn about applications and uses of computer science in every day life and begin using terminology important to the field.

Session 2 - **Procedural thinking and Abstraction 1**
Learn about computers using explicit and specific instructions. Begin to understand the difference between specific and general instructions. Work in Python to create randomly generated art.

Session 3 - **Abstraction 2**
Learn about how the internet works and message passing protocols. Learn about interface and implementation and identify the difference. Define abstraction and understand its importance.

Session 4 - **Data Structures 1**
Learn about the importance of organizing data using different representations. Learn about common data structures including lists, stacks, queues and dictionaries and apply these representations to organizing simple datasets.

Session 5: **Data Structures 2:**
Learn about why computer scientists might need more complex representations for their data. Learn about the graph data structure, its importance in social networks (and networks in general), and apply knowledge to building a human social network graph for the class.

Session 6 - **Algorithms 1**
Learn what is an algorithm and the need of efficient algorithm. Student will have a few opportunities to identify algorithm to solve problems and then "count" the number of steps taken to achieve each solution. Through practice, students would acquire the understanding of what is an algorithm. Through application, students will generate meaning of why algorithms are used in real life context.

Session 7- **Algorithms 2**
Learn about developed algorithm and its usage.

Session 8 - **Debugging and Troubleshooting:**
Learn what is a "bug" and how to ensure the code is doing what it is specified to do

Session 9 - **Final Project Session 1**
Students will be asked to form groups and work on final projects. For the final project, students will research a topic related to computer science and in the next session present to the class about how the project is related to the transfer goals: Abstractions, Algorithms

and Data Structures. Volunteers will help students find topics of appropriate scope and guide them through the research process.

Session 10 - **Final Project Session 2**
Students will complete work on the final projects started in session 9. In teams, they will present their findings and new understandings to the other students in the course, instructors and parents. In the second half of the session, students will be introduced to new resources that will enable them to continue computational skill development and exploratory learning after the course has concluded.

Lessons

Session 1: Introduction to Computer Science

(designed by Rebecca Alford)

Lesson Overview

The first session orients students to the field of computer science. They will learn about what it means to be a computer scientist from a panel of undergraduate and graduate student volunteers working in various applied fields. Students will also start writing their own programs in the Scratch environment to create animations. These animations will be used to introduce students to the class and create a comfortable learning environment for future sessions between peers, instructors, and volunteers. Finally, students will translate their Scratch code into pseudo code towards writing real computer programs.

The following essential questions guide students thinking (**Big Ideas, R-G5**):

- What is computer science? What does a computer scientist do?
- How do I communicate with a computer? How can I use it [the computer] to solve exciting problems?
- How can I use my own creative ideas to working on interesting computer science tasks?
- How do I write a real computer program?
- What does computer code look like? What are common features?

Summary of Assessments

- **Preliminary Assessment**: What is Computer science? Discuss preconceived notions about the field of computer science including how to write code, what its used for, and what a computer scientist's role might be (Goals - N/A - pre assessment)
- **Presentation of Scratch "Introducing Me" Activities**: Students will present their Scratch blocked code to introduce themselves to the class. They will demonstrate understanding by describing how specific code blocks come together to create the animation (**AB.2.1, G.1.3, CO.3.2, CO.3.3**, **CU.6.1**)
- **Translating Scratch code into pseudo code**: Students will demonstrate their understanding of control flow, functions, and data on a novice level by translating their scratch code into pseudo code on a whiteboard (**AB.2.1, G.1.5, G.2.1, G.3.1**)
- **Post-Discussion** Students will discuss what they learned in the session – how did their perceptions change of the field? Are they using appropriate terminology? Can they volunteer answers demonstrating comfort and confidence in the learning environment? (**G.1.1, G.1.2, G.1.3, CO.2.2, CU.6.1**)

Summary of Learning Events

- **Group Discussion**: Instructors will lead a discussion about what computer scientists do and what types of problems are solved in the field. During this discussion, instructors will introduce ideas such as thinking in procedures and using data to answer big questions (**Big Ideas, R-I4**)
- **Introducing Me – In Scratch** Students will use the Scratch coding environment for building animations to make videos introducing themselves to the class. The scripting environments will allow them to add their own creative spin on the activity, while also learning how to assemble specific procedures for a computer.
- **Translating for all computers** Students will learn how computers interpret instructions from humans – using computer code. Students will learn how to translate the instructions provided by Scratch into pseudo code, a key transfer task towards writing python code in future sessions (**Big Ideas, R-G6**)
- **Current topics in CS**: Volunteers will briefly discuss the types of real-world problems they address using their computer science knowledge and skills. Topics include computational biology, search and prediction, big data analytics, social media, finance, and math (**Big Ideas, R-I4**, **consider differences in learner motivation and context**)
- **Concluding Discussion**: Instructors, volunteers and students will reconvene to discuss what they learned about the field of computer science. What was new? How is what you learned different from what you thought?

Module Alignment

Alignment between goals and assessment

Assessment	G.1.1	G.1.2	G.1.3	G.1.5	G.1.6	AB.2.1	CU.6.1	CO.3.2	CO.3.3
Preliminary Assessment	x	x	x					x	x
Presentation of Scratch Code			x	x		x	x	x	x
Translating Scratch code to pseudo code			x	x	x	x	x	x	x
Post assessment	x	x	x	x	x		x	x	x

Alignment between goals and instruction

Instruction	G.1.1	G.1.2	G.1.3	G.1.5	G.1.6	AB.2.1	CU.6.1	CO.3.2	CO.3.3
Group discussion (pre)	x	x	x					x	x
Introducing me – in scratch			x	x	x	x	x	x	x
Translating for all computers			x	x	x	x	x	x	x
Current topics in CS	x	x	x						
Group Discussion (post)	x	x	x					x	x

Alignment between instruction (row) and assessment (column)

	Preliminary Assessment	Presentation of scratch code	Translating Scratch to pseudo code	Post assessment
Group discussion (pre)	x			
Introducing me – in scratch		x		
Translating for all computers		x	x	
Current topics in CS*				
Group Discussion (post)	x	x	x	x

*Note: The current topics in CS panel is presented to accommodate for learner differences, particularly motivation and interest. It does not align specifically with course goals, but we believe it aligns with our learner context

Key Assessments

Activity 1: What is Computer Science? - Use a group discussion setting to brainstorm definitions for computer science.

Broad Description	The goal of this assessment is to determine students' familiarity with common ideas in computer science. This activity is the first in the session and the entire course. Students will likely be unfamiliar with peers and instructors. The group setting provides a familiar, comfortable environment and allows students to meet new faces during the discussion. Students will arrange themselves in a large circle of chars around the room. To encourage equal participation in the discussion, students will be asked to answer guiding questions instead of participating voluntarily.
Cognition	Students will demonstrate their understanding of the following big ideas in computer science: (1) The field of computer science is the study of how we can use computers to solve real-world problems (2) A computer scientist uses computational tools such as data analysis and writing programs to solve these challenges. (3) Computer science is an integral part of everyday life. At this point, students have received no formal instruction. This preliminary assessment is intended to survey individual differences between learners. Specifically, determine prior knowledge and comfort level in the instructional setting Specific Goals: **G.1.1, G.1.2, G.1.4**
Observation	Volunteers and instructors will be asked to ask guiding questions described above. If students ideas do not align with key ideas, volunteers

	will ask further specified guiding questions to help them consider why their prior knowledge might not align *The following essential questions will guide the discussion:* • What is computer science? (**G.1.1**) • What are the typical roles and responsibilities of a computer scientist? (**G.1.2**) • Where might you encounter computer science in your experiences? (**G.1.4**)
Interpretation	Comparing answers to definitions specified in the learning goals will assess level of prior knowledge. Degree of alignment with these definitions will be used to assess the level of prior knowledge for individual learners. Providing examples is also sufficient. *Sufficient Understanding* • Computer science: use computers to solve real-world problems, use computers to perform lots of calculations, generate big data • Computer scientist: Analyze data, write programs to solve problems, organize information *Insufficient Understanding* • Computer Science: write code to solve problems, write code, do math • Computer Scientist: Writes code
Design Standards	**Valid:** This assessment is valid because the criteria are based directly on definitions stated in the learning goals. The explicitness of these definitions ensures there is no ambiguity in their interpretation. Students who do not understand the material will not be able to provide sufficient answers to the discussion questions. Comparatively, students with a sound understanding of material will provide correct answers. **Reliable**: Volunteers will be trained to identify sufficient and insufficient responses to guiding questions prior to the session to ensure reliability in scoring. The assessment is also reliable because students would provide the same answers given no instruction between repeated questioning (where instruction includes other students in the group providing the correct answer) **Objective:** We will provide volunteers with various examples of correct and incorrect responses to ensure volunteers have a sufficient understanding of objective scoring criteria. We will also provide volunteers opportunities for feedback to refine objectiveness of assessments.
Justification	**Alignment:** Key questions posed to students derive directly from general computer science knowledge goals **Stability:** Providing a set of guiding questions for instructors ensures the same level of prior knowledge is being probed across different learners

	Critical Learning Environment: The assessment is somewhat challenging because it is the first assessment in the sequence. Students might be uncomfortable in the classroom due to lack of familiarity, which confounds the validity of this assessment. Volunteers will be told to make note of these differences during the activity. **Metacognitive Focus:** Early discussion sessions encourage students to learn to think openly and expansively about knowledge. This begins to build on key transfer goals of curiosity (specifically CU.4.1 and CU.4.2) and confidence speaking and presenting.

Activity 2: Presentation of Scratch Introducing Me activities.

Broad Description	Students will apply their understanding of procedural thinking skills and practice confidence by presenting their Scratch Animation to the class. This assessment is the second activity in the instructional sequence after the first lesson. Its purpose is to acquaint students to the learning environment and gauge understanding of procedural thinking, a skill that will be fundamental throughout the course. At this point, students will have already completed their animation in Scratch using the blocked scripting environment. Students congregate with instructors in a large group and be provided the following prompt: *Introduce yourself to the class using your Scratch animation. Identify the 2 sprites "scripts", changes in control flow, sounds and visuals added to make your animation. Tell us how your steps come together in order to make the animation and one lesson you learned from the experience.*
Cognition	The purpose of this assessment is to determine students' ability to apply procedural thinking skills to a familiar context. Presenting their ideas to the class will help evaluate students' understanding of required steps, basic ability to create a program, and confidence to present new ideas and curiosity for future activities. This assessment will assess the following goals: **AB.2.1, G.1.3, CO.3.2, CO.3.3**, **CU.4.1**
Observation	Volunteers will observe student presentations and ask appropriate guiding questions to provide immediate feedback. Volunteers will specifically monitor for appropriate use of terminology (G.1.3), discussion of specific steps in the animation (AB.2.1), and confident presentation and sharing of ideas Volunteers will use the following guiding questions for each student: • What types of details or specific steps did you need to add to get

	your animation to work? (AB.2.1) • How are steps ordered in your animation? (G.1.3) • Why did you choose the specific ordering? (CU.4.1) • What would you do differently next time (CU.4.1) • Could a computer understand your instructions? What is not detailed enough? (G.1.3)
Interpretation	Instructors and volunteers will use the following criteria to evaluate student presentations. *Sufficient* • (AB.2.1) Student describes detail elements of blocks, such as data/information and action/verb • (AB.2.1, G.1.3) Student describes control flow using appropriate language (sequence or “in order” for linear processes, “looping” or “repeating” for iterative processes) • (AB.2.1) Student is able to distinguish between general and specific. Student can identify specific details that make the procedure specific (surface level understanding) • (CU.4.2) Student can suggest alternate approaches or changes to the script • (CO.3.2) Student is able to discuss all criteria of the presentation, with or without reminders by questioning (i.e. what were your control blocks?) *Insufficient* • (AB.2.1) Student only describes the generic process and cannot identify specific steps (where a step is a block in the Scratch code) • (G.1.3) Student does not use appropriate language in discussion of blocks (i.e. “Numbers” instead of “data or information”) • (CU.4.2) Student cannot suggest changes to the script • (CO.3.2) Student is unable to proceed through the entire presentation criteria, even with guidance from volunteers and peer support Relevant Big Ideas: (**R-A2, alignment with instructional goals, R-A5, balance qualitative and quantitative**)
Design Standards	**Validity**: Guiding questions and scoring criteria are aligned directly with goals of procedural thinking and use of appropriate language, as well as goals for confidence and curiosity. If students do not know terminology (G.1.3) and cannot demonstrate specific steps (AB.2.1) demonstrated by their language, they cannot perform well. Confidence and curiosity are more challenging to assess in a valid manner, therefore completion of the presentation and being able to answer expansive questions at all are the most valid metrics for this introductory activity.

	Reliability: Scoring criteria designates specific wording for volunteers to look for that distinguish between sufficient and insufficient understanding. This makes the assessment repeatable by multiple assessors. Students also would not be able to answer the questions better unless they were taught specifically how to interpret the Scratch blocks between assessments. **Objective:** We will provide volunteers with various examples of sufficient and insufficient metrics to practice and understand how to objectively evaluate students. Volunteers are also specifically asked to not evaluate students on the complexity of the scripts or how 'colorful' the animation is which could bias the scoring and evaluation.
Justification	**Alignment:** Key questions posed to the students as well as the prompt align with the goal of exposing students to computer science terminology and procedural thinking **Stability:** The assessment is stable as all volunteers will be trained to provide the same level of guidance and feedback to students. However, it will be challenging to make this perfectly consistent **Create critical environment:** At this point in the course, students are just beginning to become comfortable in the classroom. For this reason, volunteers and instructors are allowed to provide assistance during the presentation to accommodate for initial discomfort. **Opportunities for Feedback:** Very early in the course, students gain the opportunity to receive feedback from instructors on their presentations and ask/answer questions while presenting. **Metacognitive Focus**: Incorporation of confidence and curiosity into presentation criteria by requiring students to expand on their knowledge builds on metacognitive goals from early on.

Activity 3: What is computer science? - Verification through Post-discussion

Broad Description	The goal of this assessment is to gauge students' understandings of **G.1.1** and **G.1.2** as well as increased comfort level in the classroom after the first session. The class will reconvene in a large group discussion setting and discuss what they learned about "What is computer science?" and "What does a computer scientist do?" In the post assessment setting, students will be allowed to volunteer information. This will allow volunteers to begin to build and encourage confidence answering questions and affirming choices in a group setting – a key gender-specific goal of the course. This assessment reflects big idea **R-A3, implement comparative assessments**

Cognition	Volunteers will assess student answers to the same guiding questions. However, expected answers for a sufficient and insufficient understanding will be expanded to reflect instruction and informally measure confidence. This assessment will specifically assess **G.1.1, G.1.2, G.1.3, and CU.3.3**
Observation	Volunteers and instructors will guide a large class discussion using the following guiding questions and searching for specific terms and concepts introduced in the first session. Volunteers will provide real time feedback and expansive questions. For example: Volunteer: “How were your procedures specific?” Student: “They had specific steps” Volunteer: “What details were included in those steps?” In this setting, even though the student provided an insufficient answer, the volunteer used the student’s answer as a scaffold toward the corrected answer and to construct a new mental model from the less correct or detailed one. *The following essential questions will guide the discussion:* • What is computer science? (**G.1.1**) • What are the typical roles and responsibilities of a computer scientist? (**G.1.2**) • Where might you encounter computer science in your experiences? (**G.1.4**)
Interpretation	Instructors will use the following criteria to assess sufficient and insufficient understandings from students: *Sufficient:* • Computer science: allows manipulation/changing of data with procedures, create, build, complex processes, solve real-world challenges using procedures (algorithms even better!), address challenges in building those procedures • Computer scientist: uses code as a language to communicate with the computer, instruct computer to perform complex tasks, variety of applications including biology, economics, search, and media *Insufficient* – Reflects sufficient and insufficient understandings from the pre-assessment Informal assessment of confidence: The instructor will call on students who volunteer information but also note which students are comfortable volunteering information. To provide

	feedback or responsive instruction, instructors and volunteers will encourage participation from less confident students.
Design Standards	**Validity:** The assessment is valid because it requires students to use terminology and understandings presented directly in the instruction. Students who do not understand the material would not be able to use the correct vocabulary/language, as indicated in the scoring criteria. The assessment is also directly aligned with course goals for general computer science understanding. **Reliability:** Volunteers will be trained prior to the session to identify sufficient and insufficient responses for stable scoring. The scoring criteria also mentioning specific language allows multiple scorers to evaluate based on the same criteria. Because the scoring criteria mentions specific language aligned with the course goals directed during instruction, students could not perform well if re-tested without this direct instruction. **Objective:** We will provide volunteers with various examples of sufficient and insufficient metrics to practice and understand how to objectively evaluate students.
Justification	**Informal, Low Risk Environment** – The group discussion enables students an opportunity to demonstrate their understandings by answering questions, but also have a safe environment to make mistakes and receive real-time, corrective feedback **Metacognitive Focus** – Informal, comparative metric for confidence building assesses students' comfort in the learning environment and align with course key transfer goals of confidence and curiosity **Focus on Essentials** – Leading a discussion with key course ideas, including a fundamental understanding of procedural thinking and the role of a computer scientist, enables students to focus on skills and understandings **Pre/Post Test** – Comparative assessment by pre and post discussion allows volunteers to measure effectiveness of instruction and better determine student understandings compared to prior responses.

Key Learning Event: *Introducing Me* with Scratch

Context: This introductory learning event is the second instructional event in the first lesson. Students' only knowledge of computer science will be their prior knowledge, and provoked curiosity from the introductory discussion section. Because this is the first activity, Students will likely not be comfortable in the learning environment. This activity will also be the first 1:1 interaction between students and undergraduate volunteers.

Specific Objectives:

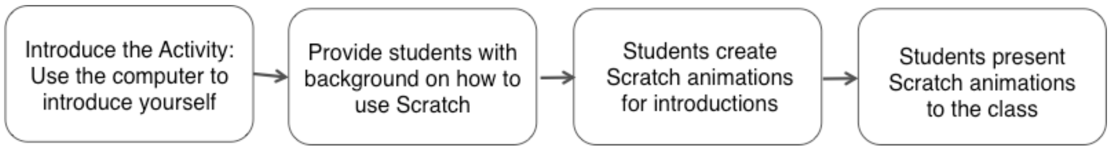
- Apply a new skill, procedural thinking, in a familiar context – creative computer drawing and introducing yourself (**AB.2.1**)
- Practice using specific instructions to communicate a task to a computer (**G.1.5**)
- Practice using new computer science vocabulary: data, loop, debug, procedure, sequence (**G.1.3**)
- Become comfortable asking questions in the learning environment (**CU.2.2**, **CU.3.2**)
- Confidently present your choices to group of students (**CO.3.2**, **CO.3.3**)

Beyond these key learning objectives, there is also a set of objectives for this activity, specific to this learning event because it is one of the first learning events in the entire course trajectory:

- Form a positive relationship with volunteers and instructors. Gain trust that volunteers are invested in creating a comfortable and respectful environment for learning (**Big Ideas: R-TR1, R-TR2**)
- Instructors and peers get acquainted – familiarize themselves with backgrounds, interests and similarities (**Big Ideas R-TR2**)

Outline of the Activity:

In this activity, the instructor will introduce students to the idea of using a computer to perform tasks. The instructor will explicitly direct students to use the computer to introduce themselves by creating animations that include 5 random facts about themselves, modeled after the Scratch "5 random Facts about me" activity. Students will work independently to create their animations with the guidance of volunteers. Students will then present their instruction to the class.


Introduce the Activity: Use the computer to introduce yourself
Provide students with background on how to use Scratch
Students create Scratch animations for introductions
Students present Scratch animations to the class

Materials, Resources and Technology

- A laptop computer with the appropriate software will be available for each student (1:1 ratio)
- The webpage for Scratch will already be loaded

Steps of the Instruction

1. Introduction to computer based instruction

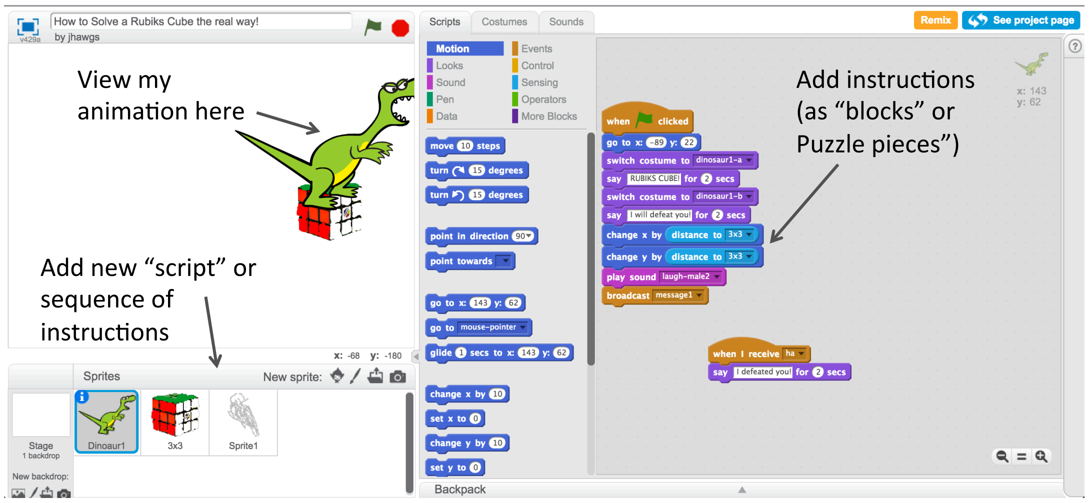
Students will learn how and why computers need to take specific instructions (**Big Ideas: R-G3)**

- Computers need very specific instructions to perform tasks (**AB.2.1**)
- Familiar context: Point-and-click on the computer – **speak aloud to decompose task**
    - “Clicking on the Microsoft word icon tells the computer to open Microsoft word, make the window the full size of the screen, create a new document with no text, and give it a temporary name of *Document 1*”
    - Connect the details of the task to the general importance of details when communicating with a computer
- Called a ‘procedure’
- Other contexts: Computer scientists write code – lots of different languages
    - Familiar context: different spoken languages, also different languages to communicate with computers (**Big Ideas: R-G4 appropriate prior knowledge**)
    - Examples: Python, C++, Ruby, etc.
    - Some are visual, some just text

2. Introduction to the Scratch Platform

Instructors will introduce the Scratch Platform (visual or block based programming, developed by MIT Media Labs) to students and walk them through an example animation.

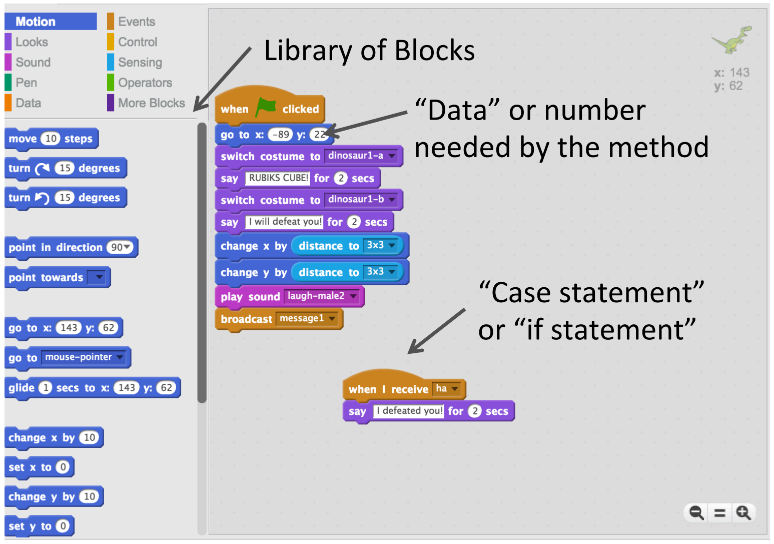
- Scratch is a language that uses blocks to send instructions to the computer. The blocks **fit together like puzzle pieces** to make a new procedure
- Students from around the world use Scratch to make and share animations by coding (**CU.2.2, CU.3.2)**
- We will use Scratch to make animations to introduce ourselves to the class
- Provide example animations and how students plug the blocks together. Using example, how to solve a Rubik’s cube: http://scratch.mit.edu/projects/36818296/?fromexplore=true#editor
- Demonstrate different features of the Scratch platform and connect to previously discussed concepts
    - Start with high level description to orient students



```
when [flag] clicked
go to x: -89 y: 22
switch costume to dinosaur1-a
say RUBIKS CUBE! for 2 secs
switch costume to dinosaur1-b
say I will defeat you! for 2 secs
change x by distance to 3x3
change y by distance to 3x3
play sound laugh-male2
broadcast message1

when I receive ha
say I defeated you! for 2 secs
```

- Describe specific details of the block ‘scripts’ or ‘programs’

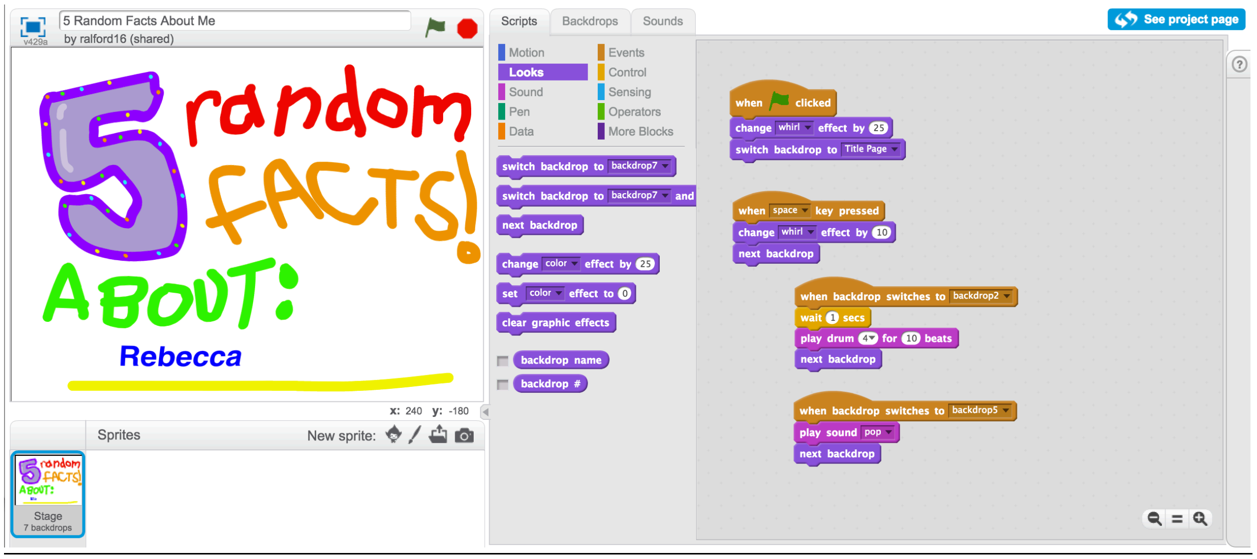


```
when [flag] clicked
go to x: -89 y: 22
switch costume to dinosaur1-a
say RUBIKS CUBE! for 2 secs
switch costume to dinosaur1-b
say I will defeat you! for 2 secs
change x by distance to 3x3
change y by distance to 3x3
play sound laugh-male2
broadcast message1

when I receive ha
say I defeated you! for 2 secs
```

3. Provide students instructions for the activity

- Create your own version of the Scratch Animation, The Five Random Facts about me – using the scripting/block tools to change the order and mode in which your visual presentation is presented to the class (**CO.6.2, CO.6.3, CO.3.4**)
- You must use:
    - 2 different ‘sprites’ (scripts)
    - 2 control flow changes (loop, reverse, if statement)
    - 2 sounds
    - 2 “looks” tools (visuals)
- Note on Instruction: Volunteers will be present to make sure students do not spend too much time coloring the text on the slides and most time working on the actual scripts. This is extra precaution to ensure the validity of this instructional activity
- Example “5 Random Facts About Me” Project: http://scratch.mit.edu/projects/38132886/



```
when [flag] clicked
change whirl effect by 25
switch backdrop to Title Page

when space key pressed
change whirl effect by 10
next backdrop

when backdrop switches to backdrop2
wait 1 secs
play drum 4 for 10 beats
next backdrop

when backdrop switches to backdrop5
play sound pop
next backdrop
```

4. Students work on the activity guided by volunteers
Students will work independently on the activity but desks will be arranged in groups to encourage peer interaction (**Classroom Arrangement – Ellis School**)**.** Volunteers will help to answer questions both on how to use scratch and mediate whether students meet the criteria of the assignment.

5. Student Presentations to the Class (Assessment step – see above)

<u>Justification and Alignment</u>

Aligned with learner differences and context:

- **Age Range**: Scratch designed to accommodate all ages
- **Individual differences**:
    - *Visual preference:* blocked instructions connect directly to animations in real-time, shapes of blocks communicate importance of specificity
    - *Non-visual preference:* text contained in the blocks also informs specificity for instructions
    - *Patience*: Adolescents ages 11-14 are known to loose patience when completing difficult activities. Volunteers will be present to accommodate this difference for students.
- **Gender-Specific Differences**:
    - *Narrative-based instruction*: Girls prefer a narrative or story to be present. The 'introducing me' story interweaved with the specific instructions provides this narrative (**Ellis School**)
    - *Peer-interaction*: Arrangement of desks in groups allows students to work independently but also become comfortable with their peers to encourage a **positive and comfortable learning environment**

Alignment with Goals and Course Principles

- Volunteer presence and interaction allows students to receive **timely feedback and clarification**
- Instructors are **explicit** about learning objectives in teaching procedural thinking (**Big Ideas R-G3**)
- **Builds from prior knowledge**: Most students are familiar with the task of introducing themselves and creative drawing on the computer (possibly using power point, paint, or other tools)
- **Key terminology** is explicitly used and consistently reinforced in multiple contexts (**Scaffolding)**
- **Creates a critical learning environment**: Accommodating for individual differences and providing opportunities for creativity strengthens dispositional goals such as volunteering ideas and becoming familiar with instructors, peers, and course dynamics **(Bain)**.
- **Encourages creativity (Big Ideas, R-I4)** Students are allowed to learn a procedural skill while also making their own independent choices about sequence, visuals, and sound. This aims to fill a key learner context component of the 'confidence gap' between women and men – **independent affirmation of choice.**

Key Learning Event: *From Puzzles to Code* (transfer task from Scratch to Python) (**Big Ideas, R-G6**)

Context: This learning event occurs immediately after students are introduced and gain practice creating specific instructions that a computer will use to execute a task. They will have novice level experience using the Scratch programming language. However, students are not expected to have any formal or informal programming experience in python. Students will have had an opportunity to introduce themselves and interact with peers, instructors and volunteers, ideally building trust prior to this next task.

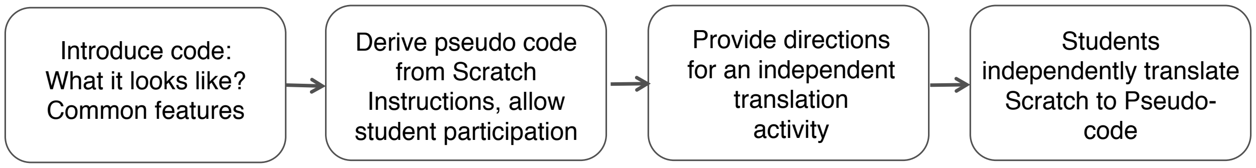
Specific Objectives

- Apply a new skill, procedural thinking, in an unfamiliar context – providing written pseudo python code that mirrors the Scratch instructions (**AB.2.1**)
- Practice using specific instructions to communicate a task to a computer (**G.1.5, G.2.1, G.3.1**)
- Practice using new computer science vocabulary: data, loop, debug, procedure, sequence (**G.1.3**)
- Become comfortable asking questions in the learning environment (**CU.2.2**, **CU.3.2**)
- Confidently present your choices to a volunteer mentor (**CO.3.2**, **CO.3.3**)

Beyond these key learning objectives, there is also a set of less prioritized but important goals for this learning event:

- Students will demonstrate initiative by volunteering to participate in the large and smaller group activity (**Curiosity, Confidence – Broad**)
- Instructors and peers get acquainted – familiarize themselves with backgrounds, interests and similarities (**Big Ideas R-TR2**)

Outline of the Activity:
In this activity, the instructor will introduce students to writing code by translating the Scratch instructions to Pseudo code on a whiteboard in real-time. The instructor will introduce the concepts of if statements, loops, variables (just as data) and general structure. The class will work together to translate the Rubik's cube Scratch code into pseudo code. Students will then independently translate their "Five facts about me" scripts into pseudocode, guided by instructors.


Introduce code: What it looks like? Common features
Derive pseudo code from Scratch Instructions, allow student participation
Provide directions for an independent translation activity
Students independently translate Scratch to Pseudo-code

Materials, Resources and Technology

- A whiteboard or smart board (something that allows the students and instructors to write in multiple colors)
- Laptops with existing Scratch activity (from previous learning events)

Steps of the Instruction

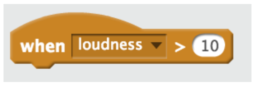
1. Introduce the idea of a ‘computer language’, otherwise known as ‘code’

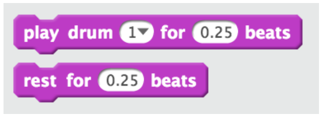
- Discuss how Scratch instructions are limited to the Scratch environment (**Connect with previous context**)
- Real computers need instructions in a different format or language.
- **Essential Question**: What is behind the blocks? (**Big Ideas: R-G5**)
    - Give students the opportunity to volunteer answers – exercise creativity and confidence in answering (**CO.3.2**, **CO.3.3**)
- Discuss the basics of computer languages – set of written, detailed instructions, given to a computer to execute tasks. Often complicated
- Example languages: Python, Java, C++
- Instructors will also survey students to determine if any have prior coding experience. This will allow volunteer mentors to accommodate to learner levels in their guidance (**Big Ideas, R-G4**)

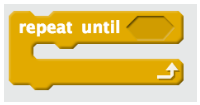
2. Introduce the idea of a ‘computer language’, otherwise known as ‘code’

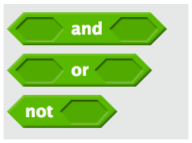
- Identify common statements used in the Scratch programming environment
    - Case statements – “when”
    - Loops/cycles – “while case, do this”
    - Actions - “play drum”
- For each example block, write out what this might look like in pseudo code. Identify the **key words** and the **format (be explicit with requirements)**

Element	Block type	Code

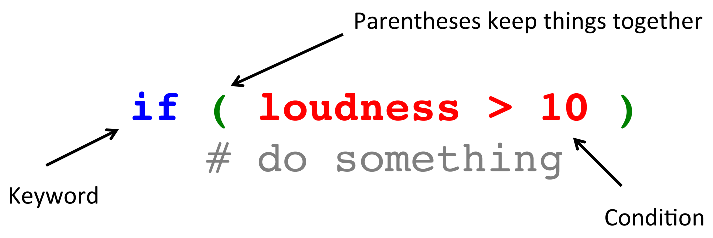
Case (“if”)	 when loudness > 10	`if (loudness > 10 )`
		`    # do something`

Function	 play drum 1 for 0.25 beats rest for 0.25 beats	`play_drum( type, num_beats )`

Loop	 repeat until	`while ( case )`
		`    # repeat action`

Operators	 and
	or
	not
		`if ( ‘this’ && ‘that )`
		`  # do something`

Identification of keywords and formatting of pseudo code:



```
if ( loudness > 10 )
    # do something
```

3. Opportunity for guided practice

- The instructor and students will work together as a group to translate the Rubiks cube script to pseudo code on the whiteboard.
- Students will take turns translating lines, receiving real-time feedback from their instructors and peers (**CO.3.2**, **CO.3.3**)

4. Individual practice – Translate your 'Five Random Facts' Into Psuedo code

- Students will be given individual whiteboards (1 per student) and several colored expo markers. They will be instructed to translate their Scratch script into pseudo code. Students will be asked to identify the keywords in their psueodo code and add comments describing what their code does.
- Volunteer mentors will provide real-time correction, feedback and encouragement during the activity. Specifically:
    - Using a color coding system to describe different parts of their code
    - Determining if the instruction in the block matches the pseudo code

<u>Justification and Alignment</u>

Aligned with learner differences and context:

- **Individual differences**:
    - *Syntax:* Grades 6-8 are a time in which syntax and grammar are continuing to be encouraged but still imperfect. For this reason, we use pseudo code instead of actual python to allow students to focus on functional features of the code instead of syntax (**removing confounds**)
- **Gender-Specific Differences**:
    - *Narrative-based instruction*: Instruction on pseudo code and formatting is still within the previous context of the 'Introducing me' activity (**Ellis School**)
    - *Peer-interaction*: Working collaboratively helps students to feel not alone in solving challenging problems, creates a **positive and comfortable learning environment**

Alignment with Goals and Course Principles

- Volunteer presence and interaction allows students to receive **timely feedback and clarification**
- Instructors are **explicit** about how the pseudo code should directly derive from the blocks used in Scratch (**Big Ideas R-G3**)
- **Prioritize transfer**: While Scratch provides an accessible learning environment, ultimately computer code is written with accommodation for syntax. The learning trajectory of this lesson ultimately funnels towards the key transferable skill students will need (**Big ideas, R-G6**)
- **Builds from prior context** – Students have familiar context from 'introducing me' – minimizing load of new information.
- **Encourage students to affirm choices** – Writing an answer on the board while gaining encouragement from peers helps students to reaffirm their choices and build confidence (**CO.3.2**, **CO.3.3**)

Session 2: Procedural Thinking and Abstraction 1

(designed by Anastassia Kornilova)

Lesson Overview

This session will be a more rigorous introduction to coding and procedural thinking. Students will learn that computers need specific and explicit instructions and can not understand human commands. Students will have several opportunities to practice creating specific instructions. At the same time, students will be introduced to the first part of abstraction: specific vs general. Students will learn that while they can come up with a specific procedure to do something, this procedure can be generalized into a procedure that can be used for many tasks. Finally, students will explore the connection of computer science to art.

Students will learn metacognitive skills related to procedural thinking, and connecting general and specific procedures. For procedural thinking, students will learn to ask "Will my instructions make sense to somebody else?", "Am I making any assumptions?" and "Am I being vague because I do not understand part of the procedure?". For abstraction students will ask "What do these specific problems have in common?" and "Have I seen a different solution that may be relevant before?".

Outline of Activities

1. Initial Discussion: Students and volunteers will discuss instructions and procedures they see in daily lives.
2. Robot activity: Interactive activity to teach students about specific instructions. **(G.1.5, G.1.6, AB.3.1, AT.2.2, CO.3.1)**
3. Procedural Flower Drawing: generate explicit instructions for drawing flowers. Compare with others instructions to come up with a general set. Distinguish more specific instructions from general ones **(G.1.5, G.1.6, AB.2.1, AT.2.1, AT.2.2)**
4. Generative Art Coding: Students will work in Python to create randomly generated art. **(AB.2.1, AB.2.5, AT.2.2,G.1.6, G.1.7, G.2.1, G.3.1, G.5.1)**

5. Wrap-up discussion: Students talk in small groups about what they learned and why it was useful **(AB.1.1, AB.1.7, AB.3.2, CO.3.1)**. Volunteers will use the following guiding questions:

- Which parts of the line and the circle code were similar? What kind of function could you write to avoid copying code?
- What other modifications can you think of?
- How would you describe the procedure for any random shape drawing?

Assessments

Activity 1: Giving a robot instructions

Broad Description	Students will work as a group to instruct a "robot" (volunteer) to make a sandwich. The robot will only respond to specific and explicit instructions, so students will be challenged to come up with those. To encourage all students to participate, commands for the robot will be collected by calling on students sequentially.
Cognition	This assessment will check if students understand that computers need specific instructions. **(G.1.5, G.1.6, AB.3.1, AT.2.2, CO.3.1, CU.6.1)**
Observation	Volunteers, each in charge of several students, will note what kinds of commands their students suggest. Each student will give one or two instructions to the robot. Volunteers note whether the student suggested instructions are specific. The specificity can be evaluated on a spectrum, but volunteers will be asked to stick to a +/- evaluation for each student. *Examples of* +: "Lift your left hand 90 degrees", "Lower your hand over the piece of bread on the right side of the table" *Examples of -:* "Open the butter jar", "Take a piece of bread out of the bag". The assessment is informal because it takes place during a in-class activity
Interpretation	If students mostly give specific instructions, then they will have mastered the goal. Scoring will be based on the +/- system described above; however, because this is an introductory activity, the assessment will be used to judge the program, and how the material is presented. If most students answers get a "+" then the topic was taught appropriately, otherwise it may need to be revisited from a different angle.
Design Standards	**Valid:** This assessment is valid because it explicitly tests for the goals listed. If students can come up with specific instructions, they would have met the goal. Even though they are using natural language, instead of code, the ideas of specific thinking should transfer to coding assignments. **Reliable:** This activity is reliable because the volunteers will be given a number of examples for "+", "-" responses which will help them assess the students. If the graders aren't sure about a response, they can make a

	note of it, which will help us come up with more specific criteria in future iterations **Objective:** This assessment is objective because the “robot” will be told specifically what kinds of commands to respond to. Thus, the robot’s actions will only depend on how specific or general instructions are. The rest of the volunteers will be given clear rubrics to assess student responses with.
Justification	Giving Practice Opportunities and Feedback (Instruction B): Students get immediate feedback to the commands they suggest - based on how the robot reacts. They will have several opportunities to contribute commands, so they can adjust their approach. Connect Abstract Ideas and Concrete Examples (Instruction B): Earlier in the class, students will be told that they should come up with specific procedures; in this task they will get to try this concept out in the specific context of a robot making a sandwich.

Activity 4: Modifying Generative Art Code

Broad Description	Students will be given Python code that when run draws random lines on a screen, it will feature a “draw random line” method. The instructor will go over how the code works, and students will practice running it. They will be encouraged to run the code and modify the variables related to speed and number of lines. After students feel comfortable running the code, the instructor will introduce the next part of the activity (this assessment). The instructor will ask the students to create a new method that will draw circles instead of lines. No more explicit instruction will be given, but volunteers help the students whenever they need it.
Cognition	Specifically students will show that they understand that certain parts of their code are specific and others are general, and they will be able to identify and re-use instructions that they are familiar with. **G.1.6 G.1.7, G.2.1, G.3.1, G.5.1, AT.2.2**, **AB.2.1, AB.2.5, CU.6.1**
Observation	Students will be given code to generate random lines on the screen, they will then be asked to write a function that generates random circles on the screen. Because the students will be given the code to generate the lines, they will need to reuse it to draw the circles. If they understand the abstraction concepts, they will be able to copy the code and modify the parts that have to do with drawing the line to circle-drawing code. The assessment is natural, it is part of the in-class activities and will not be framed as an assessment. The students work will not be officially graded, but volunteers will make sure all students finish the activity. For future analysis, student code will be saved at the end of the lesson. This is the pseudocode for the activity set-up:

```
# turtle is a basic Python drawing library
import turtle

function draw_line():
 x = random(0, screen.width)
 y = random(0, screen.height)
 # pick a starting point
 turtle.setPosition(x,y)

 # pick a direction to move in
 d = random(0, 360)
 turtle.setheading(d)

 #pickRandomColor is a provided function
 color = pickRandomColor()
 turtle.setColor(color)

 #pick length of line
 length = random(0, 100)
 turtle.forward(length)

function draw_circle():
 # Your code goes here!

function create_art():
 timeStep = 5
 numLines = 25
 for x in [1,numLines]:
   draw_random_line()
   pause(timeStep)
```

Interpretation

There will not be any official scoring; however, there will be an informal evaluation based on how the students approach the assignment. Since we desire all students to complete the task, the observational data will be based on what questions the students ask volunteers and each other.
The completed activity will look similar to this:

```
# turtle is a basic Python drawing library
import turtle
#screen is defined elsewhere and represents the
#drawing surface that the computer uses

function draw_line():
 x = random(0, screen.width)
 y = random(0, screen.height)
 # pick a starting point
 turtle.setPosition(x,y)
 # pick a direction to move in
 d = random(0, 360)
 turtle.setheading(d)

 #pickRandomColor is a provided function
 color = pickRandomColor()
 turtle.setColor(color)

 #pick length of line
 length = random(0, 100)
 turtle.forward(length)

function draw_circle():
 x = random(0, screen.width)
 y = random(0, screen.height)
 # pick a starting point
 turtle.setPosition(x,y)

 # pick a direction to move in
 d = random(0, 360)
 turtle.setheading(d)

 #pickRandomColor is a provided function
 color = pickRandomColor()
 turtle.setColor(color)

 #pick length of line
 radius = random(0, 100)
 turtle.circle(radius)

function create_art():
 timeStep = 5
 numLines = 25
 for x in [1,numLines]:
   draw_random_line()
   pause(timeStep)
```

The evaluation can be broken down into two levels:
Exceeding: The students quickly see the connection between the different kinds of line drawing code. They copy over a majority of the line-drawing code (either by re-typing or copy-pasting), and if they have questions, they are related to minor syntax issues or are additional exploratory questions.

Example questions

1. *Can I change the thickness of the circle or how fast we draw them?*
2. *I am getting an error message, I do not understand.*

Sufficient: The students understand the general connection between line drawing and circle drawing; however they struggle with some aspects of

	implementation. Example questions 1. *Are we picking the center of the circle or a point from which to start tracing the circle?* *Insufficient:* The students do not see the connection between the provided line-drawing and the circle-drawing code they have to generate. They are confused about the basics of the assignment or how to create the general code (i.e. parts that apply to all random drawing). Example questions 1. *How do I pick a starting location?* 2. *What parts of the drawing do I make random?* 3. *I do not understand what is going on at all.* Volunteers will keep a simple grid with names of their students, where they can record ✓+/✓/- based on the students' understanding, as well as additional notes. It is important to note that questions related to syntax or running the code will not be evaluated yet, because students are still getting comfortable with using Python, those skills will be evaluated later. The results will be used to evaluate the program. If many of the students demonstrate insufficient understanding, then the activities do not teach the abstraction goals appropriately. If the activity fails, then the design of the abstraction sections should be reevaluated and revised.
Design Standards	**Valid:** This assessment is valid because it tests for the specified goals. In order to create the random circle code, students will need to re-use parts of the line drawing code. In particular, students will need to pick a random place on the screen to draw the circle, the analogous code is provided for line drawing, so students will need to base their new code on it. The volunteers will answer student questions in a way that aligns with the goals: for example, if a student asks about how to pick a location on the screen to draw their circle, the volunteer will prompt them to think about where they have seen similar instructions before. If the students are able to create the new method, they will have to recognize that they are reusing some of the steps from the line-drawing code (because the code will be very similar or identical). Thus it is unlikely that the student could complete the task without meeting the goals. **Reliable:** This assessment is reliable because emphasis will be given to problem-solving skills over coding skills. Comfort with coding may vary and improve, but the underlying problem-solving skills (which volunteers are encouraged to look for) should stay consistent. **Objective:** First, completing the task can be objectively assessed because the code either does what it is supposed to or not. Second, for student understanding, volunteers are given specific criteria, they should be able to interpret all student understanding equally. The volunteers will

	be directed to avoid considering prior coding ability and instead focus on the problem solving skills.
Justification	This assessment was chosen because it is a straightforward task that targets the goals. While students will see that the procedures have a common structure, they do not have to worry about explicitly identifying it yet. This assessment *Understands Students' Mental Models (Learners C)* and *Targets the Essentials (Assessment A)* because it provides a very explicit coding assignment. Trying to learn to code and problem solve at the same time can be very challenging, so the assessment aims to simplify both aspects. Although, in later assessments, students will need to create general procedures, it is important to let students practice intermediate steps. Finally, this assessment *Connects Abstract Ideas and Concrete Examples (Instruction D)* because the students will get to (literally) see the different kinds of random art they have created.

Instruction

Activity 3: Procedural Flower Drawing

Context: This activity will follow an introductory discussion about computers using specific procedures. Students will have gotten practice giving the "robot" specific instructions in Activity 2. This activity lets the students practice the same skills, reinforcing the importance of specific procedures. The second part of the activity asks students to generalize instructions to introduce abstraction.
(Sharon, you've asked in a previous draft where they actually learn to code - it happens in Activity 4. Activity 3 is an interactive "unplugged" activity. Should we introduce coding earlier in the lesson or was it just not clear where the coding happens?)

Goals Targeted: **G.1.6, AB.1.1, AB.1.4, AB.2.1, AB.2.3, AB.2.5, CO.3.1**

Specific Objectives: During this activity students will:
1. Generate a set of instructions
2. Analyze what makes instructions useful
3. Generalize a set of instructions
4. Have an opportunity to share new ideas in a group settings

Steps of activity:
1. Students will be given markers and paper and asked to draw a flower. No additional instructions will be given, but students will be encouraged to be creative.
2. Students will be given a sheet of paper and asked to write instructions for drawing their flower. They will be told to be explicit, like they were with the robot, so that a computer or another person could use their instructions. Students will be asked to work alone and the volunteers will not answer questions regarding instructions themselves. This portion of the activity is designed to give students a chance to

practice procedure-writing skills and start to develop their own ideas of good instructions.

3. The two sheets produced by the student will be stapled together with the instructions on top, then randomly redistributed. Students will be given a fresh sheet of paper and asked to draw a flower based on the new instructions without looking at the second stapled sheet. For this portion of the activity, students will just need to follow the given instructions the best they can. If a certain part of the instruction is not clear, students will be told to make the best guess.
4. After everyone has finished, students will be asked to compare their reproduction with the original. Lead by the volunteers, they will discuss in small groups than with the whole class the following questions:
    1. Which kinds of instruction were useful or not useful?
    2. What would have helped you recreate the flower more easily?
    3. What was the most difficult part of this activity?
    4. What strategies would you use for writing good instructions?
5. As a whole class, discuss why explicit and specific instructions are important. For this part of the activity, volunteers will be called to suggest ideas. At the end, the instructor will make sure that the students covered the following points:
    1. Computers can not make guesses (like the students may have needed to for their activities)
    2. Computers should always produce the same result (what if cars had random defects each time?)
6. The students will be returned their original drawings. The instructor will introduce the next step by telling the students that they want more general instructions, so that they can be more creative.
7. As a table (group of 4-5), they will be asked to come up with a procedure that could potentially be used to draw all of their flowers. Volunteers can guide groups along by suggesting that the students look over their individual procedures and see what they have in common. The students will be encouraged to breakdown their instructions in terms of different parts of the flower (stem, petals, etc.). The goal of this step is to get students to learn to generalize.
8. The instructor will bring the students together for a final discussion with the following questions:
    1. What was similar and different between the group and the individual procedures?
    2. Why would more general instructions be useful?

<u>Justification:</u> This activity is appropriate both for the learning goals and the context. It incorporates the following big ideas:

- Connecting abstract ideas and concrete procedures (Instruction C): Students will be told that procedures can be general and specific, this activity will let them explore those terms in the familiar drawing context.
- Targeting the Essentials (Assessments A): Students will be familiar with the idea of procedures and drawing; the challenging part of the assignment will be to compare general and specific aspects.
- Encouraging Generating and Constructing (Instruction A): Students will be asked to reason about what makes instructions good, as well as define the terms “general”

and “specific”. Through the discussion, they will be able to generate the answers and understand them better.

Activity 4: Generating Art Code (introduced in assessments)

Context: After creating drawing instructions orally in the previous activity, students will get to create art through code. This is the coding activity for the day, so setting-up the students with the computers and the coding task may take some time. Students may be given a break, while the volunteers set-up the computers. Because students were introduced to coding in Lesson 1, they will be familiar with the Python coding environment.

Specific Objectives: Students will create code that randomly draws objects on a screen. This will give them practice creating their own code, as well as, understanding how certain parts of code can be reused.

Steps of Activity

1. As a large group, students will be introduced to computer generated art and shown interesting projects related to this area.
2. The instructor will introduce the code for drawing random lines and go through it line by line to explain how it works. The instructor will also introduce the function for drawing a circle.
3. Students will work individually on computers to create code that will randomly draw circles.
4. If time permits, students will be encouraged to create code that draws other random shapes.
5. The guiding questions for the follow-up discussion are included in the assessments section. The most important aspect to discuss is “how can the specific functions be combined into a general random drawing function?”

Justification

This activity gives students a chance to practice their coding skills, while continuing to learn about specific and general instructions. It also aligns with the following Big Ideas:

- Targeting the Essentials (Instruction C, Assessment A): In the template given, most of the code will be filled out and clearly commented; students will only need to fill in the code directly related to the “circle-drawing”. Volunteers will be available to help out with syntax problems, so students will only need to focus on coming up with the specific procedures.
- Connecting Abstract ideas and Concrete Examples (Instruction D): Earlier in the class, students will have discussed procedures in theoretical contexts, this activity will let them create more concrete computer instructions related to art.
- Encouraging Generating and Constructing (Instruction A): In the final discussion, students will have to think about how to generalize instructions which will lead them to have a deeper meaning of abstraction.
- Presenting Content in a Variety of Representations (Instruction C): Earlier in the class, students got to create both oral and written instructions, this is activity, they had to create coding instructions. Although in all tasks students were creating explicit and specific instructions, they used very different representations to do so.

Session 3: Abstraction 2

(designed by Anastassia Kornilova)

Lesson Overview

This session builds on ideas of abstraction from the previous class. It introduces students to the idea that a functions for specific procedures can be replaced by more general ones. Students will learn that we don't always know how everything is implemented, but we can still use known results. They will understand and identify implementation and interface. Students will understand that abstraction can make coding and problem solving simple and more efficient.

Outline of Activities

1. Introductory Discussion: Review concepts from last session, reinforcing ideas about specific versus general.
2. Internet Presentation: Introduce students to how the internet works. Show students the different steps that happen between typing in a url and getting a page back. Show that different steps can be done in different ways, but other steps do not always need to know about this. Emphasize the terms *implementation, interface* and abstraction **(AB.1.2, AB.1.4)**.
3. Message Passing Activity: Students will learn to send each other messages without knowing how the messages were created. Students come up with ciphers for messages and pass them on with decoding procedures. Students will be able to communicated without knowing how certain information was encoded. They will learn that the encoding procedure was *abstracted* **(AB.1.2, AB.1.5, AB.2.2)**.
4. Instruction Matching: Students practice connecting different kinds of instructions **(AB.1.5, AB.2.1, AB.2.3)**.
5. Coding MadLibs: Students will produce a general method for filling in MadLibs **(AB.1.7, AB.2.1, AB.2.5, AB.3.2, AB.4.2, G.2.1, G.5.1, G.5.2, CO.3.4)**.
6. Final Discussion: Conducted in small groups (**AB1.2, AB.1.4, AB1.5, AB.2.1, AB.3.2, CO.3.1, CU.2.1)**
   Guiding questions:

- Can you think of other ways of generalizing the MadLib's problem? Did you have to use any interfaces when writing the MadLib's problem?
- How would you explain abstraction?
- Why do you think abstraction is important to computer scientists?

Assessments

Activity 4: Instruction Matching

Broad Description	Students in small groups will be presented cards with different kind of procedures described. They will be asked to create sets that describe the

	same procedure, but use various levels of abstraction.
Cognition	Students will understand that procedures can be described on different levels with certain specifics hidden away. **(AB.1.5, AB.2.1, AB.2.3)**
Observation	Small student groups will take a large set of cards and separate them into piles according to what they describe. Volunteers will review the students solutions and may ask them questions about their choices. *Examples sets of cards:* • “Have breakfast”, “Get in car, drive to a dinner, order pancakes”, “Fry up eggs, make toast, eat in kitchen”, “Grab a granola bar and coffee, rush out the door to the bus” • “Communicate information to somebody”, “Open computer, open gmail, click create new message, type something, click send”, “Post on blog, send link”, “Write out letter, take to post office”.
Interpretation	The results will be evaluated based on how students sort the cards. Student solutions can be grouped into the following categories: *Completely Solved:* Students group all the cards together correctly *Main errors in vertical alignment*: Students fail to connect general and specific, but see the connection between procedures where certain steps are done differently. Example: Will group together “Start with 5, then add 3, then add 2, then multiply by 10” and “Add 2 and 3, Add to 5, then multiply by 10”, but not connect them to “For a given arithmetic problem, do all addition steps, then all multiplication steps”. *Main errors in horizontal alignment*: Students see connection between general and specific steps, but do not understand when two specific procedures connect to the same general one. Example: Will group together “Buy cake mix, frosting and icing, create cake according to instructions on cake mix box, frost and decorate with icing, bring to a party” and “Deliver a birthday cake”, but may fail to see the connection to “Acquire supplies, prepare by instruction, bring to destination”. *Mixed problems*: Students display significant errors in both horizontal and vertical alignment. This assessment will be used to evaluate the program. It will test whether the students have been sufficiently taught about connecting general and specific instructions.
Design Standards	**Valid:** In order to complete this activity, students will need to be able to identify connections between specific and general. Although some procedures may be from different disciplines, the nature of the examples

	will ensure that going by domain (cooking versus arithmetic) alone will not guarantee a completely correct solution. Furthermore, volunteers will ask students questions about their solution to make sure that the students understand the concepts and did not just guess. **Reliable:** The different subsets of cards should be equally difficult to sort, because they all rely on domains students should be familiar with. **Objective:** There is an objective solution to this challenge, and the types of errors are clearly defined. Unless, the volunteers make errors, all groups should be evaluated the same way.
Justification	*Targeting the Essentials (Assessment A):* This task is explicitly introduced as a challenge to match specific and general instructions. All the examples come from contexts that students should be familiar with, so students should not be confused by outside factors. Because volunteers work closely with the students, they can help with concerns that are not related to solving the challenge. *Presenting Content in a Variety of Representations (Instruction B):* The cards will represent tasks that have to do with a variety of contexts from sports to science. Students will see that general and specific instructions are common in all kinds of everyday life tasks.

Instruction

Activity 5: Coding MadLibs

This activity was inspired by the abstraction activity from the 20 Hours of Code curriculum (cited in the Resources section).

Context: Students will have gotten practice with abstraction in several interactive contexts and will now apply their skills to a coding activity. Because this is the first coding activity of the day, students may be given a break, as the volunteers set-up computers. Students should feel comfortable with the programming environment, because they used it in the previous two lessons.

Objective: Students will get to practice coding and learn to generalize procedures on their own **(AB.1.7, AB.2.1, AB.2.5, AB.3.2, AB.4.2, G.2.1, G.5.1, G.5.2, CO.3.4).**

Steps of Activity:

1. Students will be introduced to MadLibs. Although most students would have done MadLibs before, the instructors will go over the format to accommodate for everybody. The class will do one MadLib together. After, they will be told, that they will be to create code to fill in MadLibs automatically.
2. The students will be introduced to two new coding concepts: getting user input and printing things out on the screen. The instructor will show students how to use the **print** command to output things on the screen, and the **input()** command to prompt for input.

3. The instructor will ask students to open the first MadLibs template on the computer. It will look like this:

```
#words to ask for: noun, adjective, noun
# Text of story: “This is my first (noun). It is very (adjective). Even (noun) likes to # read it).
def first_madlib():
```

The teacher will ask for student to suggest how the problem should be approached. After collecting student suggestions, the teacher will make sure that the following steps are mentioned and outline them explicitly for the students:

1. Prompt user for every kind of words necessary [noun, adjective, verb, noun] and record them in the program.
2. Print out the lines of the story
3. When you get to a missing word, print the variable you recorded the user input in.
4. Repeat until you have printed the whole story

4. Students will work individually to create the first madlib, volunteers can help students with various coding problems they encounter. The students will get to run the code to see how it works. The result should look like this:

```
# words to ask for: noun, adjective, noun
# Text of story: “This is my first (noun). It is very (adjective).
Even (noun) likes to # read it.
def first_madlib():
 noun1 = input(“give me a noun”)
 adj = input(“give me an adjective”)
 noun2 = input(“give me a noun”)

 print “This is my first”,
 print noun1
 print “. It is very”
 print adj
 print “. Even “
 print noun2
 print “read it”
```

5. The instructor will ask students to work on a second madlib, it will be of the same format, but longer and require more inputs.

6. The instructor will ask the students, if they could have created one function to fill in both of the MadLibs. The instructor will also ask the students what possible challenges there would be with doing this. The instructor will interact with the students based on all the issues students suggest, but look for the following two concerns to address explicitly:

- We don’t what kinds of words or how many of them a general madlib needs
- We don’t know the specific lines of a new madlib or where the missing words go

Once both of these concerns come up (either directly from the students or with prompting from the instructor), the instructor will introduce a new concept of iterating over a list. Because lists will be taught more explicitly in later sessions, the instructor will encourage students to think about them as a numbered list of things. The instructor will give students the syntax for getting to the "ith" thing in the list (**list[i]** syntax). The instructor will also show students how to add things to an existing list, and how to go through all the things in a list in order (**for x in list** syntax).

7. Students will be given a new template:

```
def create_madlib(text, types_of_words):
# text is a list that contains the story, the list is
# separated around the missing words. If the madlib went
# "I like (noun) and cake", text would contain: ["I like",
# "and cake"].
# type_of_words contains the kinds of words to ask the user
# for. For the previous example, it would be ["noun"].
```

Students will work to create the general MadLibs program with the help of volunteers and students around them. This activity will be done on the computer, and students will be given MadLibs to test their code on. The final product will look like this:

```
def create_madlib(text, type_of_words):
 user_words = []
 for word in type_of_words:
   user_words+=[input("give me a:", word)]
 i = 0
 for lines in text:
  print line
  print user_words[i]
  i+=1
```

<u>Justification:</u> The program meets the following big ideas:

- Connecting abstract ideas and concrete concepts (*Instruction A and D)*: Students have previously discussed generalizing specific instructions, this tasks allows them to do that in a concrete manner by creating a general program from specific ones.
- Giving Practice Opportunities and Feedback (*Instruction B and D)*: This task gives students opportunity to practice their coding; because they can immediately run their code and ask for help, students will get immediate feedback.
- Teaching Metacognition (*Instruction C*): In the discussion steps, the teacher will guide the students through the potential concerns for generalizing the MadLib's code. The instructor will be explicit about the process and show students that for abstraction tasks one must contrast the specifics before generalizing.

Session 4: Data Structures 1

(designed by Rebecca Alford)

Lesson Overview

Information is everywhere. In fact, computers generated 80% of the world's information in the last two years alone. This session introduces students to the importance of information and finding appropriate ways to organize it. By providing a new organization, data scientists can extrapolate key trends. In this session, students will explore big data by learning about data generated in the human project (**Big ideas, R-14**) and connecting to common data structures in computer science.

Summary of Assessments

- **Introduction to Big Data**: As a group, discuss instances of data organization in every day life. Students will demonstrate understanding of large datasets by mentioning relevant and valid examples and possibly suggesting organizations. This assessment also demonstrates as a preliminary assessment to gauge learner differences and prior knowledge (**DR.1.1, DR.1.2, DR.1.3, DR.5.3, CO.6.2**)
- **Data-Structure Fit**: In small groups, students will use a decision tree like framework to determine whether a new data structure described during instruction, i.e., list, dict, stack, or queue, is appropriate for a given dataset (**DR.2.1, DR.2.2, CO.3.2, CO.6.2**)
- **Trait Organizer**: A key conclusion of genetics is that our genomic diversity results a broad variation of traits – from visual appearance and personality to susceptibility to disease. Students will be given a group of various traits and asked to try several possible organizations. Students demonstrate understanding by articulating why some data structures were better for a dataset than others (**CO.3.1, CO.6.1, DR.1.4, DR.1.7, DR.2.1, DR.2.2, DR.4.1) (Big Ideas, R-G6**)
- **Post Assessment – introduction to Big Data**: Follow up on the essential questions assessed in the preliminary group discussions. Have students' perceptions changed? Are they more aligned with course concepts? (**DR.1.1, DR.1.2, DR.1.3, DR.5.3, CO.6.2**)

Summary of Instruction

- **Group Discussion**: Students, volunteers, and instructors will discuss the importance of big datasets and the requirement for proper organization. Students will ask essential questions: (1) Why do we organize data? and (2) Why is this organization important? (**DR.1.1, DR.1.2, DR.1.3, CO.6.2**)
- **Expert Insight on Big Data**: An expert in the field of genomics will talk to students about common data organization challenges in the field. In this talk, students will also learn how proper organization enables scientists to extrapolate important information for public health and medicine (**DR.1.3**, **accommodate learner motivation**).
- **All about data structures**: Instructors will discuss types of common data structures used in computer science and key differences between them. These data

structures include a list, stack, queue, and dictionary (**DR.1.5, DR.1.6, DR.2.2, DR.2.3**)

- **Data Structure Fit with Decision trees**: Students will use a decision tree to get practice with strategies for determining the best organization for a dataset (**DR.1.5, DR.2.2, DR.2.3**)
- **Group Discussion** Students will reflect on what they learned during the session. How have their perceptions of organization changed? Why is data organization so important? (**DR.1.1, DR.1.2, DR.1.3, CO.6.2**)

Alignment between goals and assessment

Assessment	DR.1.1	DR.1.2	DR.1.3	DR.1.5	DR.1.5	DR.1.6	DR.1.7	DR.5.3
Introduction to Big Data	x	x	x					x
Data structure fit								
Trait Organizer			x				x	
Post assessment	x	x	x	x	x	x	x	x

Assessment	DR.2.1	DR.2.2	DR.2.3	CO.3.1	CO.6.1	CO.6.2	CO.3.2
Introduction to Big Data						x	
Data structure fit	x	x				x	x
Trait Organizer		x	x	x	x	x	x
Post assessment	x	x	x			x	x

Alignment between goals and instruction

Assessment	DR.1.1	DR.1.2	DR.1.3	DR.1.5	DR.1.5	DR.1.6	DR.1.7	DR.5.3
Introduction to Big Data	x	x	x					x
Expert Insights on Big Data			x					
All about data structures				x	x			
Data structure fit – decision trees	x	x				x	x	
Post assessment	x	x	x	x	x	x	x	x

Assessment	DR.2.1	DR.2.2	DR.2.3	CO.3.1	CO.6.1	CO.6.2	CO.3.2
Introduction to Big Data						x	
Expert Insights on Big Data							
All about data structures		x	x				
Data structure fit – decision trees		x	x	x	x	x	x
Post assessment	x	x	x			x	x

Alignment between instruction (row) and assessment (column)

	Introduction to big data	Data structure fit	Trait organizer	Post assessment
Group discussion (pre)	x			
Expert in Big Data				
All about data structures		x	x	x
Data structure fit – decision trees		x	x	x
Group Discussion (post)	x	x	x	x

Key Assessments

Activity 1: Trait Organizer

Broad Description	The goal of this assessment is to measure students' understanding of common data structures by organizing a set of traits (physical and personality characteristics) into custom data structures. Students must identify common features of the data such that it is organized (i.e. hair colors, eye colors, etc.) and fits into either a data structure discussed in class or a custom data structure that logically separates by feature. Students will be provided with a set of malleable wires or pipe cleaners and laminated cards with traits written on them. Trait cards will not have any identifiers, but students will be allowed to write on the cards as an aid for organization. Students will execute the activity in a small group (3-4 students) and be assessed on their ability to organize data by feature and construct an appropriate data structure (see interpretation below)
Cognition	This activity will assess students' ability to organize a larger dataset by specific features and determine an appropriate organization (i.e. data structure). This data structure can be one included in the course, or a creative, custom designed data structure that reflects a logical organization by feature. The activity assesses the following goals for the course: **DR.1.1, DR.1.2, DR.4.1, CO.6.1, CO.6.2, CU.6.1, DR.2.1, DR.2.2**
Observation	Volunteers will be asked to observe students choices for data organization and ask guiding questions specified above. Student answers to questions in alignment with specified goals for the course will be used to evaluate understanding *The following essential questions will guide the assessment:*

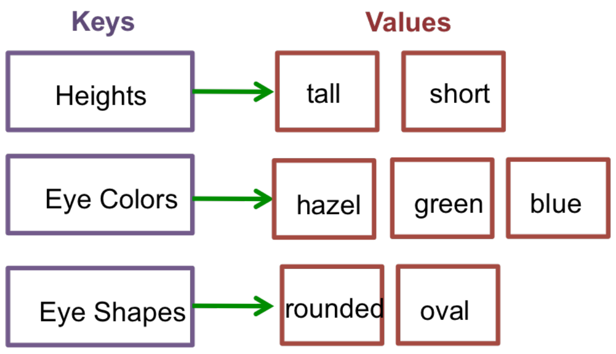
	• What are some identifying features of your data? (**identification of features**) • Which features might be the most convenient for organization? (**data-structure fit goals**) • What are different ways you can group your data? (**CU.4.2 – finding alternate approaches**) • What data structures discussed in today's section might be suitable to organize this data? • What data structure would you chose if you alter characterization of the data?
Interpretation	Volunteers and instructors will assist students in completing the activity to provide real-time feedback and guidance. They will ask guiding questions during the activity to determine if students understandings are improving and provide scaffolds for improvements. Volunteers and instructors will also assess student final products. **Scoring of Guiding Questions:** *Sufficient:* • Data is organized by 'x' feature, where a feature is some overriding characteristic of the traits. For example, hair types might be a 'feature' for curly, straight, wavy, long, short. • Multiple dimensions of features are presented where applicable. For instance, 'hair types' can be divided into 'shape'=[curly, wavy, straight] and 'length'=[short, long] • Students can identify features verbally and point them out on the physical data structure • When asked how to further detail the representation, students can search for and identify deeper levels of organization (i.e. hair types example has 2 dimensions) • Students' trial multiple organizations and evaluate if features are best divided (observation during activity) • Students' respond to feedback when asked how to better divide traits *Insufficient* • Students select the first organization that they find • No features of the data are identified • Only a single feature of the data is identified at a single level • Students cannot respond to feedback when asked how to better divide traits into groups • Students are unable to justify why choice of data structure was appropriate based on separation of traits

Scoring of Final Products
Students will be given a variety of traits and degrees of freedom to create new data structures. Below are examples of sufficient and insufficient implementations of this activity:

Sufficient Implementation
In this example, students were able to recreate a data structure discussed in class, the dictionary, to divide traits into 3 groups and then divide again by key (features) and a list of values. This student product is sufficient because it:

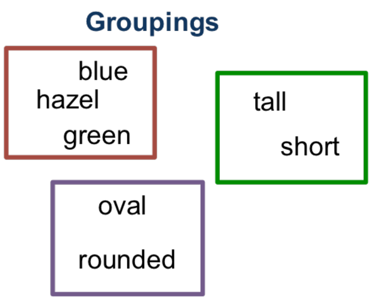
- Organizes features at multiple levels (lists, key-list pairing)
- Demonstrates understanding of the data structures discussed in class
- Also displays creativity by introduction of lists of values instead of straight key value pairing

Traits Provided: tall, short, blue green, hazel, brown, rounded, oval
Students create dictionary like data structure – keys associated with lists



Insufficient Implementation
In this example, students were only able to organize data into a single level and could not apply a data organization discussed in class. Despite instructor feedback the data is not 'organized' according to the instructional principles.

Traits Provided: tall, short, blue green, hazel, brown, rounded, oval
Students group data into simple bundles, only one level of organization shown



	Note on Interpretation: During training, volunteers would be provided with additional examples of sufficient and insufficient data representations to increase relaibiltiy and objectivity of scoring.
Design Standards	**Validity:** The assessment is valid because it directly assesses students ability to recreate data structures given simple data sets (DR.4.1), The assessment criteria for guiding question also notes language and vocabulary specific to instruction directly correlated with understanding. Students who do not understand the material might be able to identify arbitrary features, but would not be able to appropriately organize data into the data structures presented in class. **Reliability:** Volunteers will be provided with multiple examples to assess the in-progress and final products by students to determine sufficient vs. insufficient. Volunteers will also be provided with example language and ways to gauge and encourage understanding of data organization during the activity to encourage real-time feedback. Without instruction, students would not be able to produce correct and logical data structures if retested **Objective**: Volunteers will be trained to look at feature organization and similarity to data structures taught in class instead of just color choices and creativity to make assessments objective and align with goals.
Justification	**Provide multiple contexts** Students have an opportunity to practice computational thinking outside of the coding context. Accommodates for confounding challenges like syntax typically a challenge during coding **Provides a Narrative** Accommodates for gender-specific need to contextualize or provide narrative for information. Discussing data organization in context of the human genome project or human traits provides this narrative to make concepts fall more logically **Be explicit** Students are asked explicitly for multi-level stratification of data and organization into known data structures which are given assessment criteria

Key Learning Event: Using a Decision Tree to Organize Your Data

Context: The current session serves to introduce students to the importance of data organization into discrete structures. Many students will have experience entering and organizing data in excel. However, few may have thought explicitly about why the data was organized in this manner.

Instructional Goals Applicable: **CO.3.1, DR.1.2, DR.1.4, DR.4.1**

Specific Objectives: During activity, students will:

1. Organize a dataset with 2-4 characteristics

2. Identify common characteristics of their dataset
3. Use a decision tree to determine an appropriate data structure
4. Organize the data into the chosen structure
5. Present and justify their choice to the class

Steps of the Activity:

1. Students will be asked to form pairs to work on the activity. Volunteers will encourage new pairs of students to form.
2. Students will be given a set of colored pairs of letters representing DNA bases. They will identify the characteristics common between different data points
3. Students will use these characteristics and the decision tree framework discussed in class to identify an appropriate data structure for their data
4. Students will organize the data into the data structure
5. Volunteers will be asked to mediate student work which will occur in pairs.
6. Students will be asked to justify their choice to the class and evaluate their decision. Instructors will guide student thinking with questions:
    1. Why did you chose that particular data structure?
    2. Why did you chose A or B?
    3. What were the common characteristics of your data?

Justification: This specific instructional event is appropriate for both context and the course learning goals for the following reasons:

- **Enable students to gain practice organizing data into common datasets**: Organizing data into basic structures is a core cognitive skill defined in the course learning goals
- **Allows students to explicitly practice**: The decision tree framework allows students to practice the computational thought process required to organize their data.
- **Encourages creativity:** Organizing a new set of data provides students with the opportunity to problem solve where they might not have all prerequisite information needed. This encourages students to think creatively about how they might analyze and organize their data.

Session 5: Data Structures 2

(designed by Rebecca Alford)

Lesson Overview

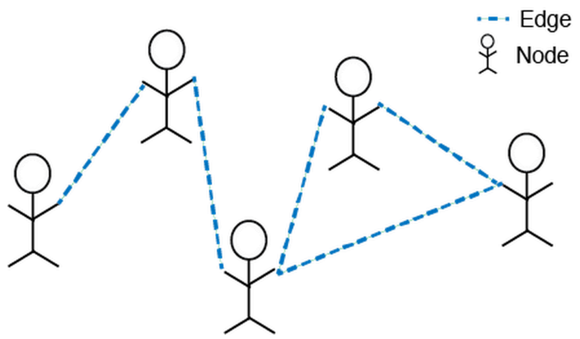
This session encourages students to ask deeper questions about the organization of their data. Students will explore the idea of data structure/data fit and gain experience working with connected a graph, a more complex, yet fundamental data structure in computer science. Learners will also explore the importance of graphs and representing networks in social media platforms, a common and relevant application of computer science.

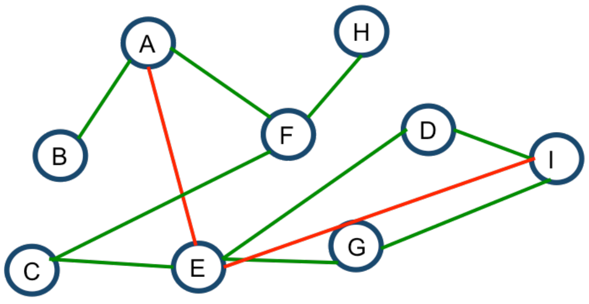
Summary of Assessments

- Introductory activity: as a group discuss examples of data that might not fit into the list/stack/queue/dictionary described in the previous session (**DR.1.6, DR.1.7, DR.2.1, DR.2.2, CO.3.1**)
- Draw a graph: draw a simple connected non-cyclic graph (**DR.1.8, DR.1.9, DR.2.5, DR.2.6, DR.2.7, DR.2.8, CO.7.2**)
- Construct a human social network graph: use a list of randomnly generated connections between students (instructor/leader mediated) to form a human social network graph and verify it's correctness (**DR.1.10**, **DR.2.5, DR.2.6, DR.2.7, DR.2.8, CO.7.2**)
- Post-discussion: discuss why alternate data formats are important and suggest new ways to organize data for non-traditional problems (**DR.5.1, CU.6.1, CO.3.1)**

Key Assessments

Activity 1: Construct A Human Social Network Graph

Broad Description	During instruction, students will learn that social networking sites organize networks of people and manage relationships using a more complex data structure called a graph. Graphs increase level of complexity because data points can have *n* relationships with other data points. During instruction, students will learn the definition of nodes and edges and have practice creating graph data structures from simple sets. Students will be provided several balls of yarn, scissors, and a list of randomly generated pairs of students. Students will be prompted to: *Generate a graph representing the class using the connections provided. Use the string to represent edges and yourselves to represent the nodes.* Guidance: Volunteers will not provide the final answer, but guide students to correct incorrect versions of the graph. Volunteers and instructors will also verify the final correctness of the graph. An example of students connected in a graph is shown here: 

Cognition	Students will demonstrate their understanding of graph terminology, components, application, and usefulness in real-world data scenarios. The assessment is also designed to engage students in a non-coding environment and This assessment aligns with the following course goals: **DR.1.10, DR.2.5, DR.2.6, DR.2.7, DR.2.8, CO.6.2.**
Observation	Students will need to identify the <u>nodes</u> and <u>edges</u> in the data structure and assemble themselves into the appropriate graph described by the list. Volunteers will ask the following guiding questions to guide the student activity: • Is your current edge reflected in the original list? • Are you connected to too many nodes? • Is the edge between person A and B correct? • Does the current class graph reflect the list of edges? Using these detailed corrections, volunteers can help students use the logic to self score the organization of the complex data structure. Volunteers will be observing for both formation of correct edges during the activity, application of feedback (do students correct bad edges when given hints during the activity) and score the final product for correctness.
Interpretation	Volunteers will gauge student understanding by scoring correctness of edges in the graph in real time. Students will be continually asked to verify whether edges match those in the list. *How to Score:* The list on the right describes pairs of nodes (circles) in the graph, which represent individual students. Students will use string to assemble themselves as edges. Green lines represent correct edges in the list, whereas red edges represent incorrect edges.  **Edge List** A-F B-A F-H C-F E-D G-I E-G D-I C-E Students will demonstrate sufficient understanding of the graph data structure by building a graph with all green edges.

	Based on the speed and level of understanding, students might be asked to build several graphs for *repeated practice.* After the activity, students will also be asked about features of the class networking graph. Answers with sufficient and insufficient responses are shown below: • Are cycles present? o Yes – Correct if cycles are present, Incorrect if not o No – Correct if cycles not present, Incorrect if yes • How connected is the graph? o Sufficient: Because the graph has lots of edges, it is very connected or dense. If the graph does not have lots of edges, it is 'sparse' – meaning it does not have lots of connections o Insufficient: Cannot identify the meaning of multiple or lesser edges present in a graph
Design Standards	**Validity:** The assessment is valid because it directly measures students understanding of the graph data structure: terminology, assembly and characteristic features. The assessment also requires students to think creatively about how to organize members of the class to most efficiently organize the data structure. Students can only perform well on this assessment if they understand the vocabulary and graph. Some students might follow the lead of their peers, however would not perform well if an instructor determined they still did not understand the appropriate terminology/be able to answer guiding questions. **Reliability**: Because the graph is either correct or incorrect based on the provided edge list, scoring is stable across raters. Specific vocabulary and criteria are also provided to volunteers for guiding questions. Students would perform equally upon retest with no instruction because insufficient understandings would result in incorrect final graph conformations. **Objective:** The assessment is objective because students will try multiple graph topologies (different edge lists) to make sure students occupy very connected and less connected positions and are not biased based on difficulty.
Justification	**Vary context for assessment** Interactive activity allows students to demonstrate understanding of data structures outside of a written or in-code context. Accommodates for learners who might perform better on interactive activities than coding assignments **Opportunity for Feedback** Volunteers provide real-time feedback by encouraging students to self-correct when creating complex graphs and practicing with various combinations of edges **Focus on Goals** This activity aligns well with goals of being able to construct and identify features of a graph

Instruction

Learning Events:

- Group Discussion: Instructors, volunteers and students will discuss the consequence of working with more complex datasets. What if there are multiple characteristics of the data? How do you evaluate data structure/data fit?
- Presentation: Instructors will present and discuss how to handle more complex data, specifically describing the parameters and properties of the graph data structure.
- Demonstration: Instructors and volunteers will demonstrate how to construct a graph from a simple set of data (<5 nodes, <10 connections)
- Student Practice: Students will be asked to draw a graph individually using markers and paper from a set of data with similar size but differing characteristics
- Student Activity: Students will be asked to assemble a human social network graph using themselves as nodes and balls of yarn for connections.
- Presentation: A volunteer or expert will present on the applicability of graphs to representing large social networks in social media platforms
- Group Discussion and Reflection: As a group, evaluate the challenges of working with more complex datasets and data structures.

Key Learning Event: Creating a Graph from a 'Simple' Dataset

Context: At this point in learning, students will have been exposed to the motivation for organizing complex data and key parameters for the graph data structure.

Instructional Goals Applicable: **DR.1.8, DR.1.9, DR.2.5, DR.2.6, DR.2.7, DR.2.8, CO.7.2**

Specific Objectives: During activity, students will:

1. Practice identifying key features of the graph data structure: nodes, edges, connectedness, density
2. Practice creating graphs from simple data sets with guidance from volunteers

Steps of the Activity:

1. Students will be provided a simple data set consisting of names of people and relationships between them (i.e. - Dan is friends with Sally, Sally is friends with Rachel)
2. Students will be instructed to identify the nodes in the data set
3. Students will identify the connections between nodes
4. Students will draw a graph representing the dataset
5. Volunteers will provide guidance and feedback for the correctness of student drawings.

Justification: This specific instructional event is appropriate for both context and the course learning goals for the following reasons:

- **Aligned with key instructional goals**: A key cognitive skill goal of the course is for students to be able to assemble data into a graph data structures and assess its components/characteristics

- **Builds from prior knowledge and motivations:** At this point in the instruction, students will already understand the importance/motivation of data organization and have a framework for organizing data based on its characteristics
- **Promotes individual confidence and thinking**: Individual work enables students to gain independence and confidence in learning new confidence

Session 6: Algorithms 1

(designed by Tom Huynh)

Lesson Overview

Learn what is an algorithm and the need of efficient algorithm. Student will have a few opportunities to identify algorithm to solve problems and then "count" the number of steps taken to achieve each solution. Through practice, students would acquire the understanding of what is an algorithm. Through application, students will generate meaning of why algorithm are used in real life context.

Summary of Assessments

1. Introductory activity: What is an algorithm? General question for class discussion to gather preconceptions of this topic to see if there are any misconception to be addressed **(AT.1.1, AT.2.1, CO.3.3, CU.2.1)**
2. Understanding algorithm by recognizing its value. Student understanding is being accessed with the Interpretation facet from UbD.
    a. Write down 2 or 3 activities in students life that has a set of steps **(AT.1.1, AT.2.1, G.4.1, CU.2.1)**
    b. Count the number of steps that is involved in each of the real life activity **(AT.1.2)**
3. Application of understanding. Accessing understanding using Interpretation and Application facets (fully specified):
    a. Given a list of story problems (e.g, given 10 foreign words, can Jenny use a dictionary to translate them in 5 minutes?), students are asked to identify which one is possible and which one is not possible. **(AT.1.3)**
    b. Tell a classmate, step by step, how to translate the list of words from think-pair-share. **(AT.1.4, AT.2.2, AT.5.1)**

Key Assessments

Algorithm 1 - Identify the need for efficient algorithm

Set-up/ General Explanation	After receiving instruction and assessment to understand what is an algorithm, this assessment is used to further develop students' ability to apply the definition they had learned in context of real life problem. Students will apply the definition of algorithm efficiency as they try to evaluate whether the problem can be solved within a given timeframe.

	Through practice, students will recognize some problems might not be solvable in a given timeframe and therefore stimulated to find better (quicker) ways to solve the problems. Through think-pair-share, students will attempt to solve this by themselves first, then pair up to explain their thinking. Finally, the class will all share their answer and verified understanding as a whole.
Cognition	Goal Specification: **AT.1.3, CU.3.1, CO.6.2**
Observation	In this activity, you are asked to think whether a fellow student from another class can do the following task within the given timeframe. Please indicate (Yes/No) 1. Given 10 Spanish words, can Jenny use a Spanish to English dictionary to translate them all in: a. 10 minutes? (Yes/No) b. 5 minutes? (Yes/No) c. 3 minutes? (Yes/No) 2. Given a Tower-of-Hanoi game with 5 dishes, can Josh complete it in: a. 10 minutes? (Yes/No) b. 5 minutes? (Yes/No) c. 3 minutes? (Yes/No)
Interpretation	During, the think-pair-share process, volunteer will rate student progress of understanding following the following scoring rubric: 1. Does student recognize there are steps involved in each task? (Yes/No) 2. Can the student identify the steps involved in each task? (Yes/No) 3. Did the student try to reduce the number of steps as the timeframe lessened? (Yes/No) In the dictionary game, one efficient student may discover is to make sure the words are alphabetized (by the 1st letters - nth letters give the degree to efficiency) In the Tower-of-Hanoi game, student might form/notice a recursive pattern in their action Each question is designed to validate a specific goal specification as this task involved a vertical transfer between goals. If all three questions can be answered yes, it is reasoned that student have understand efficiency as fewer steps performed to solve a problem. However, this only weakly imply that students understand the need for an efficient algorithm as to solve the problem in less time.

	If any of the questions were answered no, students may be missing the knowledge component (KC) specified in the goal. Instructor may use this as an indicator for more practice for the missing KC.
Validity	Since this is a formative assessment, it can only offer a glimpse of what the students' understand. Failure on this assessment would mean the instruction has not succeed while success in this assessment would mean student might have understood the instructional content. In order to have a valid indication of student's understanding, there needs to be more observation (more than 2 problems). These observation could be reliable given that the tasks have 1) steps on solving the question and 2) optimizable.

Instruction

Algorithm 1 - Identify the need for efficient algorithm

Context: This learning event is sequential to another learning event. Student should have just learned the definition of algorithm and finished the assessment for it. The classroom should be similar to a computer lab (access to computer but need to be arranged to promote collaboration). Students would be sitting in group of 3 or 4. Each group would have a volunteer to help monitor student activity. The classroom should be equipped with a enough dictionaries and tower-of-hanoi game so at least one group has the game if not per student. Though, any game with a fixed procedural pattern may be substituted in order to accommodate different interest and neurodevelopmental profile.

Instructional Goals Applicable: **AT.1.3, AT.2.1, CU.3.1, CO.6.2**

Steps of the Activity:

1. Instructor introduce the game and dictionary for the task
2. Students and volunteer as a group explore the game and dictionary
3. Instructor demonstrate one way to solve the problem (an inefficient way)
4. Student are asked, on paper, to answer the questionnaire by themselve to gather preliminary understanding - this would allow time for student to play with the game further to build up their curiosity of what is possible and what not
5. Student are asked to pair up with a neighbor and explain their solution
6. Volunteer are asked to evaluate student based on the scoring rubric.
7. Conclude learning activity and learning lesson by the opening question what is an algorithm? and why do we strive for efficiency?

Justification:

- **Use multimedia to explains new concept**
  This activity strive to give student a hands-on task of playing with two unique game (dictionary look-up + tower-of-hanoi) while making *meaning* out of the word efficiency within the context of algorithm.

- **Define learning goals with not just knowing *what*, but also knowing *when* and *where*.**
  By working in context of a game, student may be able to understand the motivation and value behind algorithm. Though this is not connected to the computer, the games should serve as a bridge between concrete and abstract concepts.
- **Allow room for change in instruction to accommodate learner's difference and variability**
  By allow room for substitution game, the instruction design leave space for teacher who may have more knowledge of the students to find suitable activity once the course actually take place.

Session 7: Algorithms 2

(designed by Tom Huynh)

Lesson *(sorry, I'm only doing a skeleton outline here)*

Learn about developed algorithm and its usage.

Assessment Overview

- Introductory activity: What is a computer scientist's toolbox? Ask students to give example of what they know and what they think is a tool. Is an algorithm a tool? Guiding student thinking through the abstract concept of "tool" as works of other computer scientists have created, and that it may be reused. **(CU.2.1)**
- Understanding the available tools by guided discovery. Given a list of algorithm: binary search, bubble sort, selection sort. Find answers to these question through browsing the web (might need a more structured educational text like a book/website) **(AT.1.6, AT.1.8, AT.2.5, AT.4.1, AT.5.1, CU.2.2)**
  - What is the algorithm do? Its purpose/
  - Find its implementation in Python through searching with the term "implementation in Python" (
- Applying skill with a given a data to manipulate. For example (one data set) **(AT.2.4, AT.2.5)**
  - Everyone in class and their height and given in the table below. You are asked to line up by height a unique number call *Identification Number* is given to each. Can you use the an implementation of the bubblesort algorithm to order your classmates and yourself by height?

Session 8: Debugging and Troubleshooting

(designed by Tom Huynh)

Lesson Overview *(sorry, I'm only doing a skeleton outline here)*

Learn what is a "bug" and how to ensure the code is doing what it is specified to do

Assessment Overview

- Introductory activity: What is Debugging? Probe student for their understanding of this word then define the term “bug” with its historic naming and its modern usage. **(DT.1.1, DT.1.2**
- Understanding debugging:
    - Find a bug from students’ algorithm from one of the previous session **(DT.2.1, G.3.3)**
- Application of debugging: test code to ensure it perform to specification **(DT.1.2, DT.2.2, G.3.4)**
    - After locating a bug, work in pair to find possible solutions **(DT.2.3)**
    - Try the solution **(G.3.4)**

Session 9 and 10: Final Projects

(co-designed by all team members)

Overview

Context: At this point in the course, students will have explored the main transfer goals of the course: general computer science understandings, abstraction, algorithmic thinking, debugging and troubleshooting, developing confidence and inspiring curiosity. The final project provides students to assemble their new knowledge and understandings into a summative project.

In the first part of the 9th lesson, students will be introduced to the final project assignment. They will be asked to split into groups of 3-4 and work with a volunteer. In the following session, parents will be invited to listen to student project presentations. This presentation provides students with an opportunity to develop confidence and receive positive feedback from their accomplishments and findings throughout the course.

Final product:

Students will give a five minute presentation to introduce their topic of study and describe its relevance to the course learning goals (see above). The presentation will be informal, however students will be encouraged to use visual such as a poster. An example of the final project outline would be:

Topic: Computer Animation
- *Connection To Algorithms: Motion Tracking, Ray Tracing, Hair animation techniques*
- *Connections To Abstractions: Characters and the background could be animated separately and by different people, putting together the animation does not require knowledge of how every aspect was animated.*
- *Connection to Data Representation: Characters, pixels and images all require different data structures (students could present specific examples).*

Resources: All the students will be given access to a computer, as well as supplies for making posters. Each student group will have a volunteer to guide them through the process.

Instruction

Students will receive a set of instructions describing the parameters for the final project. These include:

- Select a topic in an area related to computer science
- Connect your area to the technical transfer goals of the course (**AT, AB, DR, DT, G**)
- Share your findings with the class

The project can be informally split up into the following phases:

1. Brainstorming ideas: The groups of 3-4 will work together to agree on a topic, they all find interesting.

   Guidance: The volunteer will avoid intervening and suggesting topics they find personally interesting. They may, however, help students come up with vocabulary for topics (since the students may not know the keywords). If the students are struggling to come up with a project, the volunteer may ask the students to brainstorm "non-cs" topics and suggest to the students how they may be connected. (I.e all the students are interested in medicine, suggest they research medical robots like the Da Vinci).

2. Researching a topic: Once the students agree on a topic, they will use their laptop's to find out more about the topic. At first, students will be encouraged to find interesting information in general, before focusing on the connections. The students may do the research individually or altogether, but they will be encouraged to communicate throughout. The group will be asked to come up with a list of 6-10 ideas that are interesting about the topic, to help the group find connections.

   Guidance: The volunteers may be more involved in this phase. They may help the students come up with key words and explain particularly difficult topics. However, before giving an explanations, volunteers should try to guide students to come up with one on their own.

3. Finding connections: Students will be asked to use their list of interesting ideas to find connections to Abstraction, Algorithms and Data Representation. They will be asked to pick one example of each to include in the presentation. It will be emphasized that the connection does not need to be unique to the application, because the goal is just to show the students that the basic concepts they learned are applicable everywhere.

   Guidance: Volunteers should encourage students to find connections on their own. Students may have an easier time finding connections to the Algorithms, then to Abstraction or Data Representation. If students are too focused on finding unique connections, volunteers should encourage them to think of more basic elements. For example, if students are struggling to come up with how to connect Data Representation to the Da Vinci robot, volunteers may guide students to think of what kind of data the robot needs and how it may be stored.

4. Preparing the presentation: Students will create a poster describing what they researched and how it connects to the key concepts.

   Guidance: The volunteers can help students put together the presentation, but should remain an extra resource, not the guiding force. The volunteers will encourage students to use their creativity in coming up with the presentation.

5. Presentation: Students will present the material to the class and the parents.

Note on timing: Because this project will be split between two sessions, volunteers should make sure students begin step 4 during the first session. If the students start creating the content, they will be more likely to remember it for the next session. Based on timing constraints, the visual aspect of the project may be removed. Volunteers should record the students ideas, to help them finish up their projects during the second session.

Justification: The course project connects to multiple big ideas and connects all the areas of the course

- Encourages Creativity and Choice: The student groups get to research a topic of interest, so they can pick a focus they find personally interesting.
- Connects concepts and applications: This project forces students to tie together all the important aspects of the course with a new application. Students will see how everything they have learned works together in real-world computer science.
- Explicit connection to goals: The project encourages the students to find algorithms, abstractions and data representations in new context, thus explicitly targeting the goals of identifying those concepts.
- Promotes Dispositions and Personal Development: The project lets students individual interests shine and encourages students to be both creative and confident. The work will show the students how much they learned in just a few short weeks.

Assessment

Final Project and presentation (Details of project are in the instructions section)

Note – this assessment is not formatted as a table because Word will not allow us to add tables inside of the larger table while still being readable.

1. Cognition: This project is summative and will incorporate a broad range of goals from the course. In particular: **G.1.3, G.1.4, AT.4.1, AB.2.2, AB.2.3, DR.1.4, CO.3.3, CU.2.1**
2. Observation: The students will work in groups of 3 to create 5 minute presentations to be shared with the class and the parents. This presentation will cover a new topic in computer science and connect it to Abstraction, Algorithmic Thinking and Data Representation. Although the assessment is summative, it is naturally built in to the structure of the course.
3. Interpretation: The interpretation will be based on two sources of observation: the group work and the presentation. Because of the general difficulties of public speaking, most of the weight will be given to the group work.

1. Group work: Volunteers take informal notes based on the following metrics. Since the volunteers should focus on helping students, they may use a grid with student names and metrics and record checkmarks and notes.
    1. Contributing ideas: Does the student contribute ideas to the group or at least attempt to come up with the solutions? Because we want students to be confident about contributing ideas about unfamiliar topics, participation in a small group is an appropriate way to measure this.
    2. Ability to identify the key concepts: Given material on a topic, can a student find connections to Abstraction, Algorithmic Thinking and Data Representation? Because sometimes the connections may be identified based on names of concepts, the following clarifying questions may be asked:
        1. Abstraction: What is abstracted? Why is abstraction important in this content?
        2. Algorithmic Thinking: What does this algorithm do? Have we seen similar algorithms before? Could you use this algorithm somewhere else?
        3. Data Representation: What is the data and what kind of structures are being used for it? Can you think of other representations?
    3. Potential Grid:

Name	Contributes Ideas	AB Concepts	AT Concepts	DR Concepts
Ex. Sally	++++ Found a cool way to combine two ideas!	++	+	+ Identified unexpected connection to trees struct.

2. Presentation: Volunteers will take notes on the following metrics:
    1. Comfort presenting: Do the individual students feel comfortable presenting? Because public speaking is difficult and the students will not have much time to prepare, this criteria is judged leniently. The volunteers should judge this criteria in comparison to how the students felt at the beginning of the course
    2. Comfort with the material: Do the students explanations should very rehearsed and memorized or do they explain the material in a more conversational manner? If a student understands the content, they will be able to explain it more comfortably and not have a rigid definitions memorized. However, if the presentation does appear memorized, the group work will be consulted, because students who understand the material may feel nervous about public speaking.

3. Potential Grid

Name	Presentation	Material
Ex. Sally	• Spoke slowly and clearly, appeared a lot more confident than at the start of the course.	• Confused definitions covered in course

4. Design Standards
    1. Valid: This assessment is valid because it requires students to understand the three main concepts. While, they may find an “algorithm” based on the name, if they have to explain what it is, they would need to understand it. Identifying data structures and abstractions would also require understanding of the content. This assessment forces students to look at known concepts in a new context which is more likely to promote transfer.
    2. Equitable: Because presentations are informal, students should hopefully feel more relaxed about giving them. For more shy students, volunteers can reflect on how students contributed to the preparation of the presentation. While confidence in presenting is one of our goals, we want to make sure students met the other goals as well. To ensure this the interpretation is based on both the presentation and group work.
    3. Objective: The volunteers will be given metrics to grade on. Furthermore, emphasis will be given to individual growth. For presentations, students will be graded based on how much they improved, and to be fair, volunteers will reference their notes from the beginning of the course instead of relying on memory.

Research Designs

Investigating the utility of block-style vs. syntax-dependent coding in improving procedural thinking skills

(designed by Rebecca Alford)

Background

Procedural (algorithmic) thinking is the ability to communicate a task in a series of detailed steps. This skill is the basis for communicating instructions to a computer and is a key transfer goal for the ThinkTech outreach program. The ultimate transfer goal is that students can design an algorithm, no matter what coding environment they are in. In fact, in many interviews for STEM internships and problem solving sessions, students are asked to design algorithms on a whiteboard (context-independent).

In recent years, several educational tools for teaching computer science to young students have been developed. All of these tools include a graphical user interface (GUI) where students move blocks or objects together to create running programs. These include Alice, MIT App Inventor, Stencyl, and DesignBlocks. In this design, I focus on Scratch - a block-based programming language for graphical programming.

These tools are more welcoming to students by building from prior knowledge of computer game-like environments. They also eliminate distracting variables such as resolving syntax-errors in code, unrelated to the actual algorithm step. However, it's possible that the game interface prohibits students from strengthening their algorithmic thinking skills. Blocks can fit together and execute a reasonable program without students understanding why elements fit together.

It can be postulated that learning to write the code is more valuable for developing transferable algorithmic thinking skills. As computer scientists, students will need to understand how to implement algorithms. In addition, code might force students to focus on the actual step and why it is reasonable instead of the shape of the block.

Research Question

Given two modes for practicing algorithmic thinking skills, (1) the Scratch block-style programing language and (2) Python code + tkInter, which mode will lead to more transferable algorithmic thinking skills?

Hypotheses & Related Predictions

While the block-based programming interfaces are better for general exposure, they are not reliable forms of instruction. The geometry of the blocks interferes with students' ability to focus on the actual algorithmic step. Despite the challenge of resolving syntax errors, Python is a more ecologically valid mode of instruction. For this reason, one might hypothesize instruction in Python + tKInter will result in a more transferrable understanding of algorithms.

Experimental Design

Note, this experiment would serve as an alternate lesson plan for ThinkTech session 1. The assessment is changed from a presentation to a whiteboard activity to make the assessment more quantitative and practical for research with a larger group of students if available. However, both assessments test algorithmic thinking skills.

In this experiment, students will receive introductory level instruction on how to design an algorithm. They will receive an opportunity to practice their knowledge using either the Scratch platform or the Python platform. Students will then be asked to design an algorithm on a whiteboard (platform independent) as a test of their procedural thinking skills.

Control Group - Students using Python + tkInter as a practice environment
Experimental Group - Students using Scratch as a practice environment

Independent Variable - Platform used for practice (Scratch vs. Python)
Dependent Variable – Ability to design algorithms out of context (written pseudo code on a whiteboard), demonstrating strong algorithmic thinking skills

Methods:

Subjects

If the course is in a single section, subjects will be randomly split into the experimental or control group by seating. If there are too many students and there are two sections, experimental and control groups will be split by section. Regardless, all subjects in this study will be female, ages 11-14 and enrolled in ThinkTech.

Materials

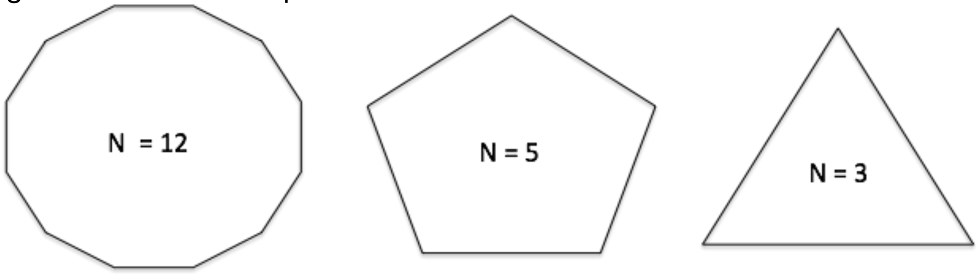
<u>Hardware</u>: 1 laptop per student - LINUX operating system
<u>Software</u>: Scratch Interactive coding platform, Python version 2.7, iPython interactive shell, Sublime Text editor (or other consistent text editor)

Procedures

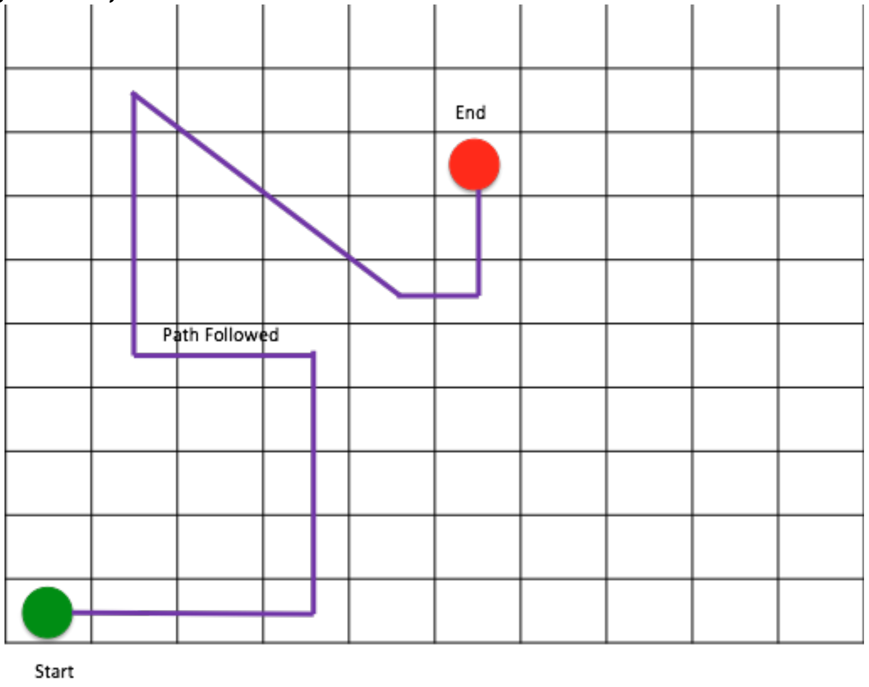
The study will be conducted during the introductory ThinkTech session, which is traditionally 2 hours in length. In each section (or each group), a different language (Scratch or Python) will be used to teach the introductory lesson on algorithms.

In both studies, the following procedure will be followed:

1. **Instruction**: Students will listen to a 45-minute lecture by a trained instructor about how to design simple algorithms. Topics of this lecture will include
    1. Define an algorithm as a set of specific instructions needed to accomplish a task
    2. The importance of specificity in step design
    3. Types of steps: sequential, parallel, iterative
    4. An example of an algorithm containing 3-5 steps for each type of step (in c)
    5. A simple example of how to iterate when encountering a problem
2. **Environment-Dependent Practice**: Students will be asked to design an algorithm in the Scratch or Python platform to draw a shape with n sides. Students will be shown example output shapes with different values of n (see below). Students will be given 30 minutes to practice the task.



3. **Transfer Test**: Students will be asked to write an algorithm that helps a game piece move on a 10 x 10 game board. The algorithm should take a series of positions and move the game piece to those positions consecutively. Aligned with **AT.2.8, AT.3.1, AB.2.1, G.1.3**



Data Collection & Scoring:

There are several elements of data collected from this experiment:

1. **Correctness**: Did the students algorithm lead to a correct result
2. **Trials**: Did students try reasonable paths or solutions even though they did not achieve the correct end result
3. **Pseudocode:** Did students' pseudo code resemble a specific algorithm? or Vague English? Scorers would look for the following specific features:
    1. Use of English words vs. use of python keywords
    2. Appropriate placement of keywords and data
    3. Presence of loops where appropriate
    4. Scorers will not score for specific syntax, but for general algorithmic structure.

Assessment of Design Quality:

Confounding Variables

There are several confounding variables that might interfere with validity:

- Individual differences between learners:
    - **Preference for visual learning**: Some students might prefer the graphical nature of the block style code, and therefore gravitate toward this method of practice
    - **Preference for syntax-based learning**: Other students might prefer the more structured python environment and syntax based feedback.
- Differences between classroom and laboratory:
    - Students cannot receive as much clarification and feedback on practice activities in a classroom due to limited resources
- Biased Prior Experience
    - Some students might enter with prior experience using Scratch from external outreach programs
- Disposition:
    - Writing on a whiteboard might make students nervous about performing the task

Triangulation

Because the proposed research uses several methods of evaluation - one point in time, and a second observation during the task, there is greater reliability in the results.

Sampling:

This research is designed in context of the ThinkTech program. Larger populations of students would be needed to extend the results of this study beyond the outreach program.

Resources

- Kafai, Yasmin B., Deborah A Fields, and William Q. Burke. “Entering the Clubhouse: Case Studies of Young Programmers Joining the Online Scratch Communities.” *Journal of Organizational and End User Computing* 22, no. 2 (32 2010): 21–35. doi:10.4018/joeuc.2010101906.
- Resnick, Mitchel, Brian Silverman, Yasmin Kafai, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, et al. “Scratch: Programming for All.” *Communications of the ACM* 52, no. 11 (November 1, 2009): 60. doi:10.1145/1592761.1592779.
- Sykes, Edward. “Determining the Effectiveness of the 3D Alice Programming Environment at the Computer Science I Level.” *Journal of Educational Computing Research* 36, no. 2 (May 1, 2007): 223–44. doi:10.2190/J175-Q735-1345-270M.

Do skills learned in interactive tasks transfer to coding tasks for novice programmers?

(designed by Anastassia Kornilova)

Research Question

Problem-solving skills can be taught in many interactive contexts that are not directly related to programming. If these skills are taught well, they should transfer to other domains. However, for novice programmers, the different nature of programming domain may hinder transfer, because the new nature of the programming assignment may be overwhelming. This study looks at that transfer, by testing if an interactive problem solving tasks can help students complete a programming task with more ease.

Experimental Design

The study will be based on activities from lesson 2 in the curriculum. The interactive task considered will be the *Activity 3: Procedural Flower Drawing* and the coding activity will be *Activity 4: Generative Art Coding*. The experiment will be organized as follows: both groups will be given an introduction (*Activity 1 and 2)*. Then, the groups will be split up: one group will be given *Activity 3* first, and *Activity 4* second. The other group will be immediately given *Activity 4.*

Experimental Group: Do *Activity 3* before *Activity 4*
Control Group: Do *Activity 4* right away.
Independent Variable: Order of activities/presence of pre-coding activity
Dependent Variable: Performance on assessment for *Activity 4* (as specified in the Lesson 2 assessments section).

Method

The subjects will be typical students in the ThinkTech program, they will be split up for the part of the lesson involving the study. All the materials and procedures will be used as described in the lesson plans. Note that students in the control group will do *Activity 3* after *Activity 4* to get a fair experience, not as part of the study.

Data Collection & Scoring

The data collected will be the results of the assessment as described in the lesson. The distribution of the *Insufficient/Sufficient/Exceeding* will be compared between the two groups. Evaluation of these results will primarily be based on comparing the ratios of the *Insufficient* and *Sufficient*, because students who got *Exceeding* may have had more prior programming or problem solving experience.

Hypotheses and Related Predictions

While programming may be a challenging and confusing paradigm for new students, practice in problem-solving contexts may still be beneficial. Even though writing code may seem different from writing instructions for flowers, it will set students up to think about procedures. My hypothesis is that students that did *Activity 3* first will perform better on *Activity 4.*

Assessment of Design Quality

Sampling: This study is conducted in the context of the ThinkTech program which may initially be a small sample. The initial results may be invalid, unless the program is expanded to a larger population.

Validity: We are testing if out-of-domain problem solving practice can transfer to a new and challenging domain. *Activity 3* represents an out of domain task which is related to the problem solving concepts; *Activity 4* uses programming which should be a fairly new domain for all participants. If the experimental group performs better, than the change should have come from the previous activity that taught the same problem-solving skills.

Confounding Factors
While some of these factors can be introduced as stratifying variables, this may require a larger sample size than available:

- Prior programming experience: it is possible that students in one group have more prior programming experience, which would make the programming domain less challenging for them
- Confidence: one of the groups may have generally more confident students; they may be willing to take on new domains more readily, regardless of prior experience.
- Comfort with environment: The experimental group may feel more comfortable with the environment of the program by the time they get to the programming task. If they feel more comfortable and confident, then the coding task would be less intimidating for them to approach.

Would task related (feedback focusing on the task) feedback be more effective in promoting student's curiosity than personal feedback (feedback focusing on student)?

(designed by Tom Huynh)

Research Question

Some research from Cognitive Psychology shown task related feedback can help students achieve higher score on post test when the goal is to do better on a task. If the goal performance task is more dispositional than skill related (e.g, a 'feeling' of wanting to know more versus complete writing a program), however, it would be more reasonable that the personal feedback be use in learning lesson that promote curiosity.

Experimental Design

Measuring the effect of feedback could be quite challenging. For one, how to really measure the 'effective' of feedback of two different instruction strategy while keeping all the classroom context? Student's curiosity may be measured with the amount of quests and questions they are willing to ask. During class, instructional feedback will be varied between task related (e.g, the quality of the code they write) and personal feedback (e.g, complement student's ability to program).

It would be the easiest randomly split the classroom into halves, one control and one experimental. This will have a fairness issue that could be resolve with the by varied the time of treatment using the following treatment plan:

Methods

Procedure:

The classroom's subject selection will be arbitrarily split by seating location. The experiment will last for two lessons where the lesson may be similar enough to keep the context the same. The questionnaire to measure students curiosity will contain concepts that students may not have seen before. It is reason that if the student's curiosity is high, they would answer as many questions as possible.

In Algorithm 1 class, the introduction of the lesson where students are probed of what they know about the subject could be used as a pretest. Student would be given the questionnaire when they enter the class. Since the program can't do formal assessment, the questionnaire will be given as extra topic guiding the student. Completion is up to the student's own will. Throughout class doing Algorithm 1, the group A will receive comments about how their ability can be good in this field whereas group B will receive comments about how their performance in the class activity. When they leave, students are encourage to do as many questions as they can when the get home.

In Algorithm 2 class, the introduction of the lesson serves as the midtest. The number of questions answered will be counted. Now group A will receive the same comment as previous class, whereas group B will start receiving personal feedback. The questionnaire will be collected at the beginning of the 3rd class.

Questionnaire:

1. What is computer algorithm?
2. When was it first come about?
3. What is computational processing limitation?
4. How “large” can data get?
5. What is the Big O notation?
6. etc.

Data Collection & Scoring

The data collection process will have to depend on the volunteer of the program. During the beginning of the class, volunteer are asked to work with group of students about the questionnaire. Then, they will be asked to temporarily turn in their questionnaire so we can scan it for documentation and see where their interest may be (dual purpose).

There would be more questions in the complete questionnaire, perhaps around 50 questions. The number of questions answered would be used to determine level of curiosity.

Hypotheses and Related Predictions

I am hypothesizing that both personal feedback and task related feedback has its purpose. They can both be beneficial to students given the different mindset it provoke in students. So when looking at the guideline for feedback, general statement like “focus on the task - not the learner” should be taken with context.

Assessment of Design Quality

Sampling:

The sample of the students population in this program is not quite generalizable to the students population of the whole, as they are mostly girl. However, this experiment can show how the design could work and be applied in a more representative sample. Also, the amount of students in the program is quite small, so there needs to be a few trials before any specific conclusion is reached in regarding the type of feedback that help girls learn.

Validity:

The questionnaire played a large role in this experiment. Its validity is based on the relevance of the questions and how it promotes students to find out more regarding the knowledge goals. The number of questions answer should be a valid indicator of curiosity as curious people want answers.

Reliability:

Its difficult to maintain the same interactions between students and teacher/volunteer across lessons/experiments. The data collected based heavily on the interaction of the teacher and the volunteer and the subject they teach. More detailed guideline of key interaction need to be establish to ensure the result is reproducible.

Triangulation:

The timing of the treatment may need more than one implementation to provide accurate measurement. The current methodology may not be the only way but it is one possible configuration with the given class outline.

Possible Confounds:

Beside the feedback, there may be other reasons (e.g, external rewards like finding out what's going on for next class) that motivate students to provide answers. Students who come to class may also be interested in the topic by themselves and want to learn strongly, therefore not affected by the treatment.

Resources

How much experience does our team have in the project domain? As students? As teachers or tutors? How much reading have we done about education in this area?
Both members of our team formally trained computer scientists and scientists and have extensive experience working in outreach environments. The background of each team member is described below:

Anastassia Kornilova is a third year undergraduate majoring in computer science, with a minor in language technologies, and has been coding since freshmen year of high school. She is interested in educational technology and is currently involved with starting an EduTech club on campus. She does research in the Language Technologies Department of the School of Computer Science, and has had internships with Square and Pinterest. Anastassia consistently engages in STEM outreach programs and is the current undergraduate coordinator for TechNights, a weekly outreach program for Middle School girls ages 5-8 interested in computer science. Anastassia has also been involved in Roadshows, programs that reach out to K-12 schools in the greater Pittsburgh area and abroad promoting computer science involvement.

Rebecca Alford is a third year undergraduate majoring in computational chemistry with a minor in software engineering. She is an active developer for Rosetta, a software suite for biomolecular structure prediction and design and currently the youngest member of the community. Rebecca is also an NIH Undergraduate Research Fellow at the Gray Lab at Johns Hopkins and works merge biophysics with computer algorithms for prediction of membrane protein structure. Rebecca has engaged in community-wide efforts to improve diversity, specifically for females and individuals with disabilities. She leads several initiatives for women in computing in the RosettaCommons, has mentored four high-school students in computational research, and has engaged in K-12 STEM outreach as both a volunteer and instructor.

Ngan Nguyen Huynh is a graduate student in the Master of Educational Technology and Applied Learning Science program at CMU. He did his undergraduate at Florida Gulf Coast University where he obtained a B.S in Computer Science with a psychology minor. His interest in education technology developed in his travel during his junior and senior year. At that same time he also worked as a research assistant developing mobile learning application at the FGCU Institute of Technology Innovation. Ngan has experience in educational challenges in a resource limiting environment. He volunteered as an English teacher in Vietnam for 3 months where he observed in vivo the needs for aligned educational materials, trained teachers, and valid accreditation.

Since beginning this project, we have read various papers on STEM outreach and curricula for computational project. We list our most relevant resources below:

- Ecklund, Elaine Howard, Sarah A. James, and Anne E. Lincoln. “How Academic Biologists and Physicists View Science Outreach.” *PLoS ONE* 7, no. 5 (May 9, 2012): e36240. doi:10.1371/journal.pone.0036240.
- “Encouraging Science Outreach.” *Nature Neuroscience* 12, no. 6 (June 2009): 665–665. doi:10.1038/nn0609-665.
- Hambrusch, Sussane, Cristoph Hoffman, John Korb, Mark Haugan, and Antony Hosking. “A Multidisciplinary Approach Towards Computational Thinking for Science Majors.” *Association for Computing Machinery (ACM)*, March 2009.
- “Report: Excerpts from Science and Engineering Indicators 2004 National Science Board.” *Science Communication* 26, no. 2 (December 1, 2004): 219–22. doi:10.1177/1075547004271518.
- “What Most Schools Don’t Teach.” *Code.org*. Accessed September 18, 2014. http://code.org/.
- Williams, Laurie A., and Robert R. Kessler. “All I Really Need to Know About Pair Programming I Learned in Kindergarten.” *Commun. ACM* 43, no. 5 (May 2000): 108–14. doi:10.1145/332833.332848.

Are there educators, designers, and/or researchers who can serve as consultants to help you identify your target goals and the learning challenges in this domain, as well as reviewing your project design as it progresses?

We have identified the following individuals who can assist us in development of our project:

- **Carol Frieze, PhD** – Dr. Frieze is the current director of Women@SCS, Carnegie Mellon’s undergraduate organization for women in computer science. She organizes and advises several programs for K-12 and undergraduate STEM outreach, including TechNights, OurCS, and Adventures in Computing. Her research involves study of computer science cultures and emerging diversity. The TechNights program has provided much of the foundations for our program.
- **Lisa-Abel Palmeri PhD** – Dr. Abel-Palmeri is the Director of Technology at the Ellis School in Pittsburgh. She developed design-thinking and “maker” education at the independent all girls school. She is focused on creating a narrative for students and encouraging active learning through projects. She has inspired many of the techniques use din our instruction philosophy.

What educational materials (instruction and/or assessment) have already been designed to teach this domain?

Two types of materials are available for this domain - proposed school curriculums and outreach programs. We review both categories below:

School Curriculums:

- CS Principles: *Computation in Action*: Set of 17 introductory lesson plans focusing on **creativity, abstraction, data, and algorithms.** High School Level (Grades 10-

12) - Suggested AP course. Link: *http://csta.acm.org/Curriculum/sub/CurrResources.html*

- *Computer Science Teachers Association (CSTA) Exploring Computer Science*: 36 week curriculum based on CSTA standards (see below) - focuses on **computational thinking** and **applications**. Age range K-12 - partnership with NSF. Link: http://www.exploringcs.org/
- Introduction to Creative Computing with Scratch: 20 introductory lessons to teach Scratch, a graphical programming language aimed to improve coding accessibility. Age range: K-12. Link: *http://scratched.gse.harvard.edu/resources/scratch-curriculum-guide-draft*
- *Hour of Code: 20 introductory lessons to teach programming through a mix of coding and interactive activities. This curriculum is designed for K-8, but the website features age specific curriculums as well. Link: http://studio.code.org/s/1*

Outreach Programs:

- TechNights: A 10 week program for middle school girls aimed strictly at **exposure** to computer science topics. Link: http://women.cs.cmu.edu/technights/
- First Bytes: Week long residential program at University of Texas for high school girls. Focuses on **problem solving**, **team work**, and **exposure** to computer science topics. Link: https://www.cs.utexas.edu/outreach/first-bytes
- Adventures in Computing: After school outreach program for 6-8th grade students (co-gender) to teach **computational thinking skills** taught at Carnegie Mellon: http://www.women.cs.cmu.edu/What/Outreach/adventures/schedule.html
- CS for High School Students at UW: Course for new or interested teachers to learn computer science. Relevant for professional development. Link: http://cs4hs.cs.washington.edu/displayPage.php?path=./content/Overview/overview.html
- TechBridge: An organization founded in Oakland, California designed to provide various outreach programs to involve girls in STEM areas. Link: http://www.techbridgegirls.org/